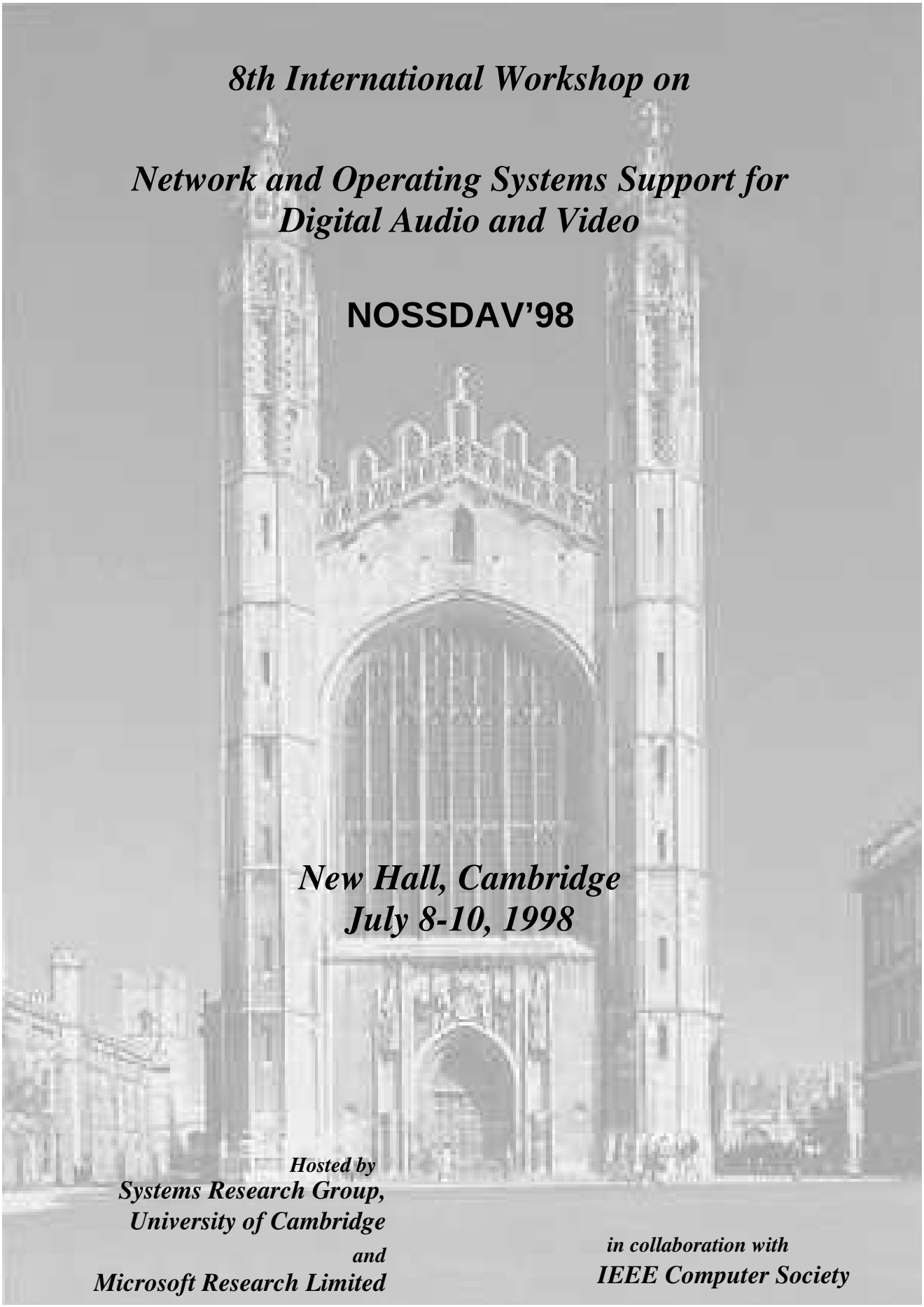


8th International Workshop on

Network and Operating Systems Support for Digital Audio and Video

NOSSDAV'98

New Hall, Cambridge
July 8-10, 1998

Hosted by
Systems Research Group,
University of Cambridge
and
Microsoft Research Limited

in collaboration with
IEEE Computer Society

IEEE COMPUTER SOCIETY

Microsoft Research

Table of Contents

NOSSDAV'98: 8^{th} International Workshop on Network and Operating System Support for Digital Audio and Video

Wednesday 8 July

Session AM 1: Network Control

Programmable Mobile Networks 1
Oguz Angin, Andrew T. Campbell, Michael E. Kounavis, and Raymond R.-F. Liao

Efficient Precomputation of Quality-of-Service Routes 15
Anees Shaikh, Jennifer Rexford and Kang G. Shin

A Soft State Protocol for Accessing Multimedia Archives 29
Angela Schuett, Suchitra Raman, Yatin Chawathe, Steven McCanne and Randy Katz

Session AM 2: IP Telephony

The TOPS Architecture for Signaling, Directory Services and Transport for Packet Telephony ... 41
N. Anerousis, R. Gopalakrishnan, C.R. Kalmanek, A.E. Kaplan, W.T. Marshall, P.P. Mishra, P.Z. Onufryk, K.K. Ramakrishnan, and C.J. Sreenan

Routing Traffic with Quality-of-Service Guarantees in Integrated Services Networks 55
Qingming Ma and Peter Steenkiste

Design and Evaluation of a Platform for Mobile Packet Telephony 71
Sandesh Goel, Partho Mishra, Huzur Saran and Cormac Sreenan

A Comparison of SIP and H.323 for Internet Telephony 83
Jonathan Rosenberg and Henning Schulzrinne

Session PM 1: Operating Systems

A new audio device driver abstraction 87
Dickon Reed

Adaptive Resource Management in a Multimedia Operating System 91
Don Oparah

On Using Intelligent Network Interface Cards to support Multimedia Applications 95
Marc E. Fiuczynski, Richard P. Martin, Tsutomu Owa, Brian N. Bershad

Supporting Continuous Media in the Piglet OS 99
Steve Muir and Jonathan Smith

Multimedia on Multiprocessors: Where's the OS when your really need it? 103
Jason Nieh and Monica S, Lam

Issues in using commodity operating systems for time-dependent tasks: experiences from a study of Windows NT 107
Michael B. Jones
An Efficient End-Host Protocol Processing Architecture for Real-Time Audio and Video Traffic 111
Khawar M. Zuberi and Kang G. Shin

Session PM 2: Panel - Operating Systems

Thursday 9 July

Session AM 3: Network Resource Allocation

LIRA: An approach for service differentiation in the Internet 115
Ion Stoica and Hui Zhang
Processor Scheduling in Routers for RSVP Message-Processing 129
Martin May, T.V. Lakshman and Anwar Elwalid
YESSIR: A Simple Reservation Mechanism for the Internet 141
Ping Pan and Henning Schulzrinne
Resource Reservation Agents in the Internet 153
Olov Schelen and Steve Pink
Achieving Quality of Service through Network Performance Management 157
S. Keshav and R. Sharma

Session AM 4: Performance Modelling/Optimisation

On the Effectiveness of Buffer in Deterministic Services 161
Pawan Goyal and Harrick M. Vin
Design, Implementation, and End-to-End Evaluation of a Measurement-Based Admission Control Algorithm for Controlled-Load Service 173
Cheng Jin and Sugih Jamin
Charging of Multimedia Flows in an Integrated Services Network 189
Burkhard Stiller, George Fankhauser and Bernhard Plattner
A Better-Than-Best-Effort Service For Continuous Media UDP Flows 193
Mark Parris, Jan Borgersen, Kevin Jeffay, F. Donelson Smith
Adapting Fair Queueing to Improve Loss, Delay, and Admission Control Performance 197
N.G. Duffield, T.V. Lakshman and D. Stiliadis

Session PM 3: Application Adaptation

An Empirical Study of Client Interactions with a Continuous-Media Courseware Server 201
Jitendra Padhye and Jim Kurose

The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation 215
Dorgham Sisalem and Henning Schulzrinne

A Toolkit for Building Migratory Continuous Media Applications 227
Tatsuo Nakajima and Hiroyuki Aizu

QoS Assurance vs. Dynamic Adaptability for Applications 231
Saleem N. Bhatti and Graham Knight

A Perspective on Application / Network Coupling 235
Thomas Gross and Peter Steenkiste

Session PM 4: Ad Hoc Panel (or punt trip)

Friday 10 July

Session AM 5: Coding/Transmission

Network Adaptive Continuous-Media Applications through Self Organised Transcoding 241
Isidor Kouvelas, Vicky Hardman and Jon Crowcroft

MPEG-2 Video Services over Packet Networks: Joint Effect of Encoding Rate and Data Loss on User-Oriented QoS 257
Olivier Verscheure, Pascal Frossard and Maher Hamdi

Issues of Quality and Multiplexing when Smoothing Rate Adaptive Video 265
N.G. Duffield, K. K. Ramakrishnan and Amy R. Reibman

Real-Time Reliable Multicast Using Proactive Forward Error Correction 279
Dan Rubenstein, Jim Kurose, and Don Towsley

Session AM 6: Video File Systems

Cost-Optimal Data Retrieval for Video Servers with Variable Bit Rate Streams 295
Ernst Biersack and Maher Hamdi

Adaptive Quality-of-Service Session Management for Multimedia Servers 303
John Reumann and Kang Shin

Efficient Schemes for Broadcasting Popular Videos 317
Lixin Gao, Jim Kurose and Don Towsley

Open Programmable Mobile Networks

Oguz Angin, Andrew T. Campbell, Michael E. Kounavis, and Raymond R.-F. Liao

COMET Group, Center for Telecommunications Research,
Columbia University, New York NY 10027
http://comet.columbia.edu/wireless

Abstract

We present the design, implementation and evaluation of an open programmable mobile network based on distributed object technology called mobiware that dynamically exploits the intrinsic scalable properties of adaptive mobile applications. While a number of adaptive mobile systems have been proposed in the literature few experimental systems exist today to assess the viability of the adaptive approach. We argue that existing mobile systems (e.g., mobile IP, mobile ATM and third generation cellular systems) lack the intrinsic architectural flexibility to deal with the complexity of supporting adaptive mobile applications in wireless and mobile environments. We believe that there is a need to develop alternative network architectures from the existing ones to deal with the tremendous demands placed on underlying mobile signaling, adaptation management and wireless transport middleware. Mobiware represents one example of such an alternative approach. Software-intensive (comet.columbia.edu/wireless/software/mobiware), mobiware provides an open programmable approach to dealing with the complexity of supporting adaptive mobile applications in time-varying mobile and wireless networks.

1. Introduction

The phenomenal growth in cellular telephony over the past several years has demonstrated the value people place on mobile voice communications. The goal of next-generation wireless systems is to enable mobile users to access, manipulate and distribute voice, video and data anywhere anytime. As the demand for mobile multimedia services grows, high-speed wireless extensions to existing broadband and Internet technologies will be required to support the seamless delivery of voice, video and data to mobile terminals with sustained high quality. New wireless services will include Internet access to interactive multimedia, video conferencing and real-time data as well as traditional services such as voice, email and web access. The wireless and mobile environment presents a number of technical challenges to this vision. First, physical layer impairments contribute toward time-varying error characteristics and time-varying channel capacity as observed by mobile devices. We describe the quality index maintained across the wireless channel as *wireless-QOS* as illustrated in Figure 3. Second, user mobility can trigger rapid degradation in the quality of the delivered signal. This can lead to handoff dropping in broadband cellular networks and loss of service in mobile ad hoc networks. As a result, mobile applications can experience unwarranted delays, packet losses or loss of service. We describe the quality index maintained during mobility as *mobile-QOS* as illustrated in Figure 3. There is growing consensus that adaptive techniques [Katz,94], [Lee,95] [Lu,97] and [Naghshineh,97] present the only viable approach to countering time varying quality of service impairments in wireless and mobile network environments. However, providing system-wide (i.e., end-system and network) adaptive quality of service support is complex to realize and not well understood [Campbell,97b].

To address these challenges, we have built an open [Lazar,97] and active [Tennenhouse,97] programmable mobile network [Campbell,96] that is controlled by a software middleware platform called *mobiware* [Mobiware,98]. Mobiware extends earlier work by the COMET group on the xbind broadband kernel [xbind,96] to the mobile and wireless domain. By open, we mean that there is a need to "open up" hardware devices (e.g., mobile devices, access points and mobile-capable switches and routers) for implementation of new mobile signaling, transport and adaptive quality of service management algorithms. At the lowest level of programmability, mobiware abstracts hardware devices and represents them as distributed computing objects. These objects (e.g., an access point object) can be programmed via a set of open programmable network interfaces to support adaptive quality of service assurances. By programmable, we mean that these programmable network interfaces are high-level enough to allow new services to be built using distributed object

computing technology, e.g., CORBA, DCOM and Java. By active, we mean that adaptive quality of service algorithms can be represented as active transport objects and injected on-the-fly into mobile devices, access points and mobile capable network switches/routers to provide value-added quality of service support when and where needed.

In this paper, we present an overview of mobiware followed by the design, implementation and evaluation of a programmable mobile network. The structure of the paper is as follows. Section 2 describes an adaptive-QOS API and service model, the mobiware architecture and the network model. Following this, Section 3 presents the design and implementation details of mobiware's programmable mobile network. This is followed in Section 4 by an evaluation of the system in an experimental setting. Finally, in Section 5 we present some concluding remarks.

2. Mobiware

Mobile applications need to be capable of responding to time-varying wireless-QOS and mobile-QOS conditions. Wireless transport systems should be capable of manipulating content in response to fluctuating quality of service conditions. Medium access controllers must be capable of supporting adaptive quality of service assurances to the mobile devices. Mobiware is well suited toward managing the evolving service demands of adaptive mobile applications and dealing with the inherent complexity of delivering audio, video and real-time services to mobile devices. Based on an adaptive quality of service API, mobiware consists of a set of controllers that interact with transport, network and medium access controller distributed objects that maintain application-specific adaptive quality of service needs.

2.1 The Adaptive-QOS API and Service Model

By trading off temporal and spatial quality with available bandwidth, mobile applications can be made to adapt to time varying conditions with minimal perceptual distortion. In [Binachi,98a], we introduced an *adaptive-QOS API* and service model specifically designed to quantitatively address the wireless-QOS and mobile-QOS needs of adaptive mobile applications. Mobile applications use an adaptive-QOS API at the transport layer specifying:

- *a utility function*, which captures the adaptive nature over which an application can successfully adapt to available bandwidth in terms of a utility curve that represents the range of observed quality to bandwidth. The observed quality index refers to the level of satisfaction perceived by an application at any moment as illustrated in Figure 1; and
- *an adaptation policy*, which captures the adaptive nature of mobile applications in terms of a set of adaptation policies (viz. fast, slow, after handoff, never). These policies allow the application to control how it moves along its utility curve as resource availability fluctuates, e.g., scale-up only after handoff, fast adaptation and slow adaptation.

The utility function allows *utility-fair* bandwidth allocation algorithms [Bianch,98a] to derive explicit optimization rules under heterogeneous application adaptation behavior. Here bandwidth is allocated fairly to all the flows so that the same utility value is achieved at an access point. For full details of the utility-fair bandwidth allocation algorithm see [Bianchi,98a]. The utility function alone, however, is not capable of capturing application specific adaptation dynamics. Rather, a simple set of adaptation policies is used to capture how an application wishes to respond to instantaneous bandwidth availability.

A mobile multimedia application's range of perceptible quality is strongly related to how and when it responds to resource changes. Frequent oscillation between what may be considered optimal and minimum utility or even the frequent small change around an average application quality may be annoying to many applications. Some applications may wish to limit the frequency of adaptation to change, e.g., multi-resolution application. In contrast, others may wish to exploit any opportunity for adaptation, e.g., real-time data application. By limiting or dampening the response to change an application attempts to follow trends in resource availability rather than fluctuations to instantaneous changes. Such a conservative adaptation policy may lead to a more stable operating point on an application's utility curve. This is in contrast to a policy that responds to instantaneous fast moving points that may suit other styles of mobile application. The API allows the application to specify temporal or event-based dimension to utility.

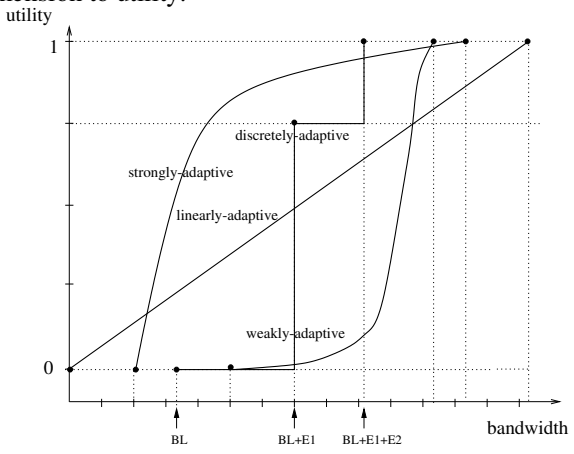


Figure 1: Utility functions

The service model supports the following adaptation 'menu' policy options[1]:

- *fast*, to instantly move up or down the utility curve responding instantly to any resources changes;
- *slow*, to move up or down the utility curve only after a suitable damping period has passed;
- *handoff*, to move up or down the utility curve only after handoff; and
- *never*, to never move up the utility curve after the initial bandwidth allocation.

The adaptive-QOS API is supported by mobiware at the transport and realized at the mobile device and in the network. This allows the mobile application to specify a flow's utility function and adaptation policy, which is supported by mobiware.

2.2 The Architecture

Mobiware is a software-intensive mobile networking environment based on distributed object technology. As illustrated in Figure 2 mobiware promotes the separation between mobile signaling and adaptation management, and the transport of media. At the transport layer, an *active wireless transport* supports the end-to-end transmission of audio, video and real-time data services based on an adaptive-QOS paradigm. The active wireless transport is an object-based transport that blurs the region over which traditional transports (e.g., TCP and RTP) typically operate to include access points and end systems.. Built on a set of Java classes, the transport system binds active and static transport objects at mobile devices and access points to provide end-to-end transport adaptation services. Static transport objects include segmentation and reassemble, rate control, flow control, playout control, resource control and buffer management objects. These objects are loaded into the mobile device as part of the transport service creation process. Active transport objects are dynamically dispatched to mobile devices and access points to support valued-added QOS. Currently, two styles of active transport objects have been implemented: active media filters [Balachandran,97], which perform temporal and spatial scaling for multi-resolution video and audio flows and adaptive FEC filters [Campbell,97a] which protect content against physical radio link impairments by matching the level of Reed Solomon channel coding to match time-varying error characteristics.

1 A generalization of this approach is detailed in [Bianchi,98a]. Adaptive mobile applications supply adaptation handlers, which implement application-specific adaptation policies supporting more sophisticated levels of adaptation than the current menu options (e.g., fast, handoff) offered in the existing system. Adaptation handlers can program simple or sophisticated adaptation strategies. Mobiware exposes a set of low level programming APIs to allow the application to control its adaptation strategy.

At the network layer, a *programmable mobile network* supports the introduction of new mobile adaptive-QOS services based on the xbind broadband kernel [xbind,96]. The network layer supports switching IP flows over ATM native transport services. Architecturally, the network comprises a set of network objects and adaptation proxies that operate at the mobile device, access points and at mobile capable switch/routers. Currently, an adaptive-QOS network service supports the delivery of multi-resolution flows having a base layer and one or more enhancement layers. The base layer provides a foundation for better resolutions to be delivered through the reception of enhancement layers based on the availability of resources in the wireless environment. Three key network algorithms include: (i) *QOS controlled handoff*, which gracefully scales flows (up and down) based on the semantics of the adaptive-QOS service during handoff when bandwidth availability varies; (ii) *mobile soft-state*, which provides mobile devices with the capability to respond to changes in wireless-QOS and mobile-QOS; and (iii) *flow bundling*, which exploits a common routing representation for all the flows to and from a mobile device to speed up handoff. *The focus of this paper is the design and evaluation of the programmable mobile network described in Sections 3 and 4, respectively.*

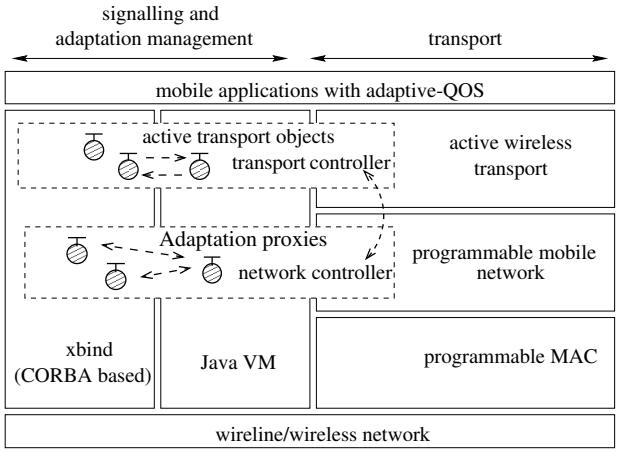


Figure 2: The mobiware architecture

At the data link layer, a *programmable MAC* [Bianchi,98b] combines a set of foundation services to support more sophisticated adaptive wireless-QOS services. Foundation services provide sustained rate services used to support minimum wireless-QOS assurances, and active and passive adaptive services to support application specific adaptation policy. The 'programmable' nature of the data link service provisioning over wireless networks provides an alternative approach to that found in the literature. Rather than supporting a specified set of 'hard wired' MAC services (e.g., VBR) by means of centralized control

schemes, it provides a programmable air-interface that allows new services to be dynamically created and installed on the fly. This programmable MAC service support relies on a simple core architecture that pushes complexity and application specific adaptation decision making to the mobile device. For full details on the programmable MAC see [Bianchi,98b]

2.3 The Network Model

Mobiware is a novel software middleware platform that controls an experimental broadband wireless access network to a next generation mobile Internet [Mobicom,97] called *mobinet*. The network model comprises a set of mobile devices, wireless access points and mobile-capable switches/routers providing broadband cellular and ad hoc communication services to mobile users. Mobinet is based on ATM switching technology that supports IP switched flows in the access network. Mobile devices can be connected to mobinet via broadband cellular or ad hoc wireless access modes. In broadband cellular mode, mobile devices receive core network services via a set of wireless access points. Ad hoc devices may operate autonomously without the aid of any fixed infrastructure and core network services or can connect to the broadband cellular network via multiple ad hoc hops as illustrated in Figure 3.

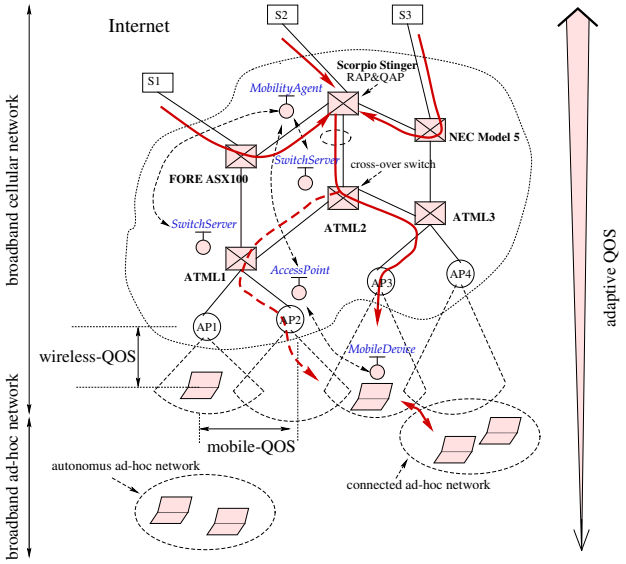


Figure 3: mobinet

Providing quality of service assurances in the broadband cellular networks is difficult. However, providing quality of service assurances without the aid of any fixed infrastructure as in the case of ad hoc is more challenging [Corson,98]. We believe there is a need understand the level of quality of service that can be supported at different points of attachment in mobinet, e.g., at the access point or multiple hops away from the access point. We observe that quality of service assurances are likely to diminish as a mobile device moves away from the core network as illustrated in Figure 3. Providing seamless quality of service support to mobile devices on the move (e.g., switching between broadband cellular and ad hoc modes) underpins mobiware's adaptive-QOS design approach[2].

3. Programmable Mobile Network

3.1 Programmable Objects

The mobile network comprises a set of programmable distributed CORBA objects[3] that support the delivery of adaptive-QOS flows to mobile devices over mobinet. The use of distributed object technology also provides support for interoperability between mobile devices utilizing different operating systems and protocol support. Mobiware objects execute on mobile devices, access points and mobile capable switch/routers supporting a set of mobile signaling and quality of service adaptation algorithms (viz. QOS controlled handoff, flow bundling and mobile soft state) as illustrated in Figure 3. Objects combine data structure (defining the object's state) with a set of methods (defining the object's behavior). Methods are executable programs associated with objects that operate on information in an object's data structure.

Per-mobile adaptation proxies support the communications of flows to/from mobile devices and are used to support fast and scalable handoff. These include *Routing Anchor Proxies (RAP)* objects, which 'bundle' (i.e., aggregate) flows to and from mobile devices for fast handoff [Campbell,97b] and *QOS Adaptation Proxies (QAP)* objects, which are used as an integral part of the mobile soft-state probing mechanism for distributed resource allocation over the wireless network.

To manage the network states introduced by flow-oriented communications and, more importantly, to gain efficiency across a wireless link, mobiware deploys a number of network objects that can execute on network nodes or on servers at the edge of the network. The mobile device object abstracts the operation of mobile stations and provides APIs for querying beaconing information, registration with an access point, flow establishment and handoff. It also includes the functionality to dynamically control the transport system at the access points, e.g., to set the media scaling or error

2 In this paper we focus on adaptive quality of service support for broadband cellular communications leaving ad hoc support for future work [Coroson,98].

3 Mobiware includes active transport objects based on Java classes that 'plugin' to the data path at wireless access points to provide value-added adaptation support. Currently, the active wireless transport supports active media scaling and adaptive FEC support.

control level for video flows. The mobile device object states mainly consist of quality of service specification (viz. utility function and adaptation policy) for all the flows directed towards/from the mobile station and end-point information for all the network nodes between the source-destination pairs.

```
interface AccessPoint : NodeServer {

  // flow setup from the current access point to
  // the network, called by the mobile device object
  void setupFlow(in long fbi,
      inout QOSSpecification qosSpec,
      inout FlowInfo flowinfo, in string<40> srcname,
      inout EndPoint peerEp,
      inout EndPoint RAP_fix, inout EndPoint RAP_mobile,
      inout EndPoint AP_fix, inout EndPoint AP_mobile,
      inout EndPointId airIP, inout double msr_time)
    raises(Reject);

  // handoff flow bundle for a specific mobile device
  void handoffFlowBundle(in long fbi,  inout QOSSpecList
      qosSpec[2],  inout FlowInfoList flowinfo[2],
      inout SourceList srcname[2], inout EndPointList
      RAP_fix[2], inout EndPointList RAP_mobile[2],
      inout EndPointList AP_fix[2], inout EndPointList
      AP_mobile[2], inout EndPointIDList airIP[2],
      inout double msr_time)
    raises(Reject);

  // refresh mobile soft-state for flow bundle
  // through the current access point to the network
  void refreshFlowBundle(in long fbi,
    inout EndPointList RAP_mobile[2],
    inout EndPointList AP_fix[2],
    inout EndPointList AP_mobile[2],
    inout double ntw_msr_time)
   raises(Reject);
};
```

Figure 4: CORBA IDL for access point object

An access point object supports APIs for binding to wireline network objects on behalf of mobile stations, propagating CORBA calls and for the establishment and periodic refreshing of local wireless connections for flows as illustrated in Figure 4. This object plays an important role in QOS controlled handoff and interacts with the transport system for the injection of active transport objects when and where needed. The mobility agent object provides flow-management, handoff signaling and mobility management services. It interacts with per-mobile RAP or QAP state in the switch servers and supports APIs for retrieving network topology information from a router object (e.g., location of the cross over switch) and communicating with each network node separately. A router object stores end-to-end route information.

Switch server objects [xbind,96] abstract and represent physical ATM switch/routers and are fully quality of service programmable. These objects support APIs for the reservation and release of namespace (viz. VCI/VPI pairs) and the allocation of network resources (viz. bandwidth). State mainly consists of per-flow connection information, stored in local hash-tables called switch caches. Switch server objects have been extended to be mobile capable, i.e., support RAP and QAP functionality. The General Switch Management Protocol (GSMP) protocol is used at the access points and switch/routers for accessing the switch tables.

3.2 QOS-controlled handoff

QOS controlled handoff gracefully scales flows (up and down) during handoff based on the semantics of the adaptive-QOS API described in Section 2.1. By scaling flows during periods of resource contention (e.g., during handoff), mobiware improves the wireless resource utilization and reduces the handoff dropping probability. While the style of handoff is entirely programmable, the current implementation style is mobile-initiated, forward handoff with soft-handoff on the down-link and hard-handoff on the uplink. By mobile-initiated, we mean that after a suitable dwell time the mobile device initiates a handoff by first registering with the forward/new access point. By soft-handoff, we mean that during handoff the mobile device simultaneously receives flows from the old and new access point on the down-link. In contrast, uplink flows use a 'break and make' approach between the old and new access points. During handoff, registration to the new access point, rerouting of flows and quality of service adaptation is accomplished by signaling objects and associated APIs outlined in Section 3.1. Signaling APIs are programmable[4] allowing various styles of handoff to be tailored toward particular radio environments.

QOS controlled handoff object-interaction is illustrated in Figure 5. Mobile device objects periodically 'hunt' for beacon signals from neighboring access points. Beacons are fully programmable in mobiware and carry low-level signal information in addition to the current bandwidth availability at the sending access point. The mobile devices' hunting algorithm periodically compares all beacons received over the current hunt period and cumulatively over multiple hunt periods. If the wireless-QOS[5] indicated in the beacon from the current access point falls below a pre-determined threshold the hunt algorithm selects a new access point for handoff. Handoff is initiated after a suitable dwell period after which the mobile device registers with the new access point starting the handoff process as indicated by (2) (3) in Figure 5.

[4] Note that the object oriented aspect of mobiware signaling makes it easy to rapidly 'program' different styles of handoff algorithm, e.g., the object APIs can support network initiated handoff that are hard in nature.

[5] The beacon informs the mobile device of the channel conditions upon the reception of each packet arrival reporting the signal level, silence level signal quality and antenna selected. The signal and silence level area derived from the receiver's automatic gain control settings. Beacon messages are augmented with a 16-bit field that indicates the available resources at the access point. The mobile device can use this to scale down request for bandwidth resources during handoff given that the bottleneck node is typically the access point. Our radios are based on WaveLAN operating in the 2.4-2.8 GHz ISM band, which we have low level access to for programming the beacon.

The device registration procedure triggers the new access point object to bind to a mobility agent object (4). The mobility agent caches bindings to the per-mobile adaptation proxies that are implemented as part of switch servers. Mobility agents and proxies can run anywhere in the mobinet, i.e., mobility agents can operate at fixed edge device, mobile device or on the switches. Mobility agents are the main controllers for managing handoff in mobiware. Currently, mobiware uses a single mobility agent to manage handoff for the complete testbed. Mobility management is a fully distributed algorithm that includes one or more mobility agents for scalability. When the mobile device initiates a handoff (5) it passes a unique mobile device identifier called the flow bundle identifier (FBI) to the access point that allows mobiware to identify the mobile device's flow bundle in the wireless access network.

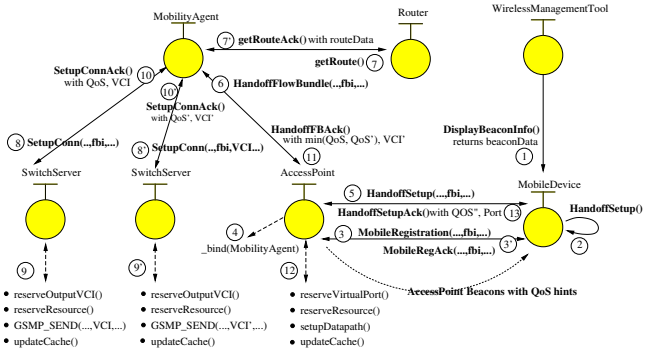


Figure 5: QOS-controlled handoff object interaction

Mobility agents are responsible for re-routing a mobile device's flow bundle from an old access point to a new one as illustrated in Figure 3. This entails the mobility agent invoking the route object to determine the location of the cross over switch as illustrated in Figure 3. Switch server objects are used to re-establish new flow state at all switches between the cross over switch and the new access point. The re-routing phase includes name space reservation (viz. outgoing VCI/VPI) and bandwidth value at each network switch and the new access point. The final process of re-routing a flow bundle through a switch includes the use of GSMP (9) (9') (12) to set up the switch table and reserve resources. However, GSMP does not support the concept of flow bundling. While the mobile agent informs the switch server objects to establish state for a complete flow bundle, the switch server interacts with GSMP on a flow by flow basis. In the evaluation section we describe enhancements to GSMP to support flow bundling. The mobility agents interact with mobile capable switch servers and the new access point in parallel resulting in a speed up of the re-routing phase of the handoff algorithm over conventional hop-by-hop signaling. After the re-routing of the flow bundle is complete the mobile agent informs the new access point of the negotiated quality of service and flow bundle VCI/VPI mappings. The new access point interacts with the active wireless transport to provide active media filters based on the available bandwidth at the interface.

To keep the name space binding between the mobile device and access points constant with mobility we have implemented the notion of virtual wireless ports. As mobile devices connect to different access points their VPI/VCIs mapping remain constant. The flow bundle to VPI/VCI bundle is resolved by a virtual wireless port, which is dynamically allocated by the new access point during handoff. This approach minimizes the impact of re-negotiation in comparison to a full name space re-negotiation, which would be disruptive during handoff.

3.3 Flow bundling

QOS controlled handoff and mobile soft state exploit flow bundling to speed up handoff and minimize the signaling overhead associated with maintaining the network state. Flow bundling [Porter,95] provides a common routing representation for all the flows to and from a mobile device as illustrated in Figure 3. This is similar to the virtual path concept in ATM networks or tunneling in IP networks. Flow bundling is a general method for encapsulating and routing. For example, flow bundles can carry both VPs and VCs in mobiware. Flow bundling is motivated by the need to reduce the complexity of re-routing multiple independent flows to and from mobile devices during handoff. By aggregating flows in this manner we can speed up handoff, simplify mobile soft-state probing and minimize signaling overhead. Using flow bundling, QOS controlled handoff simply discovers a single crossover switch then re-routes all flows to the new access point in a single object-level operation.

During handoff the flow bundle object interaction is as follows. The mobile device object invokes the access point's handoffSetup() method once for the flow bundle minimizing the signaling overhead at the air-interface. Mobiware supports the option of enabling or disabling bundling when flows are established. Note that when flow bundling is disabled the mobile device, access point, mobility agent objects treat each flow independently during handoff. As discussed in the evaluation this increases the signaling overhead and the handoff latency. During each invocation a separate cross over switch needs to be located, using a shortest path algorithm and individual flows need to be re-routed and signaled independently. The mobile agent interacts with the switch servers to re-establish flows and update switch tables for all switches between the cross over switch and the mobile devices. GSMP is used to update the switch table at each traversed switch. In order to support the atomic re-routing of flow bundles we have enhanced the GSMP protocol.

Flow aggregation at the switch control level has been implemented as a modification to the GSMP invocation mechanism. Two enhancements have been considered in Section 4. The first has no impact the current GSMP client server interaction. The mobile agent simply invokes the GSMP client for each flow without waiting for acknowledgement from each GSMP command. This results in the switch server sending a burst of GSMP setup messages to the switch then waiting for acknowledgements. The second enhancement augments the GSMP setup message to allow up to 256 VCI pairs to be passed across the interface in one client-server interaction. This allows the switch server to setup the switch table for a flow bundle in one operation. We discuss the performance benefits of flow bundling in Section 4.

3.4 Mobile soft-state

During handoff a flow bundle must be re-routed to a new access point, resources need to be reserved, and the old flow bundle state between the old access point and the cross over switch removed. Mobile devices resident in cells also need to scale flows in accordance with channel conditions, whether new flows are established or released, and when new mobile devices enter and leave cells. Mobile soft-state provides quality of service adaptation support to cater to a number of these conditions. Mobile soft-state results in the periodic establishment of bandwidth and name space resources for flow bundles between a mobile device and a per-mobile QOS adaptation proxy (QAP). Mobiware supports the idea of soft-state [Campbell,97] in the mobinet to refresh the network state. Periodically refresh messages sent by mobile devices are on a flow bundle basis not a per-flow basis. During the refresh phase mobile devices respond to any changes in allocated bandwidth (based on utility functions) to the flow bundle.

In [Campbell,97] we argue that a soft-state approach is well suited to supporting QOS adaptation in mobile networks. Mobile soft state supports a distributed probing mechanism based on flow bundles allowing mobile devices to compete fairly for bandwidth in a completely decentralized and scalable manner. During handoff mobile devices do not explicitly remove the old flow bundle-state between the old access point and the cross over switch. In this case, mobile soft-state timers located at the switches and old access point timeout and release resources automatically. Mobile devices resident at the old access point compete for these available resources thereby improving their utility.

Mobile devices periodically probe the path between the mobile device and the per-mobile QAP and contend for resources. Note, that per-mobile QAPs can be located at an access point or any mobile switch/router between the mobile and its corresponding routing anchor proxy (RAP). If the QAP is located at the access point then mobile soft state is only active over the air-interface; that is, between the mobile device and access point. The position and configuration of where these proxies reside is programmable. In many cases, the access point is the most suitable location because radio resources are generally the bottleneck in broadband wireless or wireless LAN systems.

Mobile devices independently probe the wireless access network. The probe includes a list of flow requirements for the complete flow bundle, which includes the utility function for each flow. In the current system, we have implemented the discretely-adaptive flows (see Figure 1) where a base layer (BL) and one or more enhancement layers (viz. E1, E2) would be signaled in the probe message. The probe interacts with the access point and all mobile capable switches/routers between the mobile device and its QAP. The response is returned to the mobile device indicating the available bandwidth reserved for each flow in the bundle over the next time interval. This time interval is called the refresh interval. If a probe message is received along the path before the refresh interval expires then the reservation is 'refreshed' based on available resources. In contrast, if no refresh message is received then the timer expires and the resources deallocated and state removed. This reservation-adaptation style probing and response is implemented at the object level as a set of refreshFlowBundle() method invocations (2) (3) (3'). The mobile device issues a refresh to the mobility agent object which then refreshes all switches and the current access point on the path between the mobile and QAP (see Figure 6).

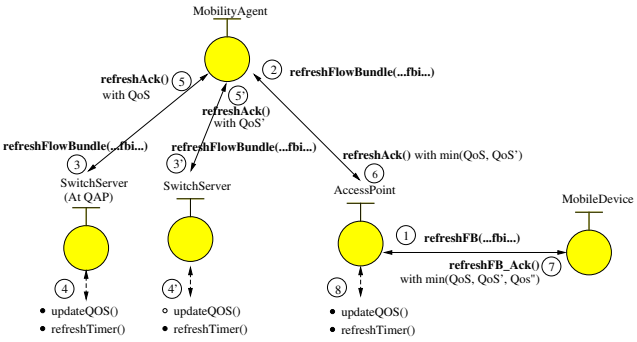


Figure 6: Mobile soft-state object interaction

4 Evaluation

To evaluate the programmable mobile network performance, we built the mobinet testbed, developed a wireless network management tool and designed a set of experiments to analyze QOS controlled handoff, flow bundling, mobile soft-state and QOS adaptation algorithms. Through the course of measurement,

evaluation and re-design we managed to substantially improve upon our initial baseline design.

4.1 The Experimental Environment

At the network level, mobinet provides wireless access to the Internet and comprises four ATM switches[6] (viz. ATML Virata, Fore ASX/100, NEC model 5, Scorpio Stinger switches) and four wireless access points as illustrated in Figure 3. The access points run the mobiware code and are based on a set of multi-homed 200 MHz pentium PCs that provide radio access to a wireline switched IP/ATM sub-network. High performance notebooks (viz. IBM, Gateway and NEC) provide support to mobile applications and mobile access to the access network. Wireline ATM links operate OC-3 rates between the switches and fixed end-systems, and at 25 Mbps between the switches and access points. Currently, the air-interface is based on WaveLAN operating at 2 Mbps in the 2.5 GHz band. A future version of the mobinet will operate at 25 Mbps spectrum in the 5GHz NII/SUPERNet unlicensed band. Mobile capable switches, access points and mobile devices are abstracted as programmable CORBA objects. Mobiware requires IIOP CORBA for mobile signaling and adaptation management. For the results provided in this paper the mobile devices and access points use IONA's Orbix CORBA running under Windows NT, and the mobile capable switches/routers use IONA's Orbix running under UNIX or proprietary operating systems.

We have designed and implemented *mobiman* a Java-based management tool for wireless networks. Mobiman drives experiments, displays recorded statistics, and provides network management capability to view the network and inspect the state of any mobile device. To measure the performance of the programmable mobile network we inserted measurement checkpoints throughout the code and recorded performance statistics during the experiments. Mobiman, which runs on any fixed or mobile device, can remotely target any mobile device operating in mobinet displaying mobile device object statistics. It can setup flows, turn on/off flow bundling and mobile soft state, interact with media scaling and adaptive FEC control at the transport level, and force handoff operations to occur. Measured information displayed by mobiman comprises flow information, quality of services measurements (e.g., signal level, silence level signal quality, and access point bandwidth availability) and experimental checkpoint measurements. Mobiman displays measured information, wireless network topology and mobile device location in a control window as illustrated in Figure 7. When a mobile device is selected by mobiman a control window indicates the state of the mobile, e.g., three flows are delivered to 'mobile-air1'. In this example, the mobile-air 1 is running mobiman and the three flows correspond to the playout of the "True Lies" video clip and two low-resolution text windows. A flow setup panel appears in the top-left corner of Figure 7.

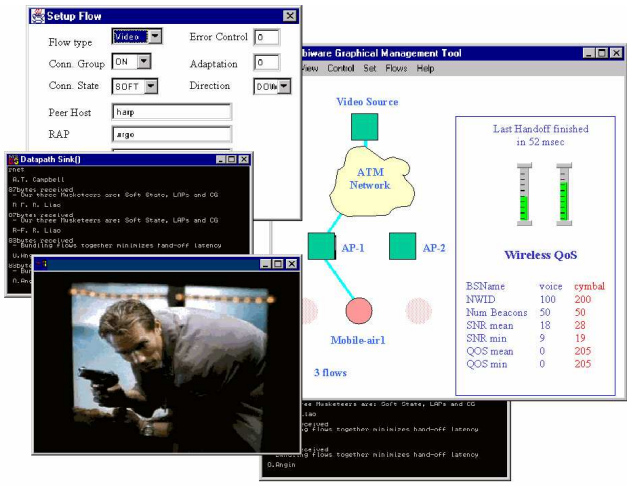


Figure 7: mobiman: a Java-based management tool for wireless networks

Microsoft's Active Movie is used for the reception, decoding and rendering of digital video. It provides a software tool capable of controlling and processing streams of multimedia data. Active movie uses modular components called filters and filter graphs. Typically, a filter graph consists of a source filter that provides the system with multimedia data, a transform filter that performs data decompression and a rendering filter. Active Movie's filter graph has been enhanced with an appropriate mobiware static transport object to perform synchronization of flows during handoff, controls delay-jitter control and rate control.

4.2 The Experiments

Our evaluation methodology is based on a set of experiments designed to investigate the performance of the programmable mobile network in supporting mobile multimedia communications. The use of CORBA for mobile signaling, wireless adaptation and mobile network programmability is a novel aspect of our work. CORBA objects run at the edges and in the mobile network to

6 We modify the concept of switchlets [Merwe,97] to provide an extended network for mobiware evaluation. Switchlets allows multiple virtual network elements to be operational within the same physical nodes. Three ATM switches (ATML 1, 2, and 3 as shown in Figure 3) in our network are switchlets physically co-located at the same physical switch. For example, packets traversing three switchets located at the one physical switch travel across the physical switch three times via cables that directly connect one port to another in the same switch. Each switchlet corresponds to a different CORBA switch server object with different name space, and manages its own resources and controls connections independently of others.

support wireless-QOS and mobile-QOS. An important aspect of our evaluation was to determine if such distributed object technology was viable in supporting mobile signaling and adaptation management.

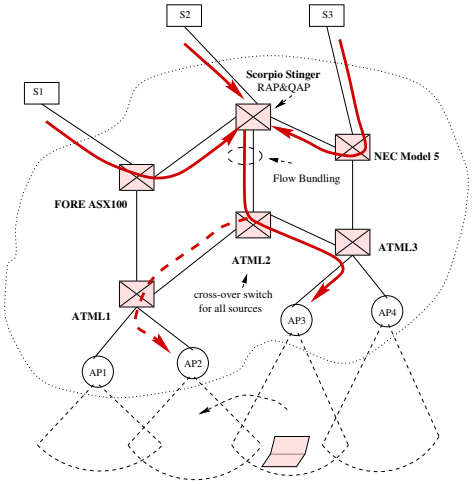


Figure 8a: Handoff with flow bundling ON

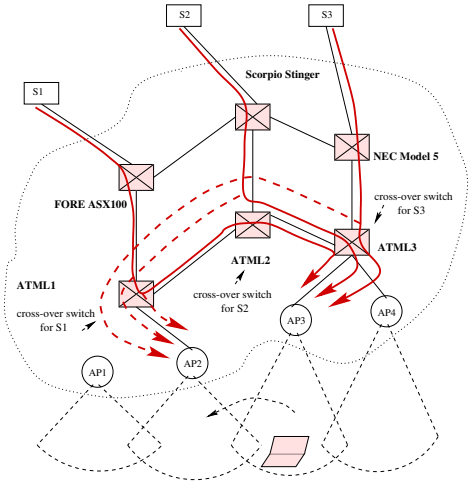


Figure 8b: Handoff with flow bundling OFF

4.2.1 *Handoff Analysis*

An important objective of this experiment is to measure the handoff latency and understand how the signaling system delays breakdown. In this experiment we investigated the handoff of a single flow. Handoff with flow bundling is described in the next section. For this experiment we streamed a single video flow from a fixed network server (S1) to a mobile device (M1). The mobile device moved repeatedly between access points AP2 and AP3 with the crossover switch located at the ATML2 switch. The average handoff latency for the baseline code is 171 msec. This figure broke down into 102 msec for mobile registration and object binding, 30 msec for wireless ATM connection setup and 35 msec for wireline connection setup. The greatest portion of the total latency time being absorbed by the binding process between objects during handoff. As described in Section 3.2, the mobile device object remotely binds to the access point object at the forward access point. Following this, the access point locally binds to a QOS mapper object and remotely binds to a mobility agent object for handoff control.

The following enhancements were made to the baseline code to reduce binding and Remote Procedure Call (RPC) overhead. First, by collapsing unnecessarily independent CORBA objects into a single object the binding overhead was reduced. To reduce binding over the air-interface the mobile proxy and QOS mapper in the access point object was collapsed. This reduced the number of CORBA requests across the air-interface reducing the binding time to from 102 msec to 42 msec. Collapsing objects in this manner reduced the handoff latency to 111 msec as illustrated in Figure 9. Next, by reducing the number of CORBA RPCs the overhead between objects during handoff we reduced. An RPC across the air-interface between the mobile and access point took an average of 15 msec to complete. The number of RPCs between the mobile and access point was reduced from four to two (viz. registration, handoff request). Reducing the number of CORBA RPCs during handoff reduced the handoff latency by 28 msec to 93 msec as illustrated in Figure 9.

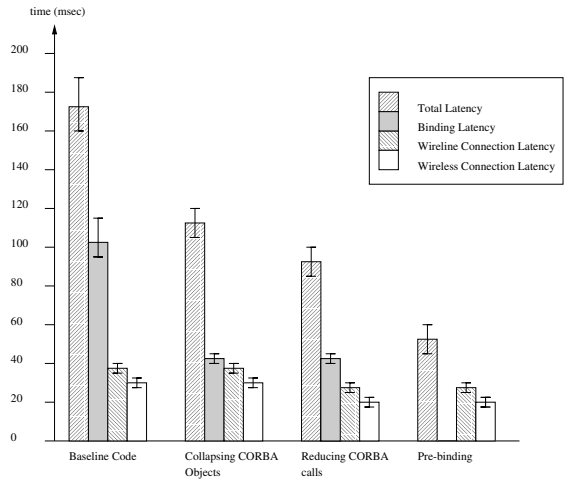


Figure 9: Handoff latency measurement results

The final enhancement to the baseline handoff code exploited the concept of caching object bindings. In order to eliminate the binding latency, we setup and cached bindings between remote CORBA objects prior to handoff being initiated which we call this *pre-binding*. All access points periodically broadcast their beacon including address information, signal strength and

available resources. When mobile devices receive these beacons in promiscuous mode they register the signal quality in lieu of a possible handoff to a new access point. A pre-bind capability was to be added to the programmable mobile network to allow mobile devices to pre-bind to neighboring access points in advance of handoff. The pre-binding criterion is based on the signal strength and available resources. The pre-binding algorithm issues a pre-bind to an access point object on-the-fly establishing TCP connections for the CORBA IIOP between the mobile device and the access points. Another enhancement establishes binding between all access point objects in a domain and its associated mobile agent object. This final enhancement reduced the average handoff latency from 171 msec to 51 msec.

4.2.2 *Flow Bundling Analysis*

This experiment evaluates the performance gains using flow bundling during handoff. We observe the performance of handing off multiple flows with flow bundling disabled and then enabled. In the experiment video is streamed from three independent sources (viz. S1, S2, S3) across the network to a single mobile device, which is repeatedly moving between access points AP2 and AP3. When flow bundling is disabled each flow S1, S2 and S3 is independently re-routed during handoff via the ATML1, ATML2 and ATML3 crossover switches, respectively. When flow bundling is enabled all flows are bundled at the per-mobile QAP located at the Scorpio switch and re-routed during handoff via a single cross over switch located at the ATML2 switch as illustrated in Figure 8a.

In this experiment, we vary the number of video streams from one to ten flows with bundling enabled and disabled and measure the handoff latency. The results from the baseline measurement highlight the speed up achieved using flow bundling in the access network as illustrated in Figure 10. For two flows with and without flow bundling enabled the handoff latency is 200 msec and 250 msec, respectively. As the number of flows increase the benefit of bundling becomes more pronounced. For ten flows with and without flow bundling enabled the handoff latency is 320 msec and 780 msec, respectively. The adoption of flow bundling provides an improvement of 20% for two flows and 59% for ten flows. With flow bundling enabled (as illustrated in Figure 8a) the handoff latency converges whereas with flow bundling disabled, the latency increases almost linearly as the number of flows increases. These results indicate the benefit of using flow bundling to reduce handoff latency and signaling overhead. This is mainly due to the fact that all interactions between objects during handoff deals with aggregated signaling rather than per-flow signaling.

The baseline code was enhanced to support the optimization described in section 4.2. In addition, the GSMP messaging between the switch server and GSMP provided some incremental improvements. The handoff latency was reduced to 56 msec with bundling and to 67 msec without bundling for two flows showing a 16% improvement. In contrast, the handoff latency for ten flows was 155 msec and 420 msec with and without bundling, respectively showing a 63% improvement.

The baseline code only provides flow bundling support between the mobile device, access point and mobility agent objects. The interface between the mobility agent and the switch server are per-flow. Another observation is that the GSMP interface between the switch server object and switch does not provide any support for aggregation, i.e., GSMP client cannot update the switch table for more that one VCI pair. To address this we enhanced the GSMP interfaces used by the switch server object. This resulted in seamless support for flow bundling aggregation from the mobile device to the switch tables providing some level of speed up as illustrated in Figure 10. The GSMP enhancements described in Section 3.3 include a "parallel" enhancement, which did not require any changes to the GSMP code. In this case, for two flows the latency for total GSMP messages is 849 μsec without aggregation, and 511 μsec with aggregation showing a 40% improvement. With increasing number of flows, the total gain obtained by aggregation increases up to 70% for ten flows (3907 μsec vs. 1184 μsec).

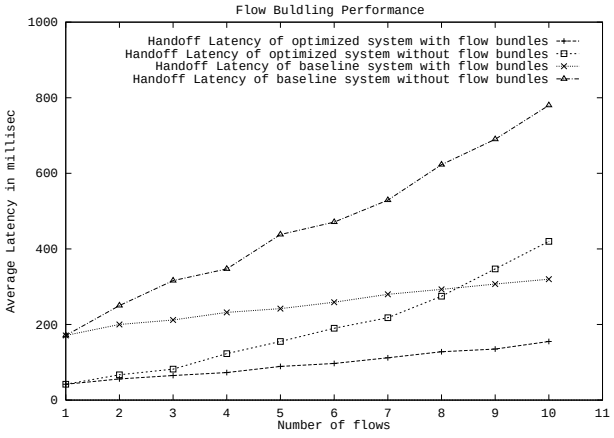


Figure 10: Performance gain with flow bundling

4.2.3 *Mobile Soft-State Analysis*

This experiment demonstrates the ability of mobile devices to adapt their bandwidth needs to changes in wireless-QOS and mobile-QOS based on mobile soft state. Mobile devices periodically probe and adapt to changes in available resources in wireless access networks. Users characterize flows using an adaptive-QOS API (described in Section 2.1) that includes a utility

function and adaptation policy. In this experiment we present two scenarios that illustrate the benefit of mobile soft state in wireless and mobile environments.

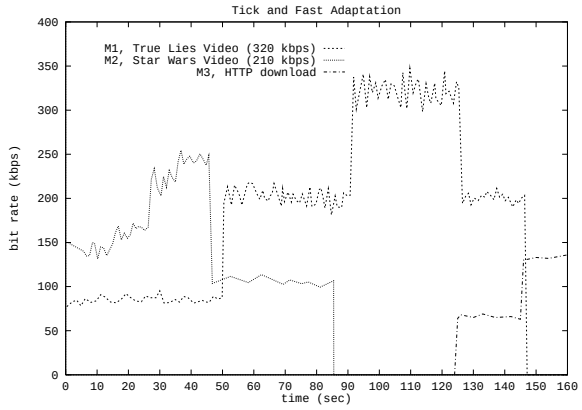


Figure 11a: QOS adaptation within a single cell

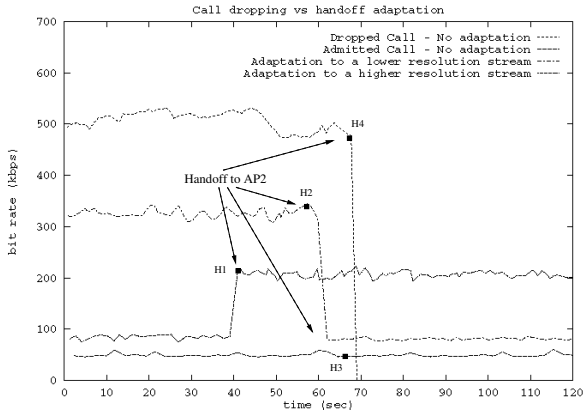


Figure 11b: Handoff driven QOS adaptation

The first scenario illustrated in Figure 11a shows the QOS adaptive behavior of two mobile devices M1 and M2 operating within a single wireless cell. Mobile devices M1 and M2 receive the "True Lies" and "Star Wars" video streams, respectively. Both video flows are based on discretely-adaptive utility functions, i.e., multi-resolution flows. Initially, M1 and M2 receive a base layer (BL) at 80 Kbps, and a base and enhancement layer (E1) at 150 Kbps, respectively. Currently, the adaptive-QOS service gives priority to support the minimum bandwidth requirements of multi-resolution flows [Campbell,95]. During the scenario, M2 registers an increase in bit error rate as it moves away from its current access point. Adaptive FEC is applied to the video between the access point and M1 based on the observed SNR and the measured bit error rate. An adaptive FEC object selectively codes the base and enhancement layers of the "star wars" video increasing the bandwidth consumed by M1 from 150 Kbps to 250 Kbps. For the experiment, the maximum capacity of the air-interface is set to 330 Kbps and around 50 seconds into the scenario the M1 video is adapted back to the base layer with FEC only. Resources released by M1 are consumed by M2 increasing its utility at 50 seconds[7] into the scenario. This situation remains constant until M1 handoffs to a new access point after 80 sec allowing the access point to deliver another enhancement layer to M2. Note that mobile initiated adaptation to released resources (i.e., scaling up) is somewhat dependent on the refresh/probe interval. When a new mobile devices M3 enters the cell around 120 sec access point M1 is explicitly scales back by dropping an enhancement layer. Toward the end of the scenario M3 probes and scale up to a better perceptible quality as M2 hands off to a new access point. At 140 sec into the scenario, mobile device M3 sets up a new flow to access web services initially downloading a GIF file at a rate of 70 Kbps scaling up to 135 Kbps. In related work [Bianchi,98a] we are investigating a generalized adaptation policy mechanism where applications can specify application specific adaptation semantics. For example, some applications would not wish to experience the adaptation observed by M2 while others may be as aggressive as M3 in exploiting any available resources.

The second experiment highlights a number of different QOS adaptation scenarios that can take place during hand off. In this experiment, mobile devices hand off to the access point AP2 from AP1 and AP3. In this experiment, QOS adaptation is not, however, based on the mobile soft-state refresh mechanism described and evaluated in the previous section. Rather, as part of the QOS re-negotiation phase during handoff, mobile devices scale their quality of service needs to match the available resources. The handoff point at which each of the four mobile devices (viz., M1, M2, M3, M4) enter the new access point AP2 is illustrated in the trace shown in Figure 11b. The type of adaptation that takes place after handoff points, which is marked as H1 through H4, is illustrated. During handoff a number of adaptation scenarios may occur depending on the available resources and the ability of existing mobile devices to adapt. For example, the new access point may force existing mobile devices to drop enhancement layers to allow a new mobile to enter the cell with minimum QOS assurances. In this experiment, mobile device M1 enters the new cell at H1 and scales up its utility to take advantage of available resources. M1 adaptation policy is to only scale

[7] Mobile device M1 probes and adapts to the additional bandwidth within a single refresh interval that currently set to 10 sec. There are a number of tradeoffs in setting the probe interval. A Smaller duration allows mobile devices to aggressively grab resources on a fast time scale. However, this increases the signaling load overhead. We are investigating coupling the probing and adaptation time scales to the application level adaptation policy [Bianchi,98a]

after handoff. At point H2 mobile device M2 hands off to the access point AP2 and is forced to scales down to its base layer. Mobile device M3 has an adaptation policy of never adapting. At H3 the mobile hands off to AP2 and maintains its current utility. In the final part of the experiment, M4 hands off to AP2 at point H4. In this instance, insufficient resources are available to support the base layers of M1, M2, M3 and M4 forcing the access point to deny the handoff.

5. Conclusion

In this paper we presented the design, implementation and evaluation of an open programmable mobile network based on distributed object technology called mobiware that dynamically exploits the intrinsic scalable properties of adaptive mobile applications. We have analyzed the performance of mobiware's QOS controlled handoff, flow bundling and mobile soft state algorithms. While the baseline code raised some initial performance concerns about the viability of using distributed object technology for controlling mobile networks the enhanced software is extremely competitive in relation to existing work. The latency measured for QOS controlled handoff was reduced from 171 msec to 51 msec for the handoff of a single flow through two ATM switches making handoff through a single switch in the order of 25 msec. In [Porter,95, Naylon,97] and [Mishra,97] handoff latencies were measured to be 10 and 30 msec for a single flow through a single cross over switch. The use of flow bundling techniques in mobile networks shows great performance increases as the number of flows increase during handoff. The handoff latency for ten flows is 155 msec when flow bundling was enabled and 420 msec when disabled. This clearly shows the advantage of such aggregation techniques. Mobiware's flow bundling compares favorably to the literature. In [Mishra,97] Partho reported a handoff latency of 125 msec for ten flows using native ATM signaling code. Mobile soft-state also exploits aggregation techniques provided by flow bundling. This allows resource probing to be based on flow bundles rather than per-flow. In the paper QOS adaptation techniques clearly demonstrates the benefit of mobile soft-state in sharing resources among competing mobiles in a cell.

A number of researchers have applied distributed object technology to mobile systems. Our work, however, differs from these efforts. First as part of the open signaling community [OPENSIG,96] we are deeply interested in identifying open programming interfaces for mobile and wireless networking. In this work we have identified a number of objects, APIs and algorithms that provide QOS support for adaptive mobile systems. The mobiware technology we have developed over the last two years marks a considerable software effort. To our knowledge we are one of the first group to apply distributed object technology as a mobile middleware solution. Mobiware objects execute on the mobile devices, at the access points and on switch/routers exposing open APIs that can be programmed to support mobile signaling, QOS adaptation management and wireless transport. We observe that once the wireless and mobile APIs have been designed the programming of new network algorithms, e.g., QOS controlled handoff, flow bundling and mobile soft-state was straightforward engineering.

6. Acknowledgements

We would like to thank Aurel A. Lazar and the xbind team for providing the xbind broadband kernel which mobiware is built on. Also, we would like to thank the following colleagues for their major contributing toward the implementation of mobiware: Oguz Angin implemented flow bundles, Anand Balachandran implemented the active media filters, Javier G. Castellanos, retooled the beacon to include QOS hints, Michael E. Kounavis implemented the transport system, Raymond Liao implemented the signaling system, Chien-Ming Yu implemented the active error control and finally Yasuro Shobatake (Toshiba Corp. Japan) implemented the wireless transport management. We would also like to thank Laura Zhou for implementing mobiman and Mun Choon Chan (Lucent Technologies) who implemented the xbind connection management system from where the mobiware mobility agent grew. Finally, we would like to thank Lucent Technologies for providing a beacon API.

7. References

[Balachandran,97] A. Balachandran, A.T. Campbell, M.E. Kounavis, "Active Filters: Delivering Scalable Media to Mobile Devices", *Proc. Seventh Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, St Louis, May, 1997.

[Bianchi,98a] G. Bianchi, A.T. Campbell and R. R.-F. Liao, "Supporting Utility-Fair Adaptive Services in Wireless Networks", *Proc. of 6th International Workshop on Quality of Service (IWQOS'98)*, Napa Valley, May, 1998.

[Bianchi,98b] G. Bianchi and A. T. Campbell, "A Programmable MAC", to appear: ICUPC'98, Florence, October, 1998.

[Campbell,95] A.T. Campbell, D. Hutchison, and C. Aurrecoechea, "Dynamic QOS Management for Scalable Video Flows" , *Proc. Fifth Intl Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Durham, New Hampshire, 1995.

[Campbell,97a] A.T. Campbell and G. Coulson, "QOS Adaptive Transports: Delivering Scalable Media to the Desk Top", *IEEE Network.*, March 1997.

[Campbell,97b] A.T. Campbell, "Mobiware: QOS-Aware Middleware for Mobile Multimedia Communications", 7th IFIP

International Conference on High Performance Networking, White Plains, NY, April, 1997.

[Corson,98] M.S. Corson and A.T. Campbell, "Toward Supporting Quality of Service in Mobile Ad hoc Networks, work in progress session, *First IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[Katz,94] R. H. Katz, "Adaptation and Mobility in Wireless Information Systems", *IEEE Personal Communications Magazine, Vol. 1, No.1,* First Quarter 1994.

[Lazar,97] A. A. Lazar, "Programming Telecommunication Networks", *IEEE Network,* October 1997.

[Lee,95] K. Lee, "Adaptive Network Support for Mobile Multimedia", *Proc. of Mobicom'95*, Berkeley, CA, Nov. 1995.

[Lu,97] S. Lu, K.-W. Lee, V. Bharghavan, "Adaptive Service in Mobile Computing Environment", *Proc. of 5th IFIP Intl. Workshop on Quality of Service (IWQOS'97)*, New York, May 1997, pp25-36.

[Merwe,97] Van der J.E. Merwe and I. Leslie, "Switchlets and Dynamic Virtual ATM Networks", Integrated Network Management V, A.A. Lazar, R. Saracco and R. Stadler, eds., Chapman and Hall, New York, 1997.

[Mishra,97] P. Mishra, "Implementation and Experimental Evaluation of Mobility-enhanced ATM Signaling", OPENSIG FALL '97 Workshop Open Signaling for ATM, Internet and Mobile Networks, New York, October, 1997.

[OPENSIG,96] http://comet.columbia.edu/opensig/

[Mobicom,97] Panel on QoS in the Next Generation Mobile Internet: What is Feasible?, Chaired by A.T. Campbell, The Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'97), Budapest, October, 1997.

[Mobiware,98] Mobiware source code: http://comet.columbia.edu/wireless/software/mobiware

[Naghshineh,97] Naghshineh M., and M. Willebeek-LeMair, "End-to-End QOS Provisioning in Multimedia Wireless/Mobile Networks" IEEE Network, March 1997.

[Naylon,97] Naylon, J., "Radio Handover Measurements", OPENSIG SPRING '97 Workshop Open Signaling for ATM, Internet and Mobile Networks, Cambridge, England, April 1997.

[Porter,95] J. Porter, A. Hopper, D. Gilmurray, O. Mason, J. Naylon, and A. Jones, "The ORL Radio ATM System, Architecture and Implementation", ORL Technical Report, 1995

[Tennenhouse,97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, G. J. Minden, "A Survey of Active Network Research", *IEEE Communications Mag.*, Vol. 35, No. 1, pp 80-86. January 1997.

[xbind,96] xbind Broadband Kernel source code: http://comet.columbia.edu/xbind

Efficient Precomputation of Quality-of-Service Routes

Anees Shaikh[†], Jennifer Rexford[‡], and Kang G. Shin[†]

[†] Department of Electrical Engineering
and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122
{ashaikh,kgshin}@eecs.umich.edu

[‡] Network Mathematics Research
Networking and Distributed Systems
AT&T Labs – Research
Florham Park, NJ 07932-0971
jrex@research.att.com

Abstract

Quality-of-service (QoS) routing satisfies application performance requirements and improves network resource usage by selecting paths based on connection traffic parameters and available link capacity. However, QoS-routing protocols can introduce significant network overhead for computing routes and distributing information about link load. Route precomputation is an effective way to amortize the cost of the path-selection algorithm over multiple connection requests. This paper introduces efficient mechanisms for precomputing one or more routes to each destination, and on-demand checking of the suitability of the routes at connection arrival, based on the most recent link-state information. Simulation experiments show that the route precomputation and route extraction techniques are effective at lowering the computational overheads for QoS routing, while achieving performance similar to the more expensive on-demand path-selection schemes.

1 Introduction

The success of distributed audio and video applications hinges on having predictable performance in the underlying communication network. The network can provide throughput and delay guarantees by reserving resources for individual connections or flows. The routing algorithm plays a pivotal role in this process by locating paths that can satisfy the performance requirements of arriving connections. This is particularly important for handling high-bandwidth multimedia streams, which often consume a relatively large fraction of the link capacity. Quality-of-service routing has the potential to optimize the usage of network resources, and increase the likelihood of accepting new connections, by selecting paths based on existing network load and connection traffic parameters [13, 20, 6]. However, distributing link load information and computing routes for new connections can consume considerable bandwidth, memory, and processing resources [19, 2]. Controlling these overheads in large backbone networks introduces a trade-off between performance and complexity. In this paper, we present an efficient path-selection scheme that precomputes routes, while still capitalizing on the most recent network load information available at connection arrival.

Our study focuses on link-state routing algorithms where the source router or switch selects a path based on connection throughput requirements and the available resources in the network. For example, the ATM Forum's PNNI standard [17] defines a routing protocol for distributing topology and load information throughout the network, and a signalling protocol for processing and forwarding connection-establishment requests from the source. Similarly, proposed QoS extensions to the OSPF protocol include an "explicit routing" mechanism for source-directed IP routing [23, 9]. Each switch maintains its own view of the available link resources, distributes link-state information to other switches, and selects routes for new connections. To improve the scalability of these protocols in large configurations, the switches and links can be assigned to smaller peer groups or areas that exchange detailed link-state information. In this paper, we focus on reducing the overheads of QoS routing within a single peer group, though the algorithms apply to the general case of hierarchical networks. We consider the case of large, relatively well-connected, backbone topologies that offer the possibility of multiple

routes between source-destination pairs. In addition, we compare performance on more sparsely-connected networks that provide fewer routing options.

Efficient QoS routing requires effective techniques for computing routes and exchanging link-state information. Link state is typically propagated in a periodic fashion, or in response to a significant change in available capacity. For example, a link may advertise its available bandwidth metric whenever it changes by more than 10% since the previous update message; in addition, a minimum time between update messages is often imposed to avoid excessive link-state update traffic. When a connection request arrives, the source typically computes a suitable route, using the most recent link-state information. If such a route exists, it initiates hop-by-hop signalling to reserve resources along that path to the destination. The set-up attempt fails if one or more of the links does not have sufficient resources to accept the new connection. After such a set-up failure, the source may decide to try a new route that excludes the offending link, or may simply block the connection. These route computations and signalling attempts consume processing resources at the switches, and introduce set-up latency for each accepted connection. Minimizing these overheads is particularly important during periods of transient overload, such as bursty connection arrivals or rerouting of traffic after a link failure.

Most previous research on QoS routing has investigated *on-demand* policies that compute a path at connection arrival. Recent work considers *precomputation* or *path caching* schemes that attempt to amortize the overheads of route computation by reusing the paths for multiple connection requests [14, 16, 12, 10, 9]. Path precomputation introduces a trade-off between processing overheads and the quality of the routing decisions. Previous work on precomputed routes has focused on quantifying this trade-off and developing guidelines for when to recompute routes. We extend this work by presenting efficient *mechanisms* for precomputing paths under source-directed link-state routing. In particular, this paper addresses several important practical questions:

- How should the precomputed routes be stored?
- How should multiple routes be computed?
- How much work should be performed at connection arrival?
- How should routing and signalling overheads be limited?

We answer these questions with four key elements that are described in detail in Section 2:

- *Coarse-grain link costs:* Path selection is based on link-cost metrics, which are a function of link-state information. Limiting link costs to a small number of values reduces the computational complexity of the path-selection algorithm. Coarse-grain link costs do not significantly degrade performance, and increase the likelihood of having more than one minimum-cost route to a destination.

- *Precomputation of minimum-cost graph:* Each switch or router precomputes a compact data structure that stores all minimum-cost routes to each destination. A small modification to Dijkstra's shortest-path algorithm can locate all minimum-cost routes to each destination. Instead of storing the precomputed paths in a cache or table, route extraction is postponed until connection arrival.

- *Route extraction with feasibility check:* As part of extracting a route, the source checks the feasibility of each link, based on the most recent link-state information and the bandwidth requirement of the new connection. The algorithm performs a depth-first search through the Dijkstra data structure, extracting the first route in the common case. Then, the source initiates signalling in the network to reserve resources along the selected path.

- *Reranking of multiple routes:* The depth-first extraction algorithm imposes an implicit ordering of the links when a node appears in multiple minimum-cost paths. As part of route extraction, the source can rerank these links to improve the path-selection process for the next connection. This provides a simple framework for a number of alternate-routing policies.

These mechanisms enable a wide range of policies for when to compute new routes, how many candidate routes to try for a new connection, and how often to update link-state information. Coupled with our efficient routing mechanisms, these policy decisions allow significant reduction of processing overheads and set-up delay, in comparison to traditional on-demand algorithms. In addition, the simulation experiments in Section 3 show that the feasibility check, and the potential for multiple candidate routes, results in lower likelihood of rejecting requests and lower signalling overheads, in relation to other precomputation schemes. The performance evaluation also investigates the influence of the network topology and link-state update mechanisms on the effectiveness of the path-selection algorithm and routing policies. Section 4 compares our

approach to related work on QoS route precomputation, and Section 5 concludes the paper with a discussion of future research directions.

2 Precomputation of QoS Routes

Reducing the overheads of route computation requires careful consideration about how much information and processing are involved on various time scales. The source receives the most accurate information about network load and connection resource requirements upon the arrival of new link-state updates and connection requests. To lower complexity, though, our precomputation scheme does not perform any work upon receiving a link-state message, beyond recording the new load information, and only modest work on connection arrival. Instead, most of the processing is relegated to background computation of a shortest-path graph of routes to each destination. By delaying route extraction until connection arrival, we exploit the most recent link-state to find a suitable path while incurring only modest overhead. Allowing occasional triggered recomputation for individual connection requests further improves the ability to find a path and to select better routes for future connections.

2.1 Compact Storage of Precomputed Routes

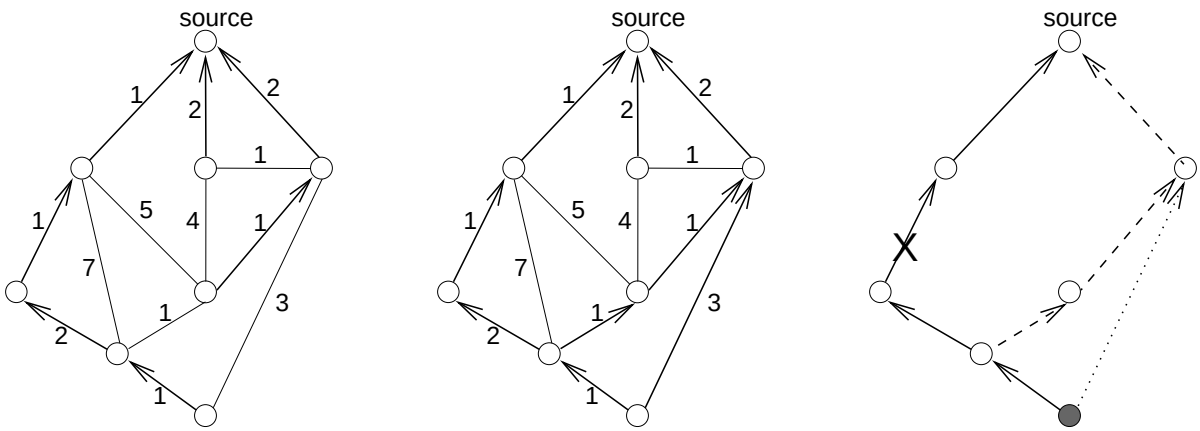
Our study of path precomputation focuses on intra-domain routing in large, flat networks with N nodes (switches or routers) and L links. In this context, we focus on topologies with relatively high connectivity, an increasingly common feature of emerging core backbone networks [22, 8], and on multimedia traffic that requires throughput guarantees. Each switch knows the underlying topology and has (possibly out-of-date) information about the unreserved bandwidth on each link. Link state is flooded periodically, or in response to a significant change in available bandwidth, ensuring that the switches have fairly accurate knowledge of network load. The source switch selects a route for an arriving connection, based on the link state and the connection's bandwidth requirement. Route computation is based on the Dijkstra shortest-path algorithm [5], where link cost (or "distance") is a function of the link load. To minimize resource requirements and end-to-end delay, we focus on link-cost functions that favor routes with a small number of links.

The Dijkstra shortest-path algorithm computes a route to a destination node in $O(L \log N)$ time, when implemented with a binary heap [5]. Although advanced data structures can reduce the average and worst-case complexity [4], the shortest-path computation still incurs significant overhead in large networks. In computing a route to each destination, the Dijkstra algorithm generates a shortest-path graph, where each node has a parent pointer to the upstream node in its route from the source, as shown in Figure 1(a). Extracting a route to a particular destination involves traversing these parent pointers, and introduces complexity in proportion to the path length. For on-demand routing of a single connection, the construction of the shortest-path graph terminates upon reaching the destination. Continuing the computation to generate a route for every destination, however, does not significantly increase the processing requirements, and the complexity remains $O(L \log N)$. This allows path precomputation schemes to amortize the overhead of the shortest-path calculation over multiple destination nodes. Thus, even if some source-destination pairs never communicate, computing their associated routes does not impose much extra cost.

Path precomputation schemes typically store one or more routes in a cache or table. In hop-by-hop routing, the router stores only the next hop of the route to each destination, allowing for simple storage in a route table. In contrast, source-directed routing typically requires the source to maintain a variable-length list of the links on the path to the destination. This list (or stack) becomes part of the signalling message that establishes a connection (e.g., a "designated transit list" in PNNI, or an "explicit route advertisement" in the QoS extensions to OSPF), instructing each intermediate node to forward the message to the next link on the route. To precompute routes, the source could generate paths to one or more destinations and store each route list in a cache, occasionally recomputing one or more routes based on the most recent link-state information. However, computing and extracting multiple routes introduces computational and storage complexity, particularly if some routes are never used. In addition, since link-state information changes over time, these cached routes must be invalidated and/or recomputed periodically. Instead of storing paths in a separate data structure, such as a cache of the variable-length routes, we consider a compact representation that maintains the routes directly in the shortest-path graph, as in Figure 1(a).

2.2 Precomputation of Multiple Minimum-Cost Routes

Path precomputation schemes benefit from having multiple candidate routes to each destination, to balance network load and have additional routing choices in



(a) Shortest-path graph (b) All-shortest-path graph (c) Route extraction

Figure 1. Delayed Pruning of Shortest-Path Routes: The left figure shows a network graph with a link-cost metric on each edge. Each node has a parent pointer to the upstream node along a minimum-cost path from the source. By allowing multiple parent pointers at each node, the graph can represent all of the minimum-cost paths for each node, as shown in the center figure. These parent pointers allow efficient extraction of a route for a specific destination, as shown in the right figure. If a link appears infeasible (denoted by the "X"), an alternate minimum-cost route (shown by dashed lines) can be extracted through a search along the parent pointers.

case of a set-up failure. However, the multiple routes should have similar cost, to avoid selecting paths that make inefficient use of network resources. A switch could conceivably precompute the $k > 1$ shortest paths (in terms of hop-count or other cost) to each destination. Alternatively, a routing algorithm could compute all paths within some additive or multiplicative factor ϵ of the best path. However, these approaches introduce considerable computational complexity. For example, computing the k shortest paths for a single destination in a directed graph has complexity as high as $O(kN^3)$ [21, 11]. In addition, the k shortest paths (or paths within ϵ of optimal) to one node may not be part of the best routes to other destinations; hence, it is usually not possible to store these multiple routes in a compact, shortest-path graph representation.

Instead, we focus on an efficient special case of computing multiple *equal-cost* paths to each destination. This formulation permits a compact representation of the multiple routes, as shown in Figure 1(b). Computing all minimum-cost routes requires a small modification to the traditional Dijkstra computation to store multiple parent pointers for each node; each pointer identifies an upstream node along a shortest path to the destination. The nodes and parent pointers form a directed, acyclic subgraph of the original graph, rooted at the source switch. Each node in the graph has a list of its parent pointers, or a bit-mask to indicate which upstream nodes reside on shortest-path routes. We maintain the parent pointers in a circular list to facilitate simple traversal of the graph when extracting routes. Figure 2 summarizes the algorithm, which is similar to a traditional Dijkstra computation, except for the addition of lines 4–5. The algorithm uses a heap to visit the nodes in order of their distance from the source, and relaxes each outgoing link. A parent pointer is assigned if a link is on a minimum-cost path (so far) to the neighboring node. To minimize the storage requirements and complexity of route extraction, the algorithm can impose a limit on the number of parents for each node (e.g., 2 or 3).

The likelihood of having multiple minimum-cost routes to a destination depends on the link cost function and the underlying network topology. To increase the chance of "ties" in the route computation, we discretize the link costs and map them into a small number of values, C (say, 5 or 10). This approach offers a much cheaper way to compute near equal-cost paths without resorting to a k-shortest computation. Although fine-grain link costs (larger values of C) usually result in lower blocking probabilities, a moderately coarse link-cost function does not significantly degrade performance, particularly if link-state information is stale [19]. When link-state information is somewhat out-of-date, the benefit of a fine-grain link cost function is greatly diminished. Perhaps more importantly,

```
 0:  Heap = set of all N nodes;
 1:  while (Heap is not empty) {                          # visit each node
 2:      u = PopMin(Heap);
 3:      foreach node v ∈ Neighbors(u) {                  # relax each link
 4:         if (dist[u] + cost[u,v] == dist[v])           # equal route
 5:            parent[v] = parent[v] ∪ {u};
 6:         else if (dist[u] + cost[u,v] < dist[v]) {     # cheaper route
 7:            parent[v] = {u};
 8:            dist[v] = dist[u] + cost[u,v];
 9:         }
10:      }
11:  }
```

Figure 2. Dijkstra algorithm with multiple parent pointers: Initially, each node `u` has distance `dist[u]`$=\infty$, except for the source which has a distance of 0. Starting with the source, the algorithm visits the node u with the smallest distance and considers the cost `cost[u,v]` of the link to each neighboring node `v`, extending or reassigning the set of parent pointers `parent[v]` if the new route has equal or lower cost, respectively. At the end, each connected node has one or more parent pointers to upstream nodes along minimum-cost routes from the source.

coarse-grain link costs reduce the processing requirements of the shortest-path computation by reducing heap complexity. The complexity of Dijkstra's algorithm, and the variation in Figure 2, decreases from $O(L \log N)$ to $O(L + CN)$ [5], while more advanced data structures offer even further reduction [4]. Hence, these coarse-grain link costs become particularly appropriate in large, well-connected networks (large L and N) where the switches have stale link-state information.

2.3 Delayed Extraction of Precomputed Routes

Rather than extracting and caching routes in a separate data structure, we store the precomputed routes in the shortest-path graph and delay the extraction operation until a connection request arrives. During the path extraction the source applies the most recent link-state information in selecting a route. The extraction process operates on the subgraph of nodes and parent pointers along minimum-cost routes to the destination, as shown in Figure 1(c). Though the switch could conceivably run a new Dijkstra computation on this subgraph to select the "best" precomputed route, we instead optimize for the common case of extracting the first route by following a single set of parent pointers (e.g., the leftmost path in Figure 1(c)). More generally, we perform a depth-first search through the reduced graph to extract the first *feasible* route, based on the current link-state information and the bandwidth requirement of the new connection. If the extraction process encounters a link that does not (appear to) have enough available bandwidth, the algorithm backtracks to the previous node and tries a different parent pointer.

The depth-first search and the feasibility check effectively "simulate" hop-by-hop signalling, using the most recent link-state information. Note that this operation is purely local at the source, and much less costly than discovering an infeasible link by sending and processing signalling messages in the network. Starting at the destination node, the algorithm builds a stack of the nodes in the route from the source, as shown in Figure 3. Each node has a circular list of parent pointers, starting with the `head` parent; a second pointer `ptr` is used to sequence through the list of one or more parents, until `ptr` cycles back to `head`. Each iteration of the `while` loop considers the link between the node at the top of the stack and its current parent (pointed to by the `ptr` pointer and denoted by `new` in line 4). If the link is feasible, `new` is added to the stack and its parents are considered in the next iteration (lines 5–8). Otherwise, if the feasibility check fails, we proceed to the next parent, or backtrack until we find a node whose parent list has not been exhausted (lines 10–13). The algorithm terminates when we reach the source via a feasible path (`top == src`), or when the path stack becomes empty (`top == NULL`) because no feasible path was found. In the former case, the route is read by popping the stack from source to destination.

A potential drawback of this approach is that a subsequent connection request may have to duplicate some

```
 1:  push(dest);                                  # start at the destination node
 2:  ptr[dest] = head[dest];                      # start at dest's first parent
 3:  while ((top != NULL) and (top != src)) {     # continue until reaching src
 4:      new = ptr[top].nodeid;                   # explore parents of top node
 5:      if (feasible(new, top)) {
 6:         push(new);                            # explore next node in route
 7:         ptr[new] = head[new];
 8:      }
 9:      else {
10:         while (ptr[top].next == head[top])
11:            pop();                             # backtrack from exhausted node(s)
12:         if (top != NULL)
13:            ptr[top] = ptr[top].next;          # sequence to next parent pointer
14:      }
15:  }
```

Figure 3. Depth-first route extraction: Starting at the destination node (`dest`), the algorithm performs a depth-first traversal of parent pointers to construct a stack of the nodes in the route. By the end of the algorithm, the top of the stack points to the source (`src`) if a feasible route exists; otherwise, the stack is empty (`top == NULL`). The algorithm explores multiple parent pointers by sequencing through a circular list, starting with the `head` parent.

of the same backtracking as the preceding request, particularly since the traversal always starts with the parent marked by the `head` pointer. We can avoid this problem by performing some simple pointer manipulations as part of the route extraction process. As we pop each node in the route from the path stack, we may alter the position of its `head` pointer so that a subsequent extraction attempt for the same destination (or any destination along the path) visits a different set of links. Three possibilities are:

- Leave parents in existing order (do nothing).
- Make the selected parent the new head (`head[node] = ptr[node]`). This amounts essentially to "sticky" routing where we stay with the first route that was found to be feasible in the last extraction.
- Round-robin to the next parent (`head[node] = ptr[node].next`). This policy attempts to balance load by alternating the links that carry traffic for a set of destinations.

These alternation policies differ from traditional alternate routing, where the switch rotates among a set of cached *paths*. Here, the rotation for one destination node influences the order in which links are visited for other destination nodes along the path. Moreover, changing the position of the `head` pointer may actually provide added benefit because the alternate routing is performed on a more global scale.

2.4 Route Computation Policy Options

The path precomputation and extraction algorithms provide a useful framework for computing, storing, and selecting from multiple quality-of-service routes. These techniques provide significant latitude in handling individual connection requests, depending on the inaccuracy of the link-state information, connectivity of the underlying network, and tolerance to set-up delay. Table 1 summarizes the key policy decisions, starting with four options discussed in the previous subsections. The remaining design decisions concern how and when to recompute routes and initiate signalling for new connections. The simplest approach relies entirely on a periodic, background computation of the shortest-path graph. Periodic recomputation is likely to simplify CPU provisioning, at the expense of blocking connections on routing and set-up failures. When a request arrives, the source extracts a route and initiates signalling. The source blocks the request if the extraction process does not produce a route, and connection set-up delay is determined almost entirely by signalling delay.

Instead of blocking the connection, the source could trigger recomputation of the shortest-path graph after a failure in route extraction or connection establish-

Policy	Description
on-demand or precomputed	recompute routes for every request or try first to extract from the existing shortest-path graph
allow multiple routes	limit extraction to just one route instead of allowing backtracking to find alternate routes
feasibility checking	whether to use feasibility checking during route extraction
re-rank multiple routes	policy to arrange `head` pointers in the shortest-path graph during route extraction (none, sticky, round-robin)
periodic recomputations	specification of a background route computation frequency
recompute on routing failure	trigger recomputation when initial route extraction fails
recompute on set-up failure	trigger recomputation on set-up failure
reextract on set-up failure	try to reextract another feasible route (rather than recompute) when signalling fails
prune on recomputation	whether to omit the links that failed the admission test when recomputing after a set-up failure
maximum signalling attempts	limit the number of times each connection request may attempt signalling

Table 1. Routing and signalling policy options: This table summarizes some possible policies for connection request handling. The choice of policies may be dictated by objectives such as minimizing processing overhead, connection set-up delay, or blocking probability.

ment. Recomputing routes with the most recent link-state information would reduce connection blocking, at the expense of additional processing load and set-up delay. Inaccurate link state or rapidly arriving connection requests may require frequent path recomputation, introducing computation overhead comparable to on-demand routing. To bound set-up delay and limit processing overheads, the source can impose a maximum number of route computations and signalling attempts for each connection. Since the source recomputes the entire shortest-path graph, the overhead is amortized over all destinations and can benefit subsequent connection requests. To bound the worst-case computational load, the source can impose a minimum time between route recomputations, even in the presence of routing and set-up failures.

When routes are recomputed after a set-up failure, the source can choose to omit the link that failed the admission test. In effect, the set-up failure provides more recent information about the load on the offending link. Temporarily removing this link from consideration is particularly important if the source attempts to compute and signal a new route for the same connection. This new route should avoid using the link that caused the previous failure. In addition, pruning the heavily-loaded link is useful for future connection arrivals, particularly if the shortest-path graph is dedicated to connections with the same (or similar) bandwidth requirements. When connections have more diverse quality-of-service parameters, the source can support a small number of different bandwidth classes (e.g., audio and video), with separate precomputed routes that are tailored to the performance requirements [14, 9]. Employing a different link-cost function and path computation policy for each class enhances the network's ability to route high-bandwidth traffic.

3 Performance Evaluation

In this section, we evaluate the proposed routing algorithm under a range of recomputation periods and link-state update policies. The experiments show that precomputation of the minimum-cost graph, coupled with a feasibility check, approximates the good performance of on-demand routing and the low computational overhead of path caching. The feasibility check is especially effective in reducing the likelihood of expensive set-up failures, even when link-state information is somewhat out-of-date.

3.1 Simulation Model

To evaluate the cost-performance trade-offs of precomputed routes, we have developed an event-driven simulator that models link-state routing at the connection level. A route is chosen for each incoming connection based on a throughput requirement (bandwidth b) and the available bandwidth in the network, based on the source's view of link-state information. Then, hop-by-hop signalling reserves the requested bandwidth at

Parameter	MCI Internet	5-ary 3-cube
offered load	$\rho = 0.65$	$\rho = 0.85$
bandwidth	$b = (0, 4\%]$	$b = (0, 6\%]$
arrival rate	$\lambda = 1$	$\lambda = 1$
connection duration	$\ell = 46.8$	$\ell = 46.8$

Table 2. Simulation invariants: This table lists the parameters that remain fixed throughout the simulation experiments., In the MCI topology $N = 19$, $L = 64$, diameter $D = 4$, and average node degree is 3.4. The 5-ary 3-cube has $N = 125$, $L = 750$, $D = 6$, and degree 6.

each link in the route. That is, if a link has a reserved bandwidth with utilization u, admitting the new connection increases the reservation to $u = u + b$. A set-up failure occurs if a link in the path cannot support the throughput requirement of the new connection (i.e., if $u + b > 1$). For the simulations in this section, we assume that a connection blocks after a set-up failure, though we briefly summarize other experiments that allow multiple signalling attempts. Blocking can also occur during path selection if the feasibility check suggests that none of the candidate routes can support the new connection; these routing failures impose less of a burden on the network than set-up failures, which consume resources at downstream nodes.

The source selects a minimum-hop route with the least cost[1]; previous studies show that algorithms with a strong bias toward minimum-hop routes almost always outperform algorithms that do not consider the hop-count [1, 7, 18, 15]. To distinguish among paths of the same length, each link has a cost in the set $\{1/C, 2/C, \ldots, C/C\}$. For the experiments in this paper, a link with reserved capacity u has cost $c = (\lceil u^2 \cdot (C-1) \rceil + 1)/C$; our experiments with link-cost functions show that an exponent of 2 biases away from routes with heavily-loaded links, without being too sensitive to small changes in link-state information [19]. For simplicity we assume that links are bidirectional, with unit capacity in each direction. We evaluate the routing algorithms on a "well-known" core topology (an early version of the MCI Internet backbone [15, 14]) and a uniformly connected 125-node 5-ary 3-cube topology (with 5 nodes along each of 3 dimensions). The relationship between size and connectivity in the 5-ary 3-cube is similar to existing commercial networks, and allows us to study the potential benefits of having multiple minimum-hop routes between pairs of nodes. The MCI network, on the other hand, is very small and loosely connected.

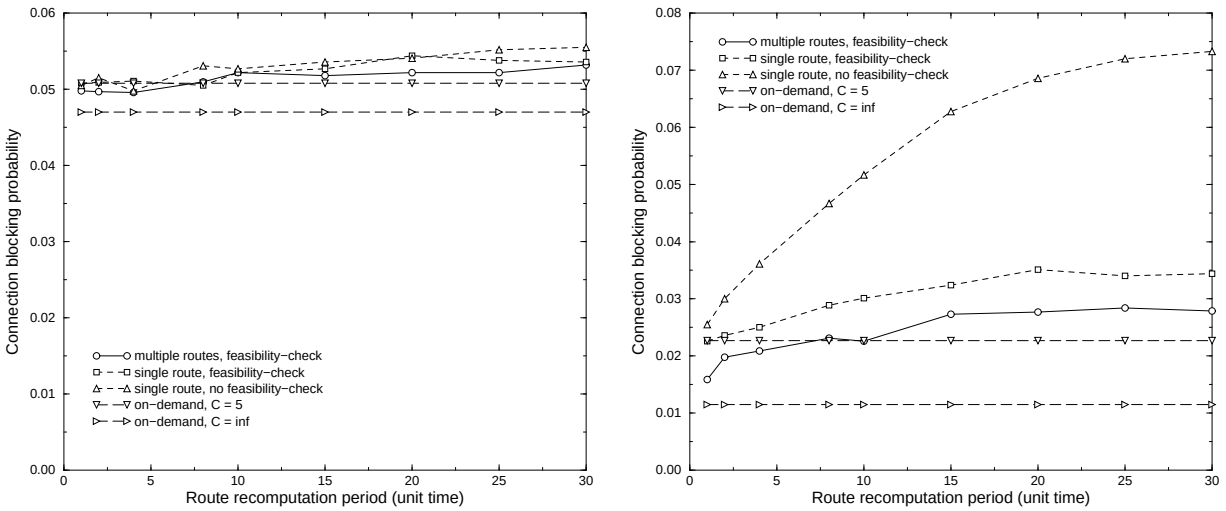
Connection requests arrive at the same rate at all nodes and destinations are selected uniformly. Connection holding times have mean ℓ and follow a Pareto distribution with shape parameter 2.5 to capture the heavy-tailed nature of connection durations[2]. Connection interarrival times are exponentially distributed with mean $1/\lambda$, and requested bandwidths are uniformly distributed with equal spread about a mean size $\bar{b}$. The experiments evaluate connections with mean bandwidth requirements in the range of 2–3% of link capacity. While smaller bandwidths may be more realistic, the large requests provide some insight to the behavior of the high-bandwidth multimedia traffic (e.g., video) we are interested in. Also, smaller bandwidths result in very low blocking probabilities, making it very difficult to gain sufficient confidence on the simulation results in a reasonable time. Our earlier work in [19] considers a wider range of bandwidth requests in the context of QoS routing with inaccurate information. With a connection arrival rate λ at each of N nodes, the offered network load is $\rho = \lambda N \ell \bar{b} \bar{h} / L$, where L is the number of links and $\bar{h}$ is the mean distance (in hops) between nodes, averaged across all source-destination pairs. Table 2 summarizes the simulation parameters for the two network topologies.

3.2 Accurate Link-State Information

The initial simulation experiments compare the performance and overhead of the routing algorithms under accurate link-state information, as shown in Figure 4 and 5. We vary the background period for path precomputation, and also allow a switch to recompute its shortest-path graph when the route extraction does

[1] A careful assignment of link weights w_i permits a single invocation of the Dijkstra algorithm to produce a minimum-cost, shortest path. In a network with diameter D, and link costs in the range $0 < c_i \leq 1$, the link weights $w_i = D + c_i$ ensure that paths with fewer hops always appear cheaper. Such an assignment for w_i results in a relatively small number of possible path cost estimates, thus reducing the complexity of the computation.

[2] We use a standard form of the Pareto distribution with shape parameter a, scale parameter β, and cumulative distribution function $F_X(x) = 1 - (\beta/x)^a$.



(a) MCI Internet topology

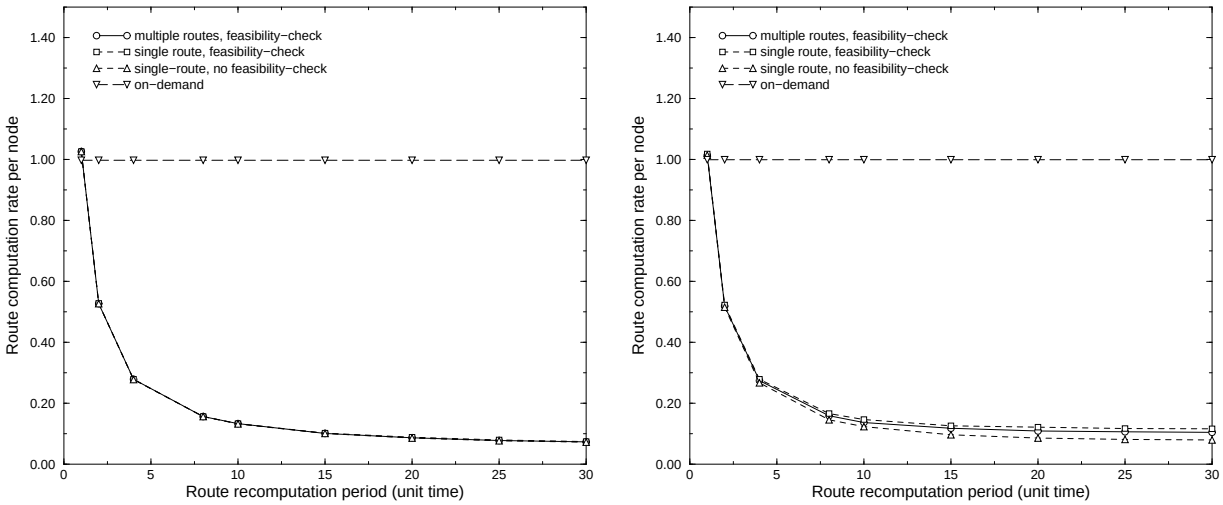
(b) 5-ary 3-cube

Figure 4. Performance with accurate link state: These graphs plot the blocking probability as a function of the background path computation period under accurate link-state information, for both the MCI and 5-ary 3-cube topologies.

not produce a candidate route (due to failed feasibility checks) and after a set-up failure. Since we allow only one signalling attempt for each connection, recomputing after a set-up failure only benefits future arrivals. The single-route algorithms use a precomputed shortest-path graph with one route, as in Figure 1(a). The multiple-route approach offers greater flexibility by allowing the source to perform feasibility checks on several paths in the graph, as in Figure 1(b). We also consider two on-demand algorithms that compute routes on a per-request basis. One has a link-cost discretization of $C = 5$, identical to the precomputed routing algorithms, and the other has discretization limited only by machine precision (referred to as $C = \infty$).

Figure 4 plots the connection blocking probability as we increase the recomputation period to more than 30 times the connection interarrival time. The precomputation algorithms perform well, relative to the more expensive on-demand schemes. Feasibility checking substantially improves performance over traditional path-caching techniques, allowing the use of much larger computation periods. In addition, under accurate link-state information, feasibility checking completely avoids set-up failures by "simulating" the effects of signalling during the route extraction process. This is particularly helpful in the 5-ary 3-cube network, since the richly-connected topology frequently has an alternate shortest-path route available when the first choice is infeasible. On-demand routing with $C = 5$ suffers somewhat by its inability to distinguish the "best" route, compared to the $C = \infty$ case, but this effect diminishes under link-state inaccuracy. Other experiments illustrate that, without a feasibility check, precomputed routing can only achieve these performance gains by allowing multiple signalling attempts for a connection, at the expense of longer set-up delays and higher processing requirements.

The ability to precompute multiple candidate routes does not substantially reduce the blocking probability in Figure 4(a), since the sparsely-connected MCI topology typically does not have multiple shortest-path routes, let alone multiple routes of equal cost. In contrast, the 5-ary 3-cube experiment in Figure 4(b) shows a more substantial performance benefit. These benefits will apply to real networks as they grow larger and more densely connected. We also expect a more significant gain under nonuniform traffic loads, since alternate routes would enable connections to circumvent regions of congestion. This is especially true during transient fluctuations in network load, caused by bursty connection arrivals or rerouting after a link failure. In these cases, the precomputation of multiple routes allows the source to survive longer intervals of time without computing a new shortest-path graph. The algorithm with multiple precomputed routes, coupled with the feasibility test, performs almost as well as on-demand routing with the same number of cost levels. However, using $C = \infty$ offers a noticeable performance advantage in Figure 4, since the accurate



(a) MCI Internet topology (b) 5-ary 3-cube

Figure 5. Overhead with accurate link state: These graphs plot the route computation frequency as a function of the background path computation period under accurate link-state information for both the MCI and 5-ary 3-cube topologies.

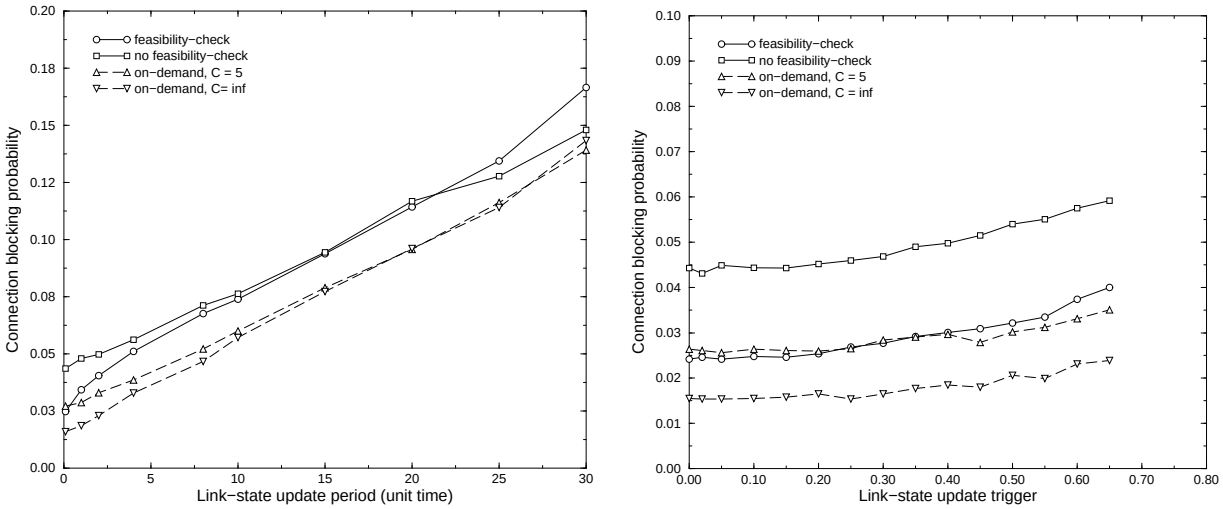
link-state information enables the fine-grain link-cost function to locate the "best" routes.

The lower blocking probabilities for on-demand routing come at the expense of a significant cost in processing load, as shown in Figure 5. In both the MCI and 5-ary 3-cube topologies, the route computation frequency is lowered by a factor of 10 below that of on-demand routing. In addition, the precomputation schemes have much lower complexity for each route computation, relative to the $C = \infty$ on-demand algorithm. Comparing the different precomputation algorithms, the processing load is dominated by the background recomputation of the shortest-path graph, though the feasibility test introduces slightly more triggered recomputations. As the period increases, the graphs flatten since triggered recomputations become increasingly common for all of the algorithms. Although disabling these triggered recomputations results in more predictable processing overheads, additional simulation experiments (not shown) indicate that this substantially increases the blocking probability.

3.3 Inaccurate Link-State Information

While the previous experiments assume that the source has accurate knowledge of network load, Figure 6 considers the effects of stale link-state information, for both periodic and triggered link-state updates, with a background recomputation period of 5 time units. As staleness increases, the $C = \infty$ and $C = 5$ curves gradually converge, since fine-grain link costs offer progressively less meaningful information about network load. However, large link-state update periods also degrade the effectiveness of the feasibility check in our precomputed routing scheme, as shown in Figure 6(a). Periodic updates can lead the source switch to mistakenly identify infeasible links as feasible, and feasible links as infeasible. When link-state information is extremely stale (e.g., an update period that is 20 times larger than the mean connection interarrival time), signalling blindly without a feasibility check offers better performance. In fact, under such large update periods, none of the QoS-routing algorithms perform well; under the same network and traffic configuration, even *static* shortest-path routing (not shown) can achieve a blocking probability of 16%. Experiments with the MCI topology configuration do not show as much benefit from feasibility checking, due to the lower number of shortest-path routes, though the feasibility test does reduce the likelihood of set-up failures.

Despite the effects of large update periods, we find that the feasibility check still offers significant advantages under more reasonable levels of link-state staleness. This is particularly true when link-state updates are triggered by a change in available link capacity, as shown in Figure 6(b). For example, a trigger of 0.2 spawns a link-state update message whenever available capacity has changed by 20% since the last update, due to the establishment and termination of con-



(a) Periodic link-state updates

(b) Triggered updates

Figure 6. Performance with stale link state: These graphs plot the blocking probability for the 5-ary 3-cube topology for periodic and triggered link-state updates, with a background path computation period of 5 time units.

nections. Triggered link-state updates, like triggered path recomputations, generate new routing information during critical fluctuations in network load. Under triggered link-state updates, the switches typically have accurate information about heavily-loaded links, even for large trigger values. Consequently, the connection blocking probability is fairly insensitive to the trigger, across all of the routing algorithms. In addition, feasibility checking remains very effective across the range of update triggers and competitive with on-demand routing (with $C = 5$), in contrast to the results for update periods in Figure 6(a). We find also that using triggered updates allows feasibility checking to reduce blocking even in the MCI topology, though the difference between feasibility and no feasibility checking is less significant than in the 5-ary 3-cube.

Although the blocking probability remains nearly constant as a function of the link-state trigger, small triggers result in fewer set-up failures, as long as the source performs a feasibility test. In contrast, set-up failures account for *all* connection blocking when routing does not involve feasibility checking. As the trigger grows, the feasibility test sometimes mistakenly concludes that a heavily-loaded link is feasible, and the connection blocks in signalling instead of experiencing a routing failure. Still, even for a 50% trigger, set-up failures only contribute 30% of the connection blocking under feasibility checking. Also, despite the fact that periodic updates cause significant staleness and worsen overall connection blocking, feasibility checking still manages to avoid signalling for connections that ultimately block about $30-40\%$ of the time. The ability of feasibility checking to reduce blocking inside the network is an important benefit since set-up failures consume processing resources and delay the establishment of other connections. Routing failures, on the other hand, are purely local and do not consume additional resources beyond the processing capacity at the source switch.

Examining the blocking relative to hop-count shows that feasibility checking at the source also helps in routing connections between distant source-destination pairs. Since bandwidth must be reserved on more links, signalling blindly without checking recent link state has a lower probability of finding a successful route. Other experiments show that when a hold-down timer is used to impose a minimum time between consecutive update messages, the blocking probability rises slightly for all algorithms, though the benefit of feasibility checking remains. The hold-down timer is useful, however, in reducing link-state update overhead, especially when using small trigger levels. For example, with a hold-down timer equal to the mean connection interarrival time, the update generation rate can be reduced by over 35% for triggers in the range $0-0.1$. Moreover, even with a hold-down timer, coarser triggers do not degrade performance. The combination of triggered updates and a small hold-down timer, coupled with feasibility checks and multiple precomputed routes, offer an efficient and effective approach to quality-of-service routing in large

networks.

4 Related Work in QoS Route Precomputation

Previous work on route precomputation has focused on path caching policies, performance evaluation, and algorithmic issues. Our work complements these studies by emphasizing lower-level mechanisms and introducing an efficient framework for precomputing and storing QoS routes, which applies to a variety of routing and signalling policies.

Research on path caching has focused on storing routes in a separate data structure and considering different policies for updating and replacing precomputed routes. The work in [16] introduces a policy that invalidates cache entries based on the number of link-state updates that have arrived for links in the precomputed paths. The proposed algorithms also check the current link-state when selecting a path from the cache and allow recomputation when the cached paths are not suitable, similar to the feasibility check in this paper. This earlier work does not, however, address route computation or path extraction mechanisms. Another study proposes a set of route precomputation policies that optimize various criteria, such as connection blocking and set-up latency [10]. The algorithms try to locate routes that satisfy several QoS requirements through an iterative search of precomputed paths (optimized for hop-count) followed, if necessary, by several on-demand calculations that optimize different additive QoS parameters, one at a time.

Other research has focused on detailed performance evaluation to compare precomputed and on-demand routing under different network, traffic, and staleness configurations. The work in [3] evaluates the performance and processing overhead of a specific path precomputation algorithm. The study adopts a Bellman-Ford-based algorithm from [9] and evaluates a purely periodic precomputation scheme under a variety of traffic and network configurations. The study presents a detailed cost model of route computation to compare the overhead of on-demand and precomputed strategies. As part of a broader study of QoS routing, the work in [14] evaluates a class-based scheme that precomputes a set of routes for different bandwidth classes. The evaluation compares the performance of several algorithms for class-based path computation to on-demand computation. These two studies do not propose any particular strategy for path storage or extraction but instead focus on performance trends.

The remaining studies consider different ways to precompute paths for multiple destination nodes and connection QoS requirements. The work in [9] proposes a Dijkstra-based algorithm that computes minimum-hop paths for different bandwidth classes. Another algorithm, introduced in [12], precomputes a set of extremal routes to all destinations such that no other route has both higher bottleneck bandwidth *and* smaller hop-count. The Bellman-Ford-based algorithm in [9] uses a similar optimization criterion to construct a next-hop routing table with multiple routing entries for each destination. The emphasis of these last three proposals is on algorithmic issues, such as reducing complexity. In addition to a focus on computational overheads, we also consider efficient ways to store precomputed paths and apply the most recent link-state information.

5 Conclusions and Ongoing Work

In this paper, we have proposed efficient mechanisms for precomputing quality-of-service routes, while still applying the most recent link-state information at connection arrival. Route computation employs a small extension to Dijkstra's algorithm, coupled with discretized link costs, to generate a shortest-path graph with one or more routes to each destination. On connection arrival, route extraction involves a depth-first search with a feasibility test, which returns the first route in the common case. Simulation experiments show that the algorithm offers substantial reductions in computational load with only a small degradation in performance, compared to more expensive on-demand algorithms. Our precomputation scheme continues to perform well under stale link-state information, particularly under triggered link-state updates. In addition to having lower blocking probabilities than traditional path-caching schemes, the feasibility test reduces network overhead by decreasing the frequency of signalling failures.

As part of future work, we are investigating enhancements to our path precomputation and extraction algorithms. To increase the likelihood of having multiple candidate routes to each destination, we are considering heuristics for generating *near*-minimum-cost alternate routes, while still storing the precomputed routes in a compact graph representation. In addition, we are pursuing other ways to test the suitability of routes during the extraction process. For example, instead of checking *link* feasibility, the source could use the most recent link-state information to compute new *path* costs as part of the depth-first search for a route; whenever the accumulated path cost exceeds a certain threshold, the algorithm can backtrack to consider other precomputed routes. This approach is well-suited

to precomputing paths that balance network load without requiring information about the traffic or QoS parameters of individual connections. Our initial examination of different policies for alternating between links during route extraction did not show a significant effect on the general performance trends. We intend to explore the load-balancing semantics of these link-level alternation policies in more detail. Finally, we are performing more extensive simulation experiments to evaluate our path-selection schemes under a wider range of network topologies, communication workloads, and routing and signalling policies.

Acknowledgment

The authors wish to thank Gisli Hjalmtysson and Khawar Zuberi for their constructive feedback on earlier versions of this paper.

References

[1] H. Ahmadi, J. S. Chen, and R. Guerin. Dynamic routing and call control in high-speed integrated networks. In *Teletraffic and Datatraffic in a Period of Change: Proceedings of the International Teletraffic Congress*, volume 14 of *Studies in Telecommunication*, pages 397–403. North-Holland, June 1991.

[2] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Quality of service based routing: A performance perspective. To appear in *Proc. ACM SIGCOMM*, September 1998.

[3] G. Apostolopoulos and S. K. Tripathi. On the effectiveness of path pre-computation in reducing the processing cost of on-demand QoS path computation. In *Proceedings of IEEE Symposium on Computers and Communication*, June 1998.

[4] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest-path algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73(2):129–174, May 1996.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press (McGraw-Hill), Cambridge, MA (New York), 1990.

[6] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A framework for QoS-based routing in the Internet. Internet Draft (draft-ietf-qosr-framework-04.txt), work in progress, April 1998.

[7] R. Gawlick, C. Kalmanek, and K. Ramakrishnan. Online routing for virtual private networks. *Computer Communications*, 19(3):235–244, March 1996.

[8] A. G. Greenberg and R. Srikant. Computational tecniques for accurate performance evaluation of multirate, multihop communication networks. *IEEE/ACM Transactions on Networking*, 5(2):266–277, April 1997.

[9] R. Guerin, A. Orda, and D. Williams. QoS routing mechanisms and OSPF extensions. In *Proceedings of IEEE GLOBECOM*, Phoenix, AZ, November 1997. Extended version appears as Internet Draft (draft-guerin-qos-routing-ospf-03.txt), March 1998.

[10] A. Iwata, R. Izmailov, H. Suzuki, and B. Sengupta. PNNI routing algorithms for multimedia ATM internet. *NEC Reserach & Development*, 38(1), January 1997.

[11] E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, March 1972.

[12] J.-Y. Le Boudec and T. Przygienda. A route precomputation algorithm for integrated services networks. *Journal of Network and Systems Management*, 3(4):427–449, 1995.

[13] W. C. Lee, M. G. Hluchyj, and P. A. Humblet. Routing subject to quality of service constraints in integrated communication networks. *IEEE Network Magazine*, pages 46–55, July/August 1995.

[14] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *Proceedings of IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997.

[15] Q. Ma and P. Steenkiste. Quality-of-service routing for traffic with performance guarantees. In *Proc. IFIP International Workshop on Quality of Service*, pages 115–126, Columbia University, New York, May 1997.

[16] M. Peyravian and A. D. Kshemkalyani. Network path caching: Issues, algorithms and a simulation study. *Computer Communications*, 20:605–614, 1997.

[17] PNNI Specification Working Group. *Private Network-Network Interface Specification Version 1.0*. ATM Forum, March 1996. Document available at ftp://ftp.atmforum.com/pub/approved-specs/af-pnni-0055.000.

[18] C. Pornavalai, G. Chakraborty, and N. Shiratori. QoS based routing in integrated services packet networks. In *Proceedings of IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997.

[19] A. Shaikh, J. Rexford, and K. Shin. Dynamics of quality-of-service routing with inaccurate link-state information. Technical Report CSE-TR-350-97, Computer Science and Engineering Division, University of Michigan, Ann Arbor, MI, November 1997.

[20] Z. Whang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 1996.

[21] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, July 1971.

[22] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM*, pages 594–602, March 1996.

[23] Z. Zhang, C. Sanchez, B. Salkewicz, and E. S. Crawley. Quality of service extensions to OSPF or quality of service path first routing (QOSPF). Internet Draft (draft-zhang-qos-ospf-01.txt), work in progress, September 1997.

A Soft State Protocol for Accessing Multimedia Archives

Angela Schuett, Suchitra Raman, Yatin Chawathe,
Steven McCanne, Randy Katz
{schuett,suchi,yatin,mccanne,randy}@cs.berkeley.edu
Department of EECS
University of California, Berkeley

Abstract

The advent and deployment of IP Multicast has fueled the growth of multi-user collaborative applications. The MBone, the Internet multicast infrastructure, extends the traditional best-effort unicast delivery model of the Internet to enable efficient multipoint packet delivery. Using IP Multicast to deliver archived multimedia content can provide significant advantages in scalability and efficiency. With the popularity of collaboration tools for audio/video conferencing on the MBone, we have seen a number of applications developed to record and play back these MBone sessions. In this paper, we look at the requirements for an archive system control protocol that allows the initiation and manipulation of a playback session. We present a control protocol which itself uses IP multicast. This provides benefits in scalability, fault recovery, and allows simultaneous control of an archive session by multiple clients. The interesting features of our protocol include the application of soft *server state techniques, and a built-in mechanism for fault detection and recovery. We have implemented and deployed a prototype and are actively using it in our research group.*

1 Introduction

The development and deployment of the IP multicast infrastructure (the MBone) in the Internet has enabled the development of a number of collaborative applications. IP Multicast [Dee91, DEF+96] provides an efficient mechanism for multi-point data delivery. The new class of multi-party applications enabled by IP Multicast includes a variety of audio and video conferencing tools [Sch92, MJ95, JM], shared whiteboards and text editors [McC92, RT96, HC97], and floor control applications [MR97]. If computer supported collaboration is to become successful and mainstream, it must at the very least provide functionality that was previously unavailable in face-to-face meetings. By harnessing the computing technology at our disposal today, instead of merely imitating face-to-face communication, we can *enhance* collaboration by intelligently adding functionality through the electronic medium [HS92].

A key functionality that is possible and useful in this new medium is the ability to record and play back collaborative sessions, for example, a group meeting. A recording agent can participate in the multicast session associated with the meeting and archive the entire session. Such an archive would be valuable resource for people who missed the meeting, or who wanted to review certain portions of the meeting. Moreover, the ability to play back archived data on to the MBone as part of a new session allows the archive itself to be a tool for collaboration.

Delivering stored multimedia content to users is a complex, multi-part problem. Such a system must include components for disk scheduling on the server, data replication algorithms, media codecs, admission control schemes, etc. This paper is concerned primarily with the archive control protocol, i.e., the protocol used to initiate contact with an archive server, request playback of a presentation, and control that presentation (for example, with pause and seek operations). However, the design of the archive control protocol does interact with and impact other areas of archive architecture, and this needs to be considered in order to achieve fault tolerance, scalability, and robustness.

In this paper we present a soft state protocol for controlling access to archived sessions. Soft state denotes a type of protocol where state is constantly refreshed by periodic messages. State which is not refreshed in time expires. Advantages of the soft state approach include implicit error recovery. The concept of soft state has been widely used for building robust and reliable systems. A number of network protocols for Internet routing [MRR80], IP multicast routing [DEF+96], and the Snoop protocol for improving TCP performance over wireless networks [BSAK95] use soft state to achieve robustness and good performance. We heavily rely on the use of soft state throughout our design. As we will demonstrate, this design choice greatly simplifies fault management resulting in high availability.

Soft state protocols are also amenable to implementation on multicast. We explore the use of multicast to carry the archive control protocol. This allows clients that are sharing a data session to also share a control session, enhancing scalability. We also use multicast in our archive architecture to mask failures in the archive service.

The remainder of the paper is organized as follows: Sec-

tion 2 outlines functions of the archive control protocol. Section 3 describes SSAC, our soft state archive control protocol. Section 4 presents our implementation of SSAC-based client and server applications. Section 5 describes the mechanisms for evaluating this type of archive control protocol. Section 6 discusses related work, and Section 7 summarizes this paper.

2 Protocol Functionality

We describe here the three core pieces of functionality provided by our archive control protocol: (1) *maintaining shared control state*, (2) *group organization* and (3) *fault detection and recovery*. We discuss each of these issues in more detail below.

2.1 Maintaining Shared State

The primary function of an archive control protocol is the exchange of control parameters, for example, the title of the presentation to be played, data streaming addresses, the server's current position in time in the stream, etc. This parameter exchange causes both the server and the client to construct a shared state table.

One approach to establishing this state table is to use a reliable, connection-oriented byte stream transport protocol such as TCP. The Real-time Streaming Protocol (RTSP) [SRL98] uses this approach[1]. While this is straightforward in the single client-single server case, it is not suitable when multiple clients share a single control session. If each client in the shared session requires a separate connection to the server, the server must maintain expensive per-client connection state. An alternative to this approach is for a single client to contact the server, obtain a session multicast address and announce the multicast channel where the presentation is being played. This approach, however, does not allow more than one client to cooperatively control a session. If the controlling client fails, the "remote control"is lost!

2.2 Group Formation and Management

The full benefits of using IP multicast for data and control depend on the ability of the control protocol to group multiple end-clients together. Two different scenarios exist for multiple client control: explicit and implicit client groups. An *explicit* group is formed when a group of clients wishes to view a single presentation together, under group control. For example, in an "online" distributed classroom setting, geographically distributed users interacting with MBone conferencing tools may wish to jointly view a recorded presentation. As the presentation progresses, participants may pause the playback in order to discuss interesting points among themselves. Using a VCR metaphor, each participant has a "remote control", which operates on a single archive system shared by all. If one group member performs a seek or pause operation, that operation must be propagated to all other members of the group.

An *implicit* group, on the other hand, is formed when independent clients simply happen to be viewing the same playback session at the same point in time. Servers may coalesce these clients into a single group to improve efficiency. If any of the members of such an implicit group perform a seek/pause operation, the group is automatically split up. The use of implicit client groups has been explored in several systems as a means to improve server scalability [DSS94, BNKL98].

2.3 Fault Detection and Recovery

Fault detection in the context of media streaming systems involves detecting both client and server failures. If a client-side failure occurs, the server should detect this, and stop delivering data. This is necessary so that server resources do not become "orphaned". Similarly, the client should be able to detect server failures and notify the user and, possibly, try to recover from the failure by switching to a different server with minimal disruption of service.

Client or server failure can be detected through several means, such as the loss of a TCP connection, or the timeout of some periodic "heartbeat" message. It is also possible to use the playback data stream as a source for fault detection. However, this approach has several problems:

- First, some data delivery protocols may not contain the periodic "heartbeat" messages required to detect faults.
- Second, detecting archive faults through data streams impedes reuse of mechanism. For example, a typical scenario for an Internet archive system may involve developers who provide data streaming modules corresponding to proprietary formats which plug into the archive system. If the archive control interacts on a fine-grained basis with these data streaming modules, then adding support for new modules will be more difficult.
- Finally, clients should not be required to receive data in order to control playback, and playback viewers should not necessarily be required to run the archive control protocol. As a motivating example, consider a presentation room which features a variety of display devices. The speaker controls the playback of a presentation from her palmtop, while the video data stream is actually received on a computer with a large display screen. With protocols that depend on detecting faults through the data stream, the palmtop controller would unnecessarily be required to receive all the data streams.

[1] RTSP does not require the underlying protocol to be TCP, but does expect RTSP messages to be received in-order and reliably.

We believe that the appropriate place for fault detection and recovery mechanisms in an archive system is the archive control protocol. Fault discovery and recovery are especially important for an MBone archive system. New protocols and codecs are constantly being developed, and software is being deployed in the wide area well before all the bugs have been worked out. In addition, network connectivity problems are especially frequent on the MBone. By incorporating a robust fault tolerance protocol in the archive design, we can ensure that the service continues to operate even in the presence of buggy software modules or network failures.

2.4 Design Principles

We have developed an archive control protocol that addresses maintenance of shared state, group formation and management, and fault detection and recovery through the use of the multicast protocol design principles of *light-weight sessions* (LWS) [McC98, Jac94]. Light-weight sessions refers to the fault tolerant, scalable, decentralized communications model that has been used successfully in many MBone conferencing protocols such as RTP/RTCP [SCFJ96] for audio and video and SAP [HJ97] for session announcements. Instead of instantiating "hard" server state as in centralized protocols, light-weight sessions use a "soft state announce/listen approach," in which an agent periodically transmits a status announcement. Listening members update their status information on receiving an announcement. If no announcements are received, state eventually times out.

An important design goal that has consistently underscored the development of the LWS architecture is an explicit attempt to accommodate the highly heterogeneous and loosely-controlled nature of the Internet. The Internet is large, complex, and composed of heterogeneous components with diverse failure modes and mixed levels of reliability. To account for this, LWS protocols are deliberately designed to be robust and forgiving. For example, if a network link outage causes a temporary network partition, then the higher-level LWS protocols do not "reset" or terminate the session. Instead, the session continues its existence as a set of partitioned sub-sessions. Later, when the partition heals, the session gracefully recovers and integrates independently generated activity and state across the healed partition. LWS achieves this robustness to network failures by exploiting receiver-driven, soft state protocols that treat recovery from such failures as part of normal protocol function.

We apply LWS to multicast archive control, to ensure that our protocol is scalable, fault tolerant and efficient. Specifically, we use soft state between clients and server, and, whenever possible, application and protocol decisions are performed in a decentralized manner. This lessens the server load and simplifies the implementation of explicit client groups.

3 Soft State Archive Control (SSAC) Protocol

The central functionality of the Soft State Archive Control Protocol (SSAC) is to implement the exchange of control information between the clients and server of an archived session. For example, before playback can begin, the client must inform the server about the session that it wishes to be played, and a playback point within the session. While the session is in progress, the client may decide to momentarily pause the session, or seek to a different point in the session. The SSAC protocol must robustly convey all such information between the clients and the server.

In addition to the above control information, the client must be able to determine other parameters such as the location of the server, the "name" of the presentation to be played out, media types in the presentation, data formats in the presentation, etc. For the purpose of the archive control protocol, we assume that the client has already previously gathered this information. This information may be collated in the form of a "presentation description file" and placed on the world-wide web, where the client can easily download it.

Based on the soft state periodic update approach of light-weight sessions, we have designed the SSAC protocol to use "announce/listen" as the underlying transport mechanism. A participant in an announce/listen session periodically *announces* its state updates, while all other participants *listen* to and cache these updates. The updates are assumed to be soft state and are eventually aged out, unless over-ridden by a new update from the announcing client. Announce/listen protocols are robust and can be implemented over an unreliable datagram transport protocol such as UDP. Since no central hard state is involved in such a protocol, it makes the protocol much more resilient to failures.

The use of announce/listen eliminates the need for heavy-weight reliable protocols such as TCP or SRM [FJM+95] in the implementation of SSAC. It allows SSAC to run over unicast UDP or multicast.

To present the protocol, we begin by explaining the case where a single client connects to the server using unicast. In Section 3.2 we expand this description to include multiple clients sharing a multicast control channel. To illustrate the fault recovery mechanisms of SSAC, throughout this section we will use a simple example distributed server architecture, shown in Figure 1. The client uses SSAC to communicate with a **server manager**. When the client requests a new presentation, an **archive agent** is launched, which streams the presentation onto a data channel.

3.1 Single Client SSAC

In an archive control session, certain pieces of information need to be exchanged, some originating with the client, and some with the server. The client originates three state parameters: the control session id, the presentation id, and the

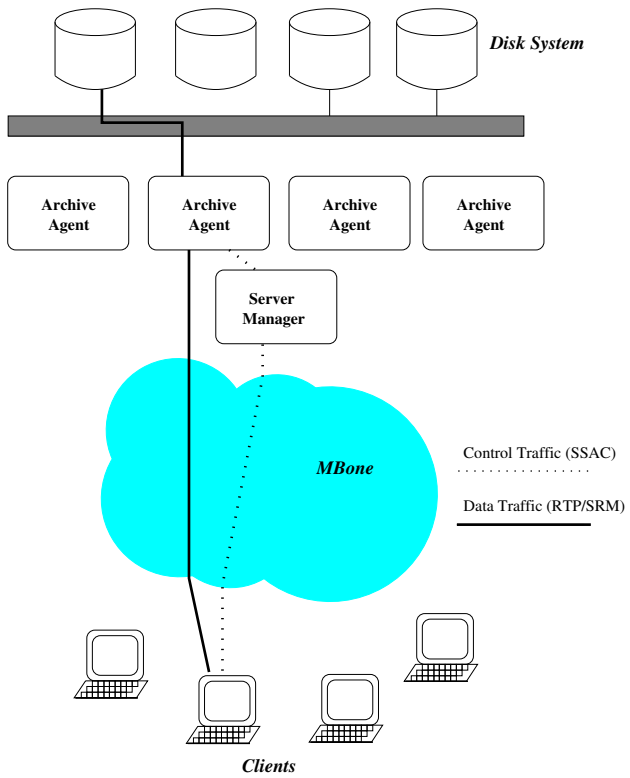


Figure 1: Archive server system showing control and data channels

offset into the playback session. If the client wishes to play the session from the beginning, the offset is 0. The server transmits the values of other parameters, including data address blocks, and the presentation duration.

CSID	210@128.32.130.14
PSID	ssac://pub/classes/CS294/jan30
Offset	0
Seq	1

Table 1: Example SSAC client announcement

Table 1 shows a sample client announcement packet. The control session id, (CSID) can be any client generated information which will be unique at the server for the life of the control session, for example, *client process id : client IP address*. The presentation id (PSID) gives the unique name of the program to be played. This could be similar to a web URL. The Offset field is the means to transmit information about the time offset into the data session. For example, a seek operation is implemented by modifying the Offset field. The Seq field is a sequence number which is incremented by the client or server whenever a change is made to the shared state.

CSID	210@128.32.130.14
PSID	ssac://pub/classes/CS294/jan30
Offset	5
Seq	2
Media	Video
Address	224.2.3.4/5555
Media	Audio
Address	224.2.3.5/6666

Table 2: Example SSAC server announcement

Once an archive agent is launched, and has received its first client announcement, it begins reflecting the control state, which includes the CSID, PSID and Offset, as well as some server-originated parameters, such as data address blocks. Table 2 shows a sample server announcement packet. The server chooses the multicast addresses where data sessions are transmitted. These addresses, along with information about which media type is transmitted on each address, are specified in the address blocks.

Each server packet uses the Offset field to transmit the current offset into the playback session. This offset is then reflected by the client in its announcement. As we will see, in the case of an archive agent crash, this allows a new agent to begin streaming from the proper place in the session. This periodic update of the offset field does not cause the sequence number to be incremented. If the client performs a seek operation, the client announces a new offset value, and increments the sequence number. When the archive agent performs the seek request, it sets the offset field to reflect the new value, and increments the sequence number field again. In this way, client and server announcements can “cross” in the network without harm. Messages which have out of date sequence numbers are discarded.

Throughout an SSAC session, the server and client periodically announce state information. This has two purposes. First, if an announcement is lost in transmission, the next announcements will serve as replacements. Second, if the archive agent crashes, the next client announcement to arrive at the server manager will contain all the information required to restart the playback session.

Fault discovery and recovery are simplified by the use of soft state. Recall that in a soft state protocol, state mappings time out if they are not refreshed periodically. If the client crashes, the server’s state will time out, and the server will stop sending data. If the client recovers and begins announcing again, the server will resume data transmission. If the archive agent crashes while streaming, SSAC messages which arrive at the server manager will cause a new archive agent to be created. The new archive agent receives the offset, and begins streaming from the logical position in the stream denoted by the offset. The client begins receiving new announcements, notes the new data channel addresses,

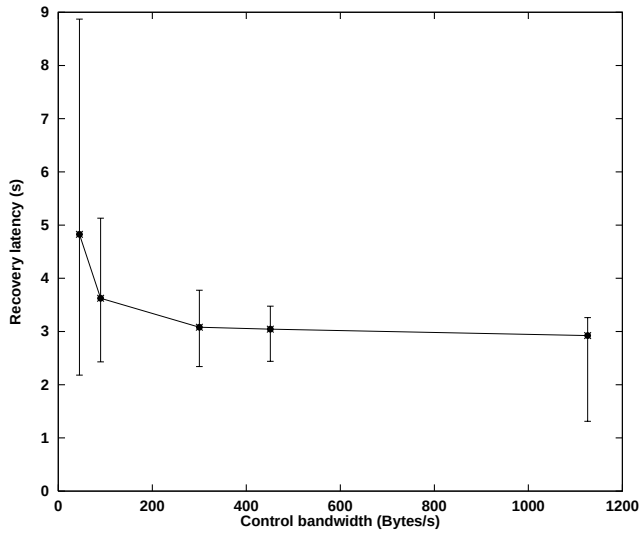


Figure 2: Tradeoff between control session bandwidth and time to discover a fault

and switches to receiving information from the new archive agent.

The time to detect and recover from a fault depends on the periodicity of client and server announcements. If a client announces once every T time units, it will take kT time to detect a fault, for some small value of k. Figure 2 shows the tradeoff between the time to detect a fault at the server side and the control session bandwidth. For these measurements we set $k = 4$, so when 4 announcements are missed, a fault is registered.

The three required parameters of control session id, presentation id, and offset are all that is required for most playback operation. However, there are cases, such as fast-view, or skimming, where other parameters will be required. In the case of fast-view, the playback rate, or scale is altered. To play back the presentation twice as fast, the scale should be set to 2. These optional parameters are usually not included in announcements. If the server does not receive values for these parameters it uses well-known defaults. For instance, the default scale is 1. However, if the client wishes to set one of these optional parameters, it must include the new value in the message, and continue to include this parameter in announcements until the end of the session, or until the session should return to the default value. This is necessary for two reasons. First, since SSAC is run over an unreliable data transmission protocol, the client needs to continue sending the new parameter until it has verified that the server has received it. This verification comes when the client receives a server announcement containing the parameter's new value and incremented offset field. However, even after receiving verification, the client needs to continue announcing this parameter so that in case of an archive agent fault, a new agent can be updated with the current state, and begin streaming.

3.2 Multi-client SSAC

In the previous section, we described the basic workings of SSAC using a single client example. In this section, we describe how the protocol is used when multiple clients control a playback session. Since the means of contacting the archive service is multicast, allowing multiple clients to share a control session requires only a few slight changes to SSAC. Four main problems need to be addressed in adjusting the basic SSAC mechanisms to multi-client operation: specifying client groups, associating control channels with groups, limiting control bandwidth, and performing state changes. We address each issue in turn.

3.2.1 Specifying Groups

The effectiveness of using IP multicast for data and control depends on the ability to group multiple end clients together. Two different scenarios exist for multiple client control: explicit and implicit client groups. An *explicit* group of clients is organized to view a presentation together, under group control. If one group member performs a seek or pause operation, the other members wish to see that operation reflected locally. An *implicit* group is composed of clients who happen to be viewing the same playback session at the same point in time. Servers may coalesce clients together in order to improve efficiency. However, if members of the merged group perform seek operations, they should be split into new groups. As we will see, the correct policy for each group can be achieved by setting the control session id field of SSAC.

To form an explicit group, all members of the group need to use the same control session id (CSID) value in their SSAC announcements. This will cause them to all be updating the same entry in the server's state table. The CSID for an explicit group can be any numeric or text string which will be unique, at the server, for the life of the control session, for example, a group member's e-mail address. This CSID is propagated to the group members through some separate means, such as *sdr* [HJ97]. In addition, group members need to share a starting time for their playback session.

Implicit groups can also be formed by using a shared CSID. This control session id can be constructed from the unique presentation name, PSID, and some timestamp which indicates the position in time within the presentation. We use a timestamp which indicates the wall clock time when the session would have started playing from offset 0. Thus, if the session began playback from offset 50 at time 10050, the start timestamp (STS) would be 10000. This timestamp remains constant throughout the session unless a seek operation occurs. Using these two fields to create the CSID handles implicit grouping of clients, and ensures that only clients viewing the same data stream will be

grouped into a logical control session. When a client performs a seek, that client's PSID:STS will change, causing the client to leave the group. If another group is watching a presentation at the new position, the seeking client can join this group. If no other group exists with the new CSID, then a new archive agent is launched.

3.2.2 Limiting Control Bandwidth

Since multiple clients are participating in each control session, it is important to set the rate of announcements correctly. The rate at which each group member announces is adjusted as the group size grows, in a mechanism similar to that used in RTCP [SCFJ96]. The goal is for each of n members to announce at a rate of p/n, where p is the announcement period. As we discussed in Section 3.1, the length of the announcement period determines the length of time before a fault is discovered and corrected.

Limiting a client to announce keep-alive messages at a rate of p/n is no sacrifice, since the client benefits from all other group member's announcements. However, if a client wishes to perform a seek operation, this message should not be limited to the usual announcement cycle, or the response time would be quite poor. We propose that clients be allowed to immediately send seek messages when needed. We do not believe this will have a major impact on control bandwidth since seek operations are relatively infrequent.

3.2.3 Performing State Changes

As we discussed in Section 3.2.1, state changes should be reflected to group members differently depending on whether the group is explicit or implicit.

Implicit groups reflect parameters in their control session id. If a member wishes to change a parameter, for instance to perform a seek operation, it is reflected by a change in the CSID. This causes the client to leave its current group and join or start a different control session. If another group is watching a presentation at the new position, the seeking client can join this group. If no other group exists with the new CSID, then a new archive agent is launched. Depending on the server workload, various policies could be used to associate users into groups. For example, programs may only be allowed to begin on the hour and half hour. If a client requests a program at an intermediate time, that request will be denied. Or, clients could be coalesced into groups by the server, through a server message which requests the client to switch to a new channel.

Explicit groups wish the state changes of one member to be reflected to the entire group. When a member attempts to perform a state change the resulting state update message is multicast to the entire group. The state change has taken place when the server announces the new state. Since all clients are allowed to make state changes, it is possible to have race conditions. Two state changes could be announced at the same "time". Members of the group would receive these messages in different orders depending on their position in the network. For this reason, if two state change messages collide, it is up to the server to decide which action to perform. Clients consider state change messages to be provisional until a server reflection is heard.

4 Implementation

We have implemented the core SSAC protocol in a distributed archive server which we call MARS (**M**ASH **AR**chive **S**erver), and a client application, the MARS rover. We utilize the established MBone tools *vic* [MJ95], *vat* [JM] and *mediaboard* [RT96] for transmitting content. We have recorded around 20 Gigabytes of content from MBone lectures, research group meetings, and local classes, and provide access to these sessions through the MARS system.

4.1 Client Application

A screen shot of our client application, MARS rover, is shown in Figure 3. The client can run as a helper application from Netscape. We have constructed a web page which lists presentations which are available for playback. Clicking on a presentation causes rover to be launched, already configured with the server multicast address, and the presentation id. The client immediately begins announcing using the SSAC protocol. When the first response from the server arrives, the client configures itself to listen on the indicated data sessions, and begins displaying data.

The MARS rover was constructed using objects from the MASH toolkit. The MASH toolkit [MBKea97] is an array of composable multimedia networking and application objects. High performance objects, such as media encoders, decoders, recording and playback objects are written in C++. These objects are then glued together using the Tcl scripting language, along with Tcl/Tk components for user interfaces, parsers, and policy components. Along with our SSAC implementation, the MARS rover includes code taken from MASH vic and MASH vat, allowing us to display audio and video data within the client application without reimplementing any of these difficult objects.

4.2 Archive Server

In order to make the best use of the scalability and fault recovery features of the SSAC protocol, an appropriate server architecture and implementation is required. An ongoing research effort at Berkeley is developing a service toolkit framework, AS1, which uses an announce/listen soft state protocol for launching service agents [AMK98]. This fits in well with the SSAC protocol design. Using AS1 we have built an archive server which utilizes SSAC messages to automatically restart failed archive agents. The AS1 framework is also implemented using the MASH toolkit.

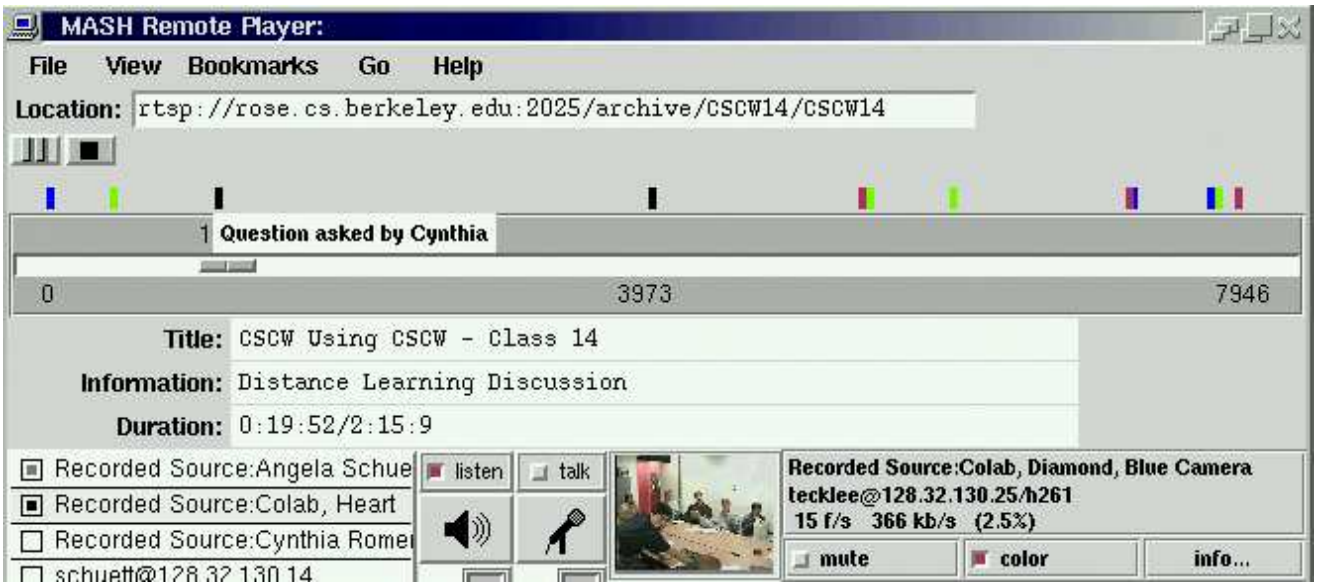


Figure 3: MARS rover user interface

The basic system architecture of AS1 is shown in Figure 4. The server is constructed out of a number of host machines, all listening to a single multicast address. Each host machine runs a **host manager** (HM) process. This process is responsible for listening to control traffic and launching **archive agent** (AA) processes when needed. See the AS1 paper [AMK98] for details on how a minimum number of host managers can be maintained in the system at all times. The AS1 system allows the set of member host machines to be completely fluid. The entire service can migrate to a new set of machines within a very short time period. In our server implementation, the host machines are on the same local network, and have access to the same disk system.

Each host manager maintains a state table matching control session ids to archive agent ids. In keeping with the soft state philosophy, this state must be updated periodically, or it will expire. Arrival of a message with a CSID which is not in the state table causes the host manager to attempt to launch an archive agent. In order to avoid launch floods, where multiple archive agents are launched for each request, timers and multicast damping are used. When a host manager attempts a launch, it first sets a randomized timer. After the timer expires, the manager will launch a new archive agent, which immediately announces its state binding to that CSID. Other host managers with launches pending for that CSID will cancel their launches. For information about other heuristics which are used to handle launch collisions, see [AMK98].

If an archive agent aborts, the soft state mapping which binds that agent to a CSID will expire. After the state expiration, the next announcement with that CSID will be treated by host managers as a new control session, and the host managers will attempt to launch a new archive agent. No action is required by the client to request a new server, and no special code is required at the server to restore state or enter a special fault recovery mode.

This architecture offers some unique advantages as the basis for an archive server. First, since there is no single point of failure, such as a centralized manager, the system is more robust. Since host machines can be added and removed on the fly, the system can adapt to server load. Finally, since the AS1 framework is designed to use multicast for service creation, fault recovery is automatic and efficient.

5 Evaluation

Evaluation of an archive control protocol can take place according to several metrics: functionality, scalability, and latency. By functionality, we refer both to the set of functions supported by the protocol, and also the set of functions which can be implemented on top of the protocol mechanisms. Scalability in this case refers to the additional cost of adding users to a control group, and the additional cost of adding sessions to a control channel. Latency covers the time required to join an existing control session, the time execute a state change, such as a seek operation. We provide a preliminary discussion of SSAC based on each of these metrics.

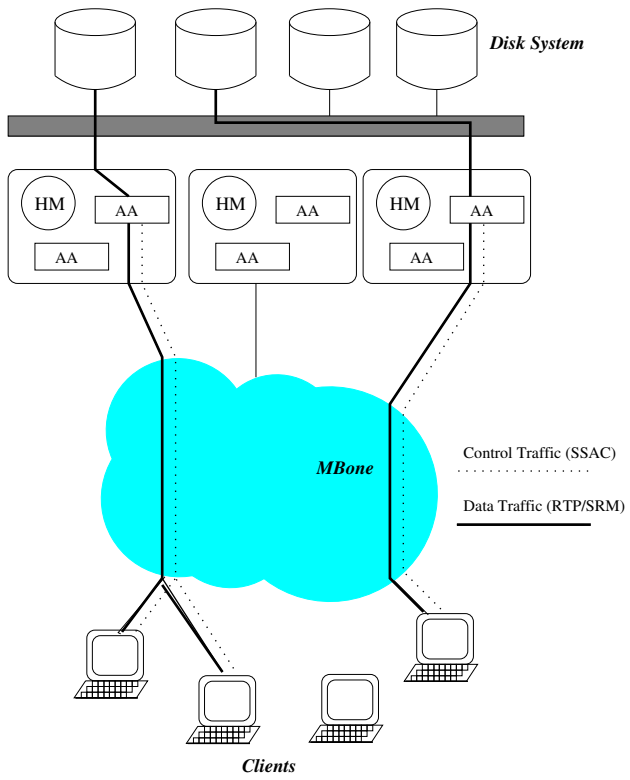


Figure 4: AS1 Architecture

5.1 Functionality

SSAC provides all of the operations needed to initiate and control a playback session. Using SSAC mechanisms, applications can request playback, perform seek operations, and set a variety of playback control parameters. SSAC also provides the ability to group together viewers into either implicit or explicit viewing groups. We believe that the combination of these mechanisms provides a platform on which to construct a rich interactive playback environment, with a variety of playback control policies, including moderated playback control, token based and lock based control schemes, or laissez-faire cooperative control.

5.2 Scalability

As stated above, in the context of an archive control protocol, scalability refers to the additional cost of adding a client to a control session.

Since SSAC was designed using the techniques of lightweight sessions, scalability of the protocol is not limited by expensive join and leave operations. When a new client connects to an existing session, no explicit server or group notification is required. Instead, clients merely start receiving state messages and repeating the state in their own heartbeat messages.

Adding a new client to a control session can have several potential costs. If the server needs to keep a list of attached clients, then each new client requires an expenditure of processing and memory at the server. Using SSAC, however, the server needs no knowledge about the number of users, only that there is at least one client attached. As long as heartbeat announcements are being received, the server need not differentiate whether they originate from a single client, or multiple clients. However, clients are required to keep track of other clients in the session. This is necessary in order to keep control bandwidth constant as clients are added to the session. To adapt the per client control bandwidth to the total number of clients, we use the mechanisms of RTCP, where clients build a soft state table listing participants. Each heartbeat message updates this table. Several papers have recently proposed schemes to improve the scalability of RTCP [RS97, Abo97], and SSAC should be able to incorporate this work.

While client heartbeat messages are rate controlled, client initiated state change messages may be sent immediately. We believe that these operations are infrequent enough that they will not affect the control session bandwidth.

5.3 Latency

According to the soft state model used by the SSAC protocol, state changes do not need to reach the entire group before they can take effect. The archive server is the arbiter and control point for state changes, and a change takes effect across the control session when it is announced by the server. In order to provide a fast response time for state changes, we advise that state change messages be sent immediately. So, the SSAC-imposed state change latency may be as low as 2 round-trip times, which is comparable to the latency of a connection-oriented protocol.

6 Related Work

Video on demand systems have been a research topic for much longer than the MBone has been part of the Internet. Several commercial systems have been deployed and tested in thousands of homes. These commercial systems have recognized the importance of fault recovery mechanisms in server implementation [NLO95, BFD97].

Recently, a large effort has been undertaken to standardize many aspects of archive control. The resulting protocol, RTSP, addresses the issues of a common connection oriented control protocol which answers the needs of a variety of current streaming application vendors [SRL98]. However, RTSP does not address multiple client control, and does not provide any explicit mechanisms for fault discovery or recovery.

The MBone VCR on Demand service [Hol97] allows multiple clients to connect to a server for session playback.

However, only one client may have control of the session, with other clients designated as listeners only. In addition, each client keeps a separate connection open to the server throughout the session, which limits the system scalability.

The Internet Multimedia Jukebox [AA98] is a research effort which is exploring the viability of grouping users together into implicit groups. The IMJ system uses the web as a means to gather requests and present play schedules. Users may request playback, but have no means to control the resulting schedule, cancel playback or perform seek or pause operations.

A number of other systems outside the realm of archive control have used the notion of soft state to achieve better performance and robustness. Networking systems for routing protocols [MRR80, DEF+96] and performance optimization [BSAK95] rely on soft state to update their tables; this allows them to continue operating in the event of partial failures and ensures that no special recovery protocols are needed to regenerate state that might have been lost in a crash. A number of soft state systems such as Bayou [DPS+] have explicitly traded consistency for availability in application-specific ways. [FGC+97] use the principles of soft state and eventual consistency heavily in their design of a general-purpose Internet service framework.

7 Summary

We have presented a new protocol for archive control that allows multiple clients to initiate and control playback of stored multimedia content in a scalable manner. Because this protocol is based completely on soft state and the announce/listen communication model, fault detection and recovery are easily achieved, in contrast to the difficulties encountered in these areas with connection-oriented protocols. The use of shared multicast control and receiver driven design allows additional clients to be added to a control session without impacting server performance or requiring additional server state.

As multicast capabilities become available to more users, large scale archives will use multicast to provide a mechanism for efficiently delivering multimedia content to clients. Our control protocol is designed to efficiently utilize the multicast delivery model to allow clients to form groups either implicitly (because they happen to be viewing the same session at the same time), or explicitly. The archive server incorporates fault detection and recovery mechanisms to ensure high availability of the system.

We have been using our prototype implementation to view our recorded weekly research group meetings. We are continuing to explore the applicability of this protocol to a variety of archival workloads, and expect to refine it as we better understand the interactions between the various pieces of the system.

8 Acknowledgments

We would like to thank Elan Amir for his help throughout the design and development of SSAC and MARS. Thanks also to James Landay for his input into the design of rover. In addition, we are grateful to Cynthia Romer, Andrew Huang and the entire MASH group for using the MARS server and client, and for their feedback. Finally, thanks go to the anonymous reviewers for their comments and suggestions.

This work was supported by DARPA contract N66001-96-C-8508, by the State of California under the MICRO program, and by NSF Contract CDA 94-01156. Angela Schuett is supported by a National Physical Science Consortium Fellowship.

References

[AA98] K. Almeroth and M. Ammar. The Interactive Multimedia Jukebox (IMJ): A New Paradigm for the On-Demand Delivery of Audio/Video. In *Proceedings of the Seventh International World Wide Web Conference*, April 1998.

[Abo97] Bernard Aboba. Alternatives for Enhancing RTCP Scalability. Internet Draft, Internet Engineering Task Force, January 1997.

[AMK98] Elan Amir, Steve McCanne, and Randy Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. Submitted for publication, 1998.

[BFD97] William J. Bolosky, Robert P. Fitzgerald, and John R. Douceur. Distributed Schedule Management in the Tiger Video Fileserver. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, October 1997.

[BNKL98] P. Basu, A. Narayanan, R. Krishnan, and T.D.C. Little. An Implementation of Dynamic Service Aggregation for Interactive Video Delivery. In *Proc. SPIE - Multimedia Computing and Networking*, 1998.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance Over Wireless Networks. In *Proceedings of the first ACM Conference on Mobile Computing and Networking*, Berkeley, CA, November 1995.

[Dee91] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.

[DEF+96] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. An Architecture for Wide-area

Multicast Routing. *IEEE/ACM Transactions on Networking*, 4(2), April 1996.

[DPS+] A. Demers, K. Peterson, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing Among Mobile Users.

[DSS94] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-demand Video Server with Batching. In *ACM Multimedia*, 1994.

[FGC+97] Armando Fox, Steven Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier. Cluster-based Scalable Network Services. In *Proceedings of SOSP '97*, pages 78–91, St. Malo, France, October 1997.

[FJM+95] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of SIGCOMM '95*, Boston, MA, September 1995. Association for Computing Machinery.

[HC97] Mark Handley and Jon Crowcroft. Network Text Editor (NTE): A scalable shared text editor for the MBone. In *Proceedings of SIGCOMM '97*, Cannes, France, September 1997. Association for Computing Machinery.

[HJ97] Mark Handley and Van Jacobson. SDP: Session Description Protocol. Internet Draft, Internet Engineering Task Force, November 1997.

[Hol97] Wieland Holfelder. Interactive Remote Recording and Playback of Multicast Videoconferences. In *Proceedings of the Fourth International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS)*, 1997.

[HS92] Jim Hollan and Scott Stornetta. Beyond Being There. In *Proceedings of CHI '92*, 1992.

[Jac94] Van Jacobson. SIGCOMM '94 Tutorial: Multimedia conferencing on the Internet, August 1994.

[JM] Van Jacobson and Steven McCanne. *Visual Audio Tool*. Lawrence Berkeley Laboratory. Software available at ftp://ftp.ee.lbl.gov/conferencing/vat.

[MBKea97] Steve McCanne, Eric Brewer, Randy Katz, and Lawrence Rowe et al. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV)*, May 1997.

[McC92] Steven McCanne. A Distributed Whiteboard for Network Conferencing. Class report, UC Berkeley, May 1992.

[McC98] Steven McCanne. Scalable Multimedia Communication with Internet Multicast, Lightweight Sessions, and the MBone. *Proceedings of the IEEE*, 1998.

[MJ95] Steven McCanne and Van Jacobson. *vic*: A Flexible Framework for Packet Video. In *Proceedings of ACM Multimedia '95*, pages 511–522, San Francisco, CA, November 1995.

[MR97] Radhika Malpani and Lawrence A. Rowe. Floor Control for Large-Scale MBone Seminars. In *Proceedings of ACM Multimedia '97*, 1997.

[MRR80] J. McQuillan, I. Richer, and E. Rosen. The New Routing Algorithm for the ARPANET. *IEEE Transactions on Communications*, 28(5):711–719, May 1980.

[NLO95] Michael N. Nelson, Mark Linton, and Susan Owicki. A Highly Available, Scalable ITV System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, December 1995.

[RS97] Jonathan Rosenberg and Henning Schulzrinne. Timer Reconsideration for Enhanced RTP Scalability. Internet Draft, Internet Engineering Task Force, July 1997.

[RT96] Suchitra Raman and Teck-Lee Tung. Mediaboard using the Scalable Reliable Multicast Toolkit. Class report, UC Berkeley, December 1996.

[SCFJ96] Henning Schulzrinne, Steve Casner, R. Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force, Audio-Video Transport Working Group, January 1996. RFC-1889.

[Sch92] Henning Schulzrinne. Voice Communication Across the Internet: A network voice terminal. Technical Report TR-92-50, University of Massachusetts, Amherst, 1992.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Internet Draft, Internet Engineering Task Force, February 1998.

The TOPS Architecture for Signaling, Directory Services and Transport for Packet Telephony

N. Anerousis, R. Gopalakrishnan, C.R. Kalmanek, A.E. Kaplan, W.T. Marshall, P.P. Mishra, P.Z. Onufryk, K.K. Ramakrishnan, C.J. Sreenan

AT&T Labs - Research
Florham Park, NJ 07932, USA

Abstract

Packet telephony is of increasing interest in both the telecommunications and Internet communities. The emergence of packet telephony will create new services, and presents an opportunity to rethink how conventional telephony services are implemented. We present an architecture for Telephony Over Packet networkS (TOPS). TOPS allows users to move between terminals or to use mobile terminals and be reachable by the same name. TOPS users can have multiple terminals and control how calls are routed to them. TOPS allows terminals with a range of capabilities such as video, whiteboard and allows a variety of coding formats. TOPS retains the necessary information on terminal capabilities to establish appropriate communication with the remote terminal. The architecture assumes that the underlying network supports end-to-end connectivity between terminals, with an appropriate quality of service.

The components of TOPS are a directory service, an application layer signaling protocol and a logical channel abstraction for communication between end-systems. The directory service maps a user's name to a set of terminals where he may be reached. A user can control the translation operation by specifying profiles that customize how his name is mapped to this set of terminals. Terminal capabilities are also stored in the directory service. The application layer signaling protocol establishes and maintains call state between communicating terminals. The logical channel abstraction provides a shared end-to-end context for a call's constituent media and control streams, while isolating the applications from the details of the network transport mechanisms. We also introduce a simple encapsulation format for voice.

1. Introduction

The interest in packet telephony stems from the desire to integrate multiple modes of communication on a single infrastructure. Packet networks allow the integration of data and voice and provide a straightforward way to offer a variety of new communication services. Increased intelligence and more elaborate user interfaces at the terminals[1] allow increased flexibility in handling voice and other forms of communication. In addition, the combination of packet networks and enhanced terminal capabilities provide an opportunity to reconsider both the nature and the implementation of telephony services. Over time, we believe these services will evolve to support substantially enhanced human communication, with voice continuing to be an important mode of communication.

We present an architecture for Telephony over Packet Networks (TOPS) that exploits the flexibility provided by packet switching and by greater intelligence in end-systems. The key elements of the TOPS architecture are:

- A *directory service* to flexibly map user names to terminals where they can be reached.
- An *application layer signaling* protocol to allow end-systems to negotiate capabilities, establish and maintain call state, and support advanced telephony features.
- A *logical channel* abstraction and an efficient encapsulation format to insulate telephony applications from the underlying network transport capabilities.

A goal of the TOPS directory service is to make it more convenient to reach a user. A packet telephony service requires a directory service function that can translate between telephone numbers and network layer addresses, such as IP or ATM addresses. This simple functionality is enhanced in TOPS to allow callers to reach a person by *name* rather than address. Information is stored in the directory service about user names and the set of terminals where they can be reached. Callers obtain terminal addresses of users by issuing a name resolution query to the directory service. As users move between terminals, they can securely modify their record entries to reflect the locations where they can be reached. Thus, user mobility is a fundamental capability of the TOPS architecture. Since the di-

1. Throughout the paper, we use the term end-system, terminal and packet telephone interchangeably.

A longer version of this paper will appear in IEEE Journal on Selected Areas in Communications, 1st quarter 1999.

rectory service returns information about individual users, it provides a mechanism for them to control how a name resolution query is handled. Users can customize access to their information. The directory service also helps callers by providing them with a flexible set of call appearances and options. Compared to existing telephony features such as "personal phone numbers" and "intelligent 800-number" services, this approach is more flexible, and gives the user more control. TOPS terminals range from computers running telephony applications to low-cost packet telephony appliances. Terminal capabilities are stored in the directory and returned to the caller as part of name resolution so that appropriate communication resources can be setup. Fast terminal mobility is handled by interfacing the directory service to a *terminal tracking server* (TTS) that deals with mobility, thus isolating the user from its complexity.

Traditional telephone signaling was designed to support relatively simple terminals. The functions of routing, number translation, connection establishment, call control, terminal control, usage recording, and advanced feature support are all implemented within the network and are tightly intertwined. As a result, the complexity of the signaling software, the reliability requirements of this integrated system, and interactions among features make it difficult to deploy new services. In contrast, packet networks already provide connectivity efficiently while services are implemented at the application layer. We therefore view the remaining pieces of signaling required to establish a call as application-layer functions. An *application-layer signaling* (ALS) protocol sets up and manages the necessary state at the end-systems and supports the negotiation of terminal capabilities and media types. ALS has the advantage that it can evolve to support a wide-range of functions, including multiple media, independent of the underlying network architecture. The capability negotiation function included in ALS allows for flexibility: there is no restriction to use only predefined media channels and encodings. The ALS protocol is also used to transfer and teardown calls, dynamically add or remove media channels and perform conferencing control functions.

The third element of the TOPS architecture is a *logical channel* (LC) abstraction and an efficient *encapsulation* format to insulate telephony applications from the underlying network transport capabilities. ALS uses logical channels as pipes for communicating both control and media data, so as to abstract the details of the underlying transport. LCs can be used to multiplex control and media channels onto a single shared context at the transport layer. The TOPS encapsulation format has been designed to contain only the essential overhead needed for transport of audio data, while allowing for synchronization among multiple media streams. This format avoids some of the fields, such as timestamps, that are typically included in a more general real-time transport protocol such as the Internet Real-Time Protocol (RTP) [14]. However, header extensions are used to carry additional information if required.

Packet phones in TOPS use a gateway to interoperate with the public switched telephone network (PSTN). The gateway acts as an end-system for the ALS protocol on the packet network and translates signaling messages between the packet network and the PSTN. It optionally performs media transcoding.

There are a number of related efforts to design pieces of a packet telephony service. The IETF Session Initiation Protocol (SIP) [15,16] combines user location and call establishment in a single protocol. In contrast, with TOPS a directory provides a caller with options for reaching a user and the services supported prior to call establishment. This allows for evolution of the functions of user location and call establishment somewhat more independently. The ITU H.323 protocol suite [7] defines a family of protocols for multimedia communications which, unlike TOPS, emphasizes use of an intermediate entity for signaling between terminals, has considerable protocol overhead when used for ordinary telephony, and provides limited address translation facilities. The Call Management Agent (CMA) [2] of the Voice-over-IP consortium has aims similar to those of the TOPS directory server, but does not address user or terminal mobility, or signaling issues.

Figure 1 illustrates the various components of the TOPS architecture and their relationship. Users register their location with the directory server. To place a call, a user contacts the directory server to determine the locations where the called party may be reached. The directory server may consult with a terminal tracking server to reach a mobile terminal. Data and application layer signaling are exchanged directly between end-systems.

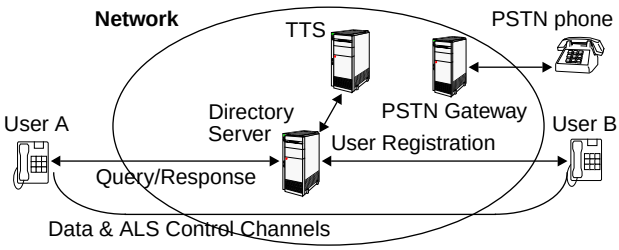


Figure 1: TOPS Components

This paper is organized as follows. Section 2 discusses the directory service. Section 3 focuses on application-layer signaling. Section 4 describes our approach to end-to-end transport of control and real-time data. Section 5 discusses interworking with the PSTN. Section 6 describes telephony services in TOPS. Section 7 presents our implementation efforts. Section 8 concludes.

2. TOPS Directory Service

Network directory services conveniently insulate users from dealing with network addresses. The Domain Name System [4] is an example of a directory service that translates host names to Internet addresses. The PSTN also uses directory services to support features such as 800 numbers, which allow a dialed number to be mapped to different telephone numbers based on certain rules or preferences. However, in order to reach the vast majority of PSTN subscribers, a caller needs to know the network address (telephone number) of the terminal nearest to the subscriber, e.g., his office phone, home phone, car phone, etc.

The goal of the TOPS directory service is to provide a dial-by-name capability: callers use a *distinguishing name* (DN) that uniquely identifies the person they wish to reach. The directory maps a query for a distinguishing name to one or more *call appearances*, each of which represents a terminal or other ways of "reaching" a user, such as a voice mail server or secretary. In this way the directory can support both user mobility (users can be reached at any terminal by updating their directory entry with the address of their present location) and terminal mobility (a user can receive calls at a mobile terminal even if its network address changes as it roams). In the PSTN, the "single number reach" capability is achieved by a network switch attempting to set up the call based on the translation of the name to a unique telephone number. Our approach allows the end-systems to use the information provided by the directory service directly, thus exploiting the enhanced capabilities of these terminals. The directory service also allows for the incorporation of query handling profiles that allow users to control which call appearances are returned by name resolution queries and when they are returned.

The directory is a distributed database. A hierarchical name space is used to derive the distinguishing names of users. TOPS allows multiple hierarchical name spaces to be defined over the database. In this way, a user record can be accessed through an X.500 distinguishing name (e.g., CN=Graham Bell, O=ATT, C=US) [19], an e-mail address (e.g., graham.bell@att.com) or a regular telephone number. The directory optionally supports traditional database functions. For example, it may be possible to look up users by last name, address, etc.

The naming hierarchy typically corresponds to administrative domains. Different network operators or large businesses own portions of the name space and operate their own servers for their part of the name space. The master copy of a user's record is always stored in one authoritative server (the user's home directory). Terminals access the directory service through a local directory server. (This server is usually the one closest to the location of the terminal and is administered by the network operator responsible for the domain to which the terminal is currently attached.) An inter-directory server protocol is responsible for locating the home directory of the called user and fetching the appropriate record. TOPS does not allow user records to be cached by local servers; only the address of the home directory server can be cached. This ensures that the user's call appearance information is always up-to-date.

2.1 Directory Contents

Figure 2 illustrates the contents of a user record in the directory database. Each record is associated with one or more DNs. The record contains information about the user (e.g. full name, address, etc.), a number of *query-handling profiles* (QHP), one or more sets of call appearances where the user can be reached, and credentials. The credentials are used by the directory to authorize a modification of the user record and by external servers to authenticate access to network services.

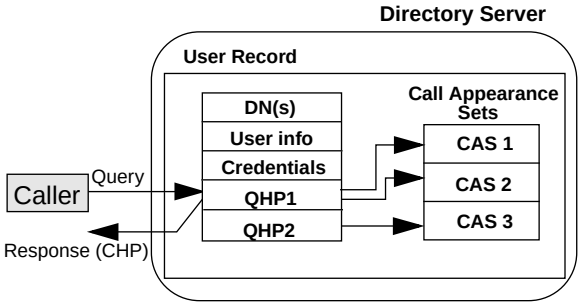


Figure 2: Structure of a user record

The query handling profile (QHP) determines the response provided to queries about this user. The QHP allows users to specify who can reach them, and where and when they can be reached. The QHP consists of a sequentially executed table of matching rules. Every rule is associated with a *call appearance set* (CAS), a group of call appearances that are returned to the querying terminal, representing the different ways in which the user can be reached. There is always a default CAS which is returned in case the QHP cannot match the incoming query to a particular rule. A call appearance set consists of a number of call appearances (see Table 1). Call appearances contain a set of fields:

- The *type* field identifies the type of the call appearance. In most cases, the type refers to a terminal or server, such as a voice mail server. In addition, a call appearance may be a pointer to another user record (DN) or indicate that the call appearance exists within another network domain, in which case a gateway must be employed.
- The *identifier* field contains optional information that can be used to identify the call appearance among others of the same type, if necessary. For example, call appearances for terminal devices may contain an electronic serial number (ESN). The ESN may be used to

Table 1: Example of a call appearance set

Type	Identifier	NLA	TLA	Terminal Capabilities	Application Info	Priority	Timeout	Comment
Terminal	ESN1	NLA1	TLA1	audio G.711	none	1	30 sec	work phone
Terminal	ESN2	NLA2	TLA2	audio G.711 video H.261	none	2	20 sec	home phone
TTS	TTS_ID	NLA3	TLA3	audio	ESN3	2	30 sec	mobile phone
DN	my_secy@ att.com					3		secretary
PSTN				audio G.711	(973) -555-1212	4	15 sec	parent's POTS phone
Server	VM_ID	NLA4	TLA4	audio	VM box #	5	10 sec	voice mail

support terminal mobility, in order to update the relevant call appearances when a terminal changes its network address.

- The *NLA/TLA* fields contain the network layer address of a terminal device or server, and the transport layer address of the TOPS application. The format of the addresses stored in these fields may differ depending on the network to which this terminal is attached. For telephony gateways, the NLA/TLA of the gateway may be statically assigned, or may be dynamically generated by the local directory server using a specialized gateway location mechanism since the address of the gateway to be used may depend on the location of the caller (or callee). The NLA/TLA fields are omitted when the call appearance contains another DN.
- The *terminal capabilities* field describes the media processing capabilities of the terminal (e.g., video, voice or fax), or the preferences of the user for a particular media type.
- The *application information* field is used to convey additional information to the calling application. For example, for a call from a packet telephone to a call appearance on the PSTN, the phone number associated with the DN is stored in the directory server, and passed to the gateway by the calling application.
- The *priority* field indicates the order in which call appearances should be attempted from the caller's terminal. Entries with the same priority are attempted at the same time (the equivalent of having several phones ring at once).
- The *timeout* field indicates the amount of time before trying the next call appearance if there is no answer. If there are multiple timeout values in a given priority the longest one dominates.
- A *comment* describes the nature and function of the call appearance.

Table 1 above illustrates a call appearance set for a user who has indicated as his first preference his office. The caller tries this terminal first, and if no reply is received within 30 seconds, tries simultaneously the next two call appearances: the terminal at the user's home and a cellular phone (accessed indirectly through a terminal tracking server (TTS) - see Section 2.3). If these call appearances fail to provide a response, the terminal attempts to contact the person's secretary by initiating a second directory query. If this fails as well, the terminal attempts to reach a PSTN phone through a gateway. If everything else fails, the caller is connected to the callee's voice mail server.

2.2 Client Interactions with the Directory Service

2.2.1 Service Subscription

Directory records are created when a user subscribes for service. At that time, he supplies the full name, billing address, and service preferences to a service representative, who assigns a DN. To maintain compatibility with domains that use numeric addressing, such as the PSTN, a unique telephone number may be assigned as an alternative DN. Cryptographic credentials for user authentication and registration are also created at this time.

A directory entry is required only for users who wish to receive calls. An "anonymous" user who does not have a directory entry with stored credentials can also place calls as long as he has provided a means of being charged for the service (e.g., credit card).

2.2.2 User Registration and User Mobility

In order for a user to receive calls, one or more call appearances must be entered into the directory record. Call appearances can be configured in three different ways:

- Statically, when the service representative creates the record,
- Off-line, when the service subscriber uses directory management software to configure his record from, say, a computer,
- Dynamically, when the user registers at a new terminal to receive calls, or moves a previously registered terminal. The user may register at a terminal that retains his information (e.g., his home phone) or to a

terminal not associated with a user (e.g., a public phone).

In the last two cases, the directory must authenticate the user by using the credential information stored in the user's record. Once authenticated, the user can update or delete existing call appearances, add new ones, or update his QHP to implement a new query resolution policy.

When a directory entry is configured off-line or dynamically, a challenge-response protocol may be used to authenticate the user. The terminal device may not have a keyboard, in which case the user may employ a smart card where the necessary credentials are stored. When the smart card is used at a public terminal, the directory can be updated automatically with a new call appearance entry for this terminal. Similarly, when the user's terminal moves to a new location, it may have been authorized beforehand to update the call appearance record in the directory with its new NLA/TLA for that user (possibly with a password). The user has the flexibility to define the desired level of security for updating the directory record. In addition, the introduction of a new call appearance in the user's record or the modification of an existing one to reflect the new address of a terminal can be specified to be valid only for a specific period of time. For example, user registration at a public phone need only be valid while the smart card is in place. Explicit user deregistration triggers updates of the relevant call appearance records. Deregistration from a public terminal triggers a deletion of the corresponding call appearance.

2.2.3 Querying the Directory

In order to call another user, the caller queries the directory to obtain the call appearance(s) of the person he wishes to call. The only information required in a query is a DN of the called party. Callers may also supply their own DNs, the types of media to be included in the call, the capabilities of the calling terminal and an indication of urgency.

The directory also supports the concept of a user persona. In this case, the caller supplies additional information that identifies the role in which a user is being called, for example professional, social, emergency, etc. Every user persona can be handled by a different QHP. The QHP contains the logic for handling a query related to the user/persona being called. The QHP may produce different responses indicating where a user can be reached based on who sent the query, the time-of-day, and the purpose of call. In addition to providing customizability, this allows the user greater control over the privacy of information compared to a traditional directory service.

The directory service does a small amount of processing to ensure that the caller's and callee's terminals are compatible. This optimization minimizes the amount of end-to-end capabilities negotiation during call establishment for some forms of communication, such as simple telephony.

2.2.4 Query Response

The response to a query is a call handling profile (CHP). A CHP is a subset of a call appearance set stored in the user's record and is determined after executing a QHP. Call appearances in the CHP are derived from call appearance records in the directory by adding or filtering out some fields.

The terminal capabilities returned with the CHP may not be the full set supported. This allows a callee to limit the terminal capabilities depending on the caller, the time of day, etc. For example, a user may restrict incoming calls from certain people to be audio only.

Fields can be added to a call appearance to implement interesting services. For example, the directory could insert an estimate of the per-minute cost for communicating with a particular call appearance. The caller could then use this information to restrict "expensive" call appearances or keep track of the accumulated cost of a call.

When the calling application receives this information, it may use the CHP directly, or it might display the CHP and await further instructions. For example, the caller may want to skip call appearances that do not meet his needs (e.g., a voice mailbox) or are expensive.

2.3 Support for Terminal Mobility

A directory record is updated either when a user moves between terminals or a terminal changes its point of attachment to the network and gets a new network-layer address (NLA). This works well if a user is registered at a terminal that does not move frequently; i.e., on the time scale of seconds or minutes. However, several performance problems occur for faster moving terminals. First, directory servers are usually optimized to support rapid lookups but relatively slow updates. Secondly, it may be inefficient to update the home server every time a terminal changes its NLA, because the directory server may be far from the current terminal location.

To avoid having to update the home directory service each time the terminal changes its NLA, we introduce an additional component in our architecture: a "terminal tracking server" (TTS). A mobile terminal acquires an NLA and the address of the nearest TTS from the network via auto-configuration. It then registers its electronic serial number and current NLA with the TTS. When a user registers at a terminal, the user record in the home directory is updated with a call appearance containing the network layer address of the TTS, rather than the address of the terminal. In response to a query, the home directory server contacts the TTS to obtain the current NLA of the terminal and returns this value to the caller.

As long as a terminal moves within the domain of a particular TTS, there is no need to update any entry at the directory server. When a terminal moves to a new TTS, it is necessary to:

1. update the entry at the previous TTS to point to the new TTS
2. update the directory server entry (for each user registered at the terminal) to point to the new TTS.

Support for terminal mobility also requires the underlying network layer to reroute ongoing calls as terminals move. We assume this is supported at the network or link-layer, for example using techniques such as ATM VC rerouting or mobile IP [10].

3. Application-layer signaling

Signaling in the telephone network supports multiple functions, including routing, number translation, connection establishment, call control, terminal control, usage recording and advanced feature support. In packet networks, routing, connectivity and resource management are already supported by existing network layer mechanisms. Moreover, intelligent end-systems reduce the need for network entities to be involved in terminal control. In our view, telephony applications communicate with each other directly, using an application-layer signaling protocol for control, and media streams for data. In this section, we describe the application-layer signaling (ALS) protocol that is used in the TOPS architecture.

The primary functions supported by ALS are establishing and maintaining call state between applications. ALS is used at an end-system to setup a call, establish and manage media channels, negotiate parameters for these channels, and to implement additional service features. The protocol has a small and simple set of basic messages augmented by optional messages through the use of header extensions. ALS supports call management for a variety of call types, ranging from simple, point-to-point voice calls, to multiparty, multimedia calls, to calls that involve other application services, such as a game or storage server. We now describe use of ALS in point-to-point telephony and multimedia calls.

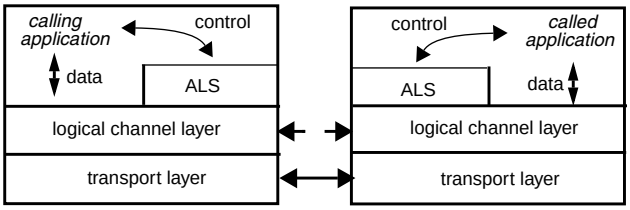


Figure 3: ALS protocol layering

All communication between the calling and called parties takes place over logical channels (LC) that are used to distinguish information flows within a call. Each call has a default LC for exchanging ALS messages, and one or more LCs for media data. Figure 3 illustrates the protocol relationship between ALS and the logical channel layer; details of logical channel operation are provided in Section 4. LCs may share a transport context while providing different semantics to the constituent flows. For example, the ALS protocol may use an LC that provides reliable and ordered message delivery, whereas an audio stream requires an LC that transports real-time data while meeting the performance constraints of that audio stream.

ALS is invoked after the calling terminal has queried the directory service to obtain one or more call appearances for the called party. The sequence of message exchanges are shown in Figure 4. An INVITE message is sent to the default LC at the address returned in the call appearance of the directory response. The message contains the distinguishing name of the called party, and optional information identifying the calling party, subject and urgency. A key function of the invitation message is capability negotiation which is used by the two parties to agree upon the number of media channels for media data, as well as their attributes such as media type, framing, encoding, priority and transport related parameters. The ALS invitation contains the set of logical channels and their attributes desired by the caller. These values are derived by taking into account the called-party capability information provided in the directory reply.

The called party application replies with a RESPONSE message that indicates which of the desired media channels are available, as well as additional information that the called party application wishes to provide to the calling party application. If no channels are available, the ALS exchange terminates. If the response indicates that a set of channels that the calling party considers necessary are available, the calling party application then obtains the network resources required to support these channels. If it succeeds, it sends a RING message to the called party; otherwise the ALS protocol terminates due to lack of resources. The RING message indicates the set of media channels that were allocated. When RING is received, the called user is alerted by a local mechanism. When the user accepts the call, a ANSWER message is sent to the calling party application. The ANSWER can contain the set of media that the user chose to accept. The called user can also indicate in the ANSWER message that he wishes to decline the call and a reason e.g., user is busy. This exchange is all that is necessary to establish call state between the parties and allow communication to commence. We note that the invitation/response performs capability negotiation and provides additional information about the called party application in a single message exchange. For example, the RESPONSE may indicate that the called party does not

wish to be disturbed. Once this initial handshake has completed, the calling party application can allocate network resources if desired. This occurs prior to alerting the called party, as in the PSTN.

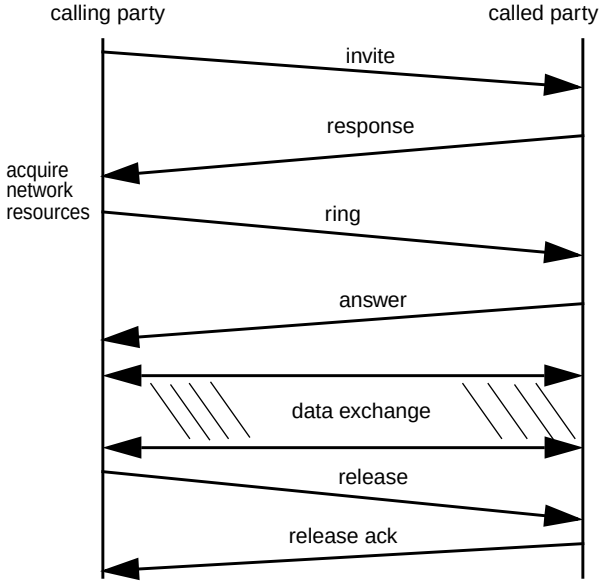


Figure 4: Fundamental ALS message exchange

Table 2 lists ALS messages. During a call, either party can issue an ALS request to modify the call state. This can be to setup a new logical channel, to renegotiate parameters for an existing logical channel, or to teardown a logical channel. The called party's reply is used to accept, reject or negotiate the proposed change. To enable this negotiation of the characteristics of the flow with the network, it is desirable that the network layer provide a convenient means for negotiation and renegotiation. An example of a lightweight network layer signaling protocol that allows end-systems to efficiently and conveniently negotiate quality of service characteristics is UNITE [18]. It supports both negotiation and renegotiation in a uniform manner..

To terminate a call, either the calling or called party must issue a RELEASE message to its peer, which in turn must send an acknowledgment. At that point the coders are stopped, logical channels are torn down, and related transport resources are freed up. Note that all ALS messages contain a calling party-generated call identifier, allowing the context of each message to be easily determined.

4. Logical Channels and Encapsulation

A goal of TOPS is that packet telephony applications are shielded from details of the underlying transport and network layers. The logical channel (LC) abstraction provides this independence by presenting a uniform communications interface to the application layer. LCs are used to carry application-layer signaling and media data, and support

Table 2: Key ALS messages

Type	Function
INVITE	invite called party to enter call; negotiate call state
RESPONSE	supply called party's reply; negotiate call state
RING	alert called party
ANSWER	notify caller the result of alerting the remote user
MODIFY	specify desired changes to the call state e.g. adding a media channel
MOD-ACK	reply, selecting acceptable changes from a modify request
RELEASE	notify intention to terminate the call
REL-ACK	acknowledge call termination; delete call state

multiplexing of related data streams onto a single transport connection. For an ordinary telephone call this can reduce the delay of call establishment by multiplexing signaling and data on a single transport connection. Multiplexing of LCs allows temporal ordering to be preserved among streams, without the need for explicit inter-stream synchronization. Use of multiplexing has implications for meeting LC quality of service requirements. The LC layer allows applications to control which LCs are multiplexed, considering their different quality of service, reliability and delivery order requirements.

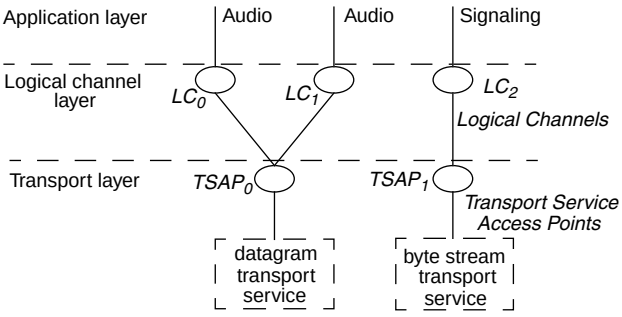


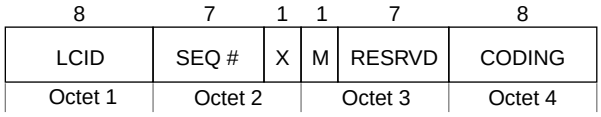
Figure 5: Mapping of logical channels to transport services

Figure 5 shows a mapping of logical channels to transport services for a packet telephony application comprising three LCs - two for audio and one for application-layer signaling. The audio LCs are multiplexed onto a transport service access point (TSAP) that uses best effort datagram transport service. The signaling LC is mapped to a reliable byte stream transport service. This illustrates the function of multiplexing application data over available transport services. LCs are mapped to TSAPs and multiplexing is achieved by allowing LCs to share a TSAP. On the receive

side, the LC layer demultiplexes data arriving on a given TSAP. An implementation is described in Section 7.

Demultiplexing data requires messages to include a header identifying their destination LC. TOPS defines a header format which is used to encapsulate audio packets, provide LC identifiers, and other packet telephony specific fields. The design is simple and does not make assumptions about the network layer used. In most packet telephony implementations, an application reads blocks of audio samples from an A/D converter, encodes them using a negotiated format, adds an encapsulation header, and arranges for transmission. The receiving application reassembles packets, performs delay adaptation, and writes decoded audio samples to a D/A converter. Since the size of the audio blocks is fairly small due to delay constraints, the TOPS encapsulation scheme contains only essential information in the header thus minimizing overhead. The encapsulation format is generic and may be used for control messages and other media data.

The format of the encapsulation header is shown in Figure 6. An 8-bit logical channel ID (LCID) allows demultiplexing of LCs that share a transport connection. This is followed by a 7-bit sequence number which is used for reassembly and delay adaptation. Each packet contains an 8-bit coding field in the header which defines the compression and coding format used by the audio data in the packet. This field allows the coding format to be changed on-the-fly without the need for a separate control message exchange. These basic fields provide the minimum information needed for audio encapsulation. Flexibility is achieved through use of a header extension bit (X), allowing an application to provide additional header information, possibly using type-length-value fields. For example, it can provide timestamps for inter-media synchronization.



LCID: Logical channel ID
SEQ# Sequence number
M: Marker bit
RESRVD: Reserved for future use

Figure 6: Packet Header Format

The encapsulation scheme also supports synchronization of generic control operations for a media LC by using a marker (M) bit in the header. As an example of the use of this bit, consider the synchronization of audio packets with related control packets, assuming that the control and audio LCs are not multiplexed together. This could be used to enable switching to a new set of Huffman tables for use with a decoder. Since the control messages may experience different delays than the audio data, a mechanism is needed for synchronization. This is achieved by setting the marker bit in an audio packet indicating to the receiver that a control packet was transmitted and that the action it specifies must be performed before the received audio data may be used. The receiver is required to acknowledge the receipt of the control packet. Once acknowledged, the sender clears the marker bit in subsequent audio packets. On detecting this transition the receiver sends an acknowledgment to indicate the end of synchronization. This handshake has the restriction that only a single control operation can be outstanding at any time. It also requires sequential delivery semantics for the messages on the control and data LCs.

5. Interworking with the PSTN

Interworking with the PSTN is provided by telephony gateways that convert between the packet stream and its voice encoding, and TDM (time division multiplexing) 8-bit mu-law format used by the PSTN.

5.1 Directory Support

For a call from a packet terminal to the PSTN, the user may have specified either a telephone number or the DN of a user, such as an X.500 name, that has a PSTN telephone number as one of its associated call appearances. In either case, the directory service returns the address of a telephony gateway rather than the address of a terminal in the call appearance. When a query contains a telephone number, the local directory server may need to map from local dialing plan numbers to a standard E.164 address. Since a gateway is likely to handle a range of telephone numbers and since there may be multiple gateways serving the same range, the directory service may use specialized mechanisms for storing and searching for PSTN telephone numbers and for locating gateways. When a query contains a DN that is not a telephone number, the application information field in the gateway's call appearance contains the telephone number to be used by the gateway to place a PSTN call.

For calls from the PSTN to a packet terminal, the subscriber dials a regular telephone number that is routed through the PSTN to a gateway. The gateway then tries to resolve the telephone number to a user inside the TOPS domain. Since the information conveyed from the PSTN is in numeric format only, it is necessary to assign an additional DN in the form of a (personal) telephone number. This number can be used from the TOPS domain as an alternative DN. Once the user record has been located in the directory, a call handling profile is returned to the gateway to complete the call. When the called number answers, the gateway bridges the connection between the two domains. The gateway generates ringback or busy tones.

5.2 ALS Support

ALS is used to establish calls between packet terminals and gateways. Each gateway acts as an ALS proxy, processing

messages on the packet network and taking appropriate action on the telephone network side. Figure 7 shows calls originating both in the packet network and the telephone network.

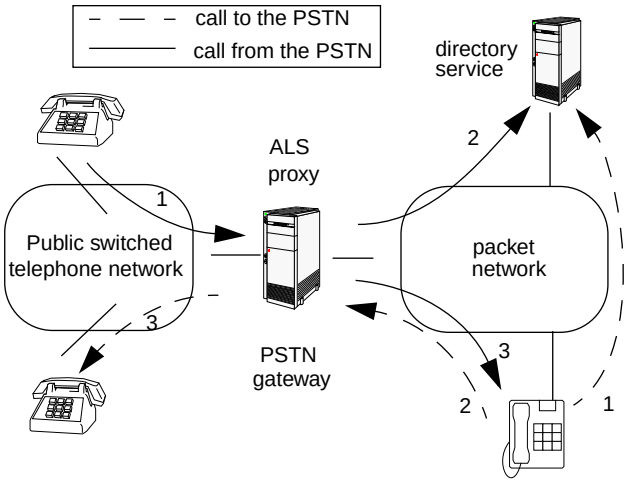


Figure 7: Calling to/from the PSTN

To establish a call the terminal initiates a regular ALS exchange with the gateway. The telephone number returned by the directory is included in the ALS invitation as a message extension. The gateway terminates the ALS protocol, translates messages to actions required on the telephone network side, responds appropriately to the called party, and maintains call state for each active call. This signaling interworking is illustrated in Figure 8 for the case of a Q.931 ISDN gateway. A gateway provides media transcoding between the encodings used by the packet telephone and the PSTN when necessary.

From the telephone network's point of view, a gateway is a termination point for a set of telephone numbers. A call originating on the PSTN and destined for one of the TOPS terminals is routed to the gateway and terminated there. Having queried the directory, the gateway issues an ALS invitation to the destination terminal, performs signaling translation and maintains call state throughout the call.

6. Implementing Telephony Services in TOPS

As an example of the migration of existing services to a packet network, consider the implementation of the familiar *call waiting* feature. In the POTS network, an indication of a second incoming call is provided through the insertion of special tones by the local telephone switch into the call that is being interrupted. This feature not only requires the local telephone switch to insert tones and detect user responses (i.e., a flash hook), but also requires it to maintain the signaling state for two simultaneous calls. The ability of a packet telephone to support multiple simultaneous streams, coupled with its processing capabilities and enhanced user interface, allows this service to be implemented entirely at the end-system. When a call is placed to a packet telephone which is already participating in a call, the called packet telephone may decide how to handle the incoming call request. It may locally generate a call waiting tone, as in current call waiting, or on packet telephones with an enhanced user interface it may create a pop-up icon with information about the call. This pop-up icon may include information about the call such as the caller's name, the topic of the call, the importance of the call, and so on. Other common POTS services such as: call forwarding, call transfer, three-way calling, and return-call may similarly be implemented entirely by end-systems.

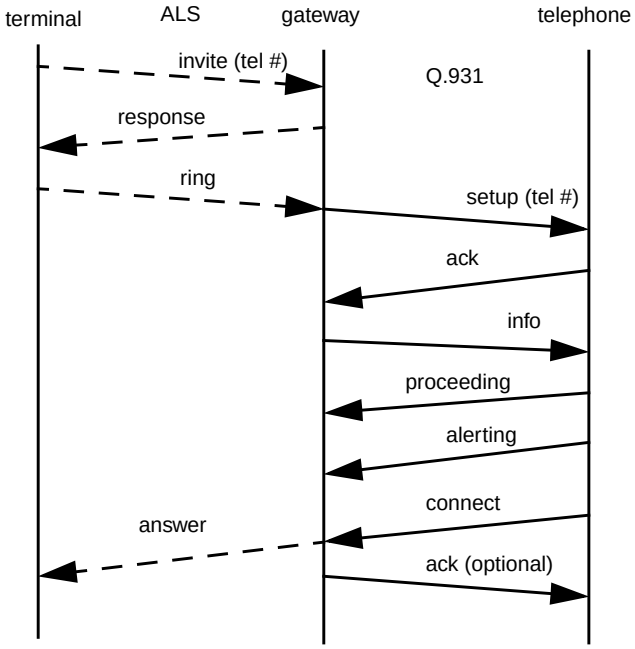


Figure 8: Example of ALS proxy interworking

Packet networks allow traditional POTS services to be enhanced without modification to core network switches or their software. An example of such an enhancement is *n-way call waiting*. In this service a packet telephone's enhanced user interface may be used to accept multiple incoming calls and allow a user to cycle through them in an arbitrary manner. We also envision more imaginative capabilities enabled by the richer signaling allowed by packet networks. For example, a meek caller may indicate in the invite request that he wishes to be notified if the called party is already in a call. With this notification, he is given the option of whether or not to interrupt the called party with a ring message, allowing him to choose to not disturb the call in progress.

Thus far we have considered traditional voice calls, but it is clear that in the future, telephone calls will expand to multimedia calls. For example, while users speak they may share a common whiteboard, view the same video clip, or

participate in a video call. This creates the need for parties to be able to discover compatible media formats when placing a call, and dynamically modify media formats during a call. Network directory servers may be used to provide information on media formats supported by a called party, simplifying the task of capability negotiation during call setup.

Packet communication is opening the door for a variety of new services. An example is that when a multimedia call is established, the caller may wish to move to a terminal with different capabilities, while saving the context of the current communication. For example a caller may answer a call in the bedroom saying "Hold-on, I'll take your call on my video phone in the living room." Such a "teleporting" [12] service requires dynamic adaptation of media formats and possibly the transfer of state. Dynamic adaptation of media formats is also required when a call is transferred, for example by a receptionist, who supports a different set of media formats than those required by the ultimate destination of the call. Multimedia calls may benefit from a "virtual hold" service in which parties involved in a call are allowed to pause for an extended time period and later resume the call where they left off. This requires saving of the call state and the ability to reinstate it once the call is resumed. To conserve network resources, connections used by media streams should be released during a virtual hold. This is conveniently achieved in packet networks where the mechanism used to modify a call's state may be separate from the mechanism used to establish end-to-end connectivity.

Many of the standard POTS services can be supported in TOPS by appropriate customization of the query handling profile for each distinguishing name, or the call handling profile returned for each query. For example, POTS supports various flavors of call forwarding in which a call placed to a certain telephone number is forwarded to another number either by default, or if a user is busy, or if he does not answer. Supporting default call forwarding in TOPS is easy; a query to the directory service for a particular user name returns the address corresponding to the packet telephone (or perhaps a voice mailbox) to which the user would like the call forwarded. Subsequently, the caller's terminal uses ALS to setup a call to this address.

Forwarding a call when a user is busy, is accomplished by having the directory service return two NLAs in response to the initial query with a call handling profile that indicates which one to try first. If ALS fails to get a busy response to the RING sent to the first NLA, it tries the next NLA.

The simultaneous ring feature provided by PBXs is emulated by having the directory service return a call handling profile that has more than one address at the same priority. ALS then attempts to establish a call to all of these addresses simultaneously. Services similar to the "advanced 800 number" service can also be emulated using the directory service. This service supports sophisticated policies for translation of an "800" number to a terminating telephone number. The directory can for example do time-of-day based routing. The QHP provides a flexible way of customizing these features by allowing individual users to specify the behavior and policies they need.

In addition to customizing the QHP, a user may control the service features he receives by suitably customizing the behavior of his telephony application. For example, when there is an incoming call to a user who is already involved in another call, ALS determines based on a user service profile whether to provide the user with an indication of the incoming call, return a busy indicator to the caller, or to provide the caller access to a voice mailbox.

7. Implementation

A packet telephony system has been implemented to gain experience with our architecture. The initial focus for voice transport is native ATM, since it provides support for quality of service and offers good delay characteristics. Control information, such as directory access and ALS, is exchanged over native ATM or classical IP-over-ATM.

7.1 TOPS Terminals

The prototype system consists of a set of packet telephones and servers interconnected by an ATM network infrastructure, as shown in Figure 9. There are three types of tele-

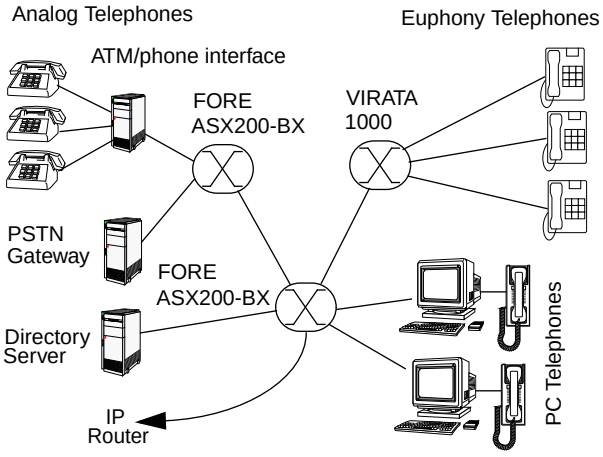


Figure 9: Current packet telephony testbed

phones: PCs running a telephone application, custom built ATM telephone appliances, and analog phones that are controlled and interfaced to ATM by a PC. A server provides a gateway to the PBX (and hence PSTN), and a directory server is used for queries to ATM addresses.

The PC telephone is implemented as an application over Linux. The hardware configuration consists of a Pentium based PC with full-duplex Gravis Ultrasound and Sound-

Blaster audio cards sound cards and a ZeitNet 155 Mbps ATM card. The application uses the Open Sound System (OSS) API [13]. It supports a variety of audio coders including G.711, G.723.1, G.726, G.728 and G.729. Both UDP as well as native ATM AAL5 are used as audio transport protocols. In addition, the software provides a graphical X windows user interface for call control, including invitation, release, call transfer, etc. A user level LC library and ALS signaling have been implemented over the Linux ATM API [1].

A custom built ATM telephone appliance has been designed and is being used experimentally. It is based on the Euphony processor [5,6] developed by AT&T Labs and LSI Logic. Euphony is a highly integrated RISC processor that incorporates signal processing functions, a 25 Mbps ATM interface, a digital audio interface, and all the system logic required to build a complete system (e.g., DRAM/SRAM controller, five DMA channels, counter timer, etc.) into a single VLSI device. The ATM telephone hardware consists of a Euphony, SRAM, Flash, A/D and D/A, UART, and a 25 Mbps ATM physical layer device. Due to its high level of integration, the entire system is packaged in a standard telephone case. The software running on the ATM telephone consists of the telephony application, VxWorks real-time operating system, and ATM signaling code. The ATM telephone augments the traditional POTS telephone user interface, which consists of a handset, a switch hook and a 12 button keypad, with two additional buttons and three LEDs. Features such as LCD screen, mouse, or keyboard may be connected to the base unit through a serial port on the back of the ATM telephone. For example, it is used for a prototype 10 inch color touch screen with a keyboard and mouse, allowing access to advanced services.

The third type of device is an off-the-shelf analog telephone connected to a PC-based ATM/telephone interface card [8]. In this arrangement audio data is handled entirely within the interface card, which provides audio A/D and D/A, ATM access and AAL functions. The card forwards off/on-hook and key press events to a control program on the PC which is responsible for generating appropriate dial tones, directory access and call signaling. Users dial using either unique user identifiers or telephone numbers which are passed to the directory server. Special key press sequences are provided for user registration and deregistration, to take advantage of user-mobility and caller-ID-based directory features.

7.2 LC and ALS Implementation

The LC layer is implemented as a user-level library and exports a synchronous receive and send API to the application. The LC layer runs on top of native ATM AAL5, UDP, and TCP. Applications use a common API to create and use LCs independent of which of the above transport protocols are used. For TCP, which is a byte stream protocol, the LC layer transparently prefixes each message with a length field to preserve LC message boundaries. Currently the LC mechanism has been implemented and tested in Linux and is being ported to Wind River's VxWorks operating system.

A subset of the ALS protocol has been implemented over VxWorks. This protocol is used by a phone application to setup calls between the Euphony ATM terminals. ALS responds to on-hook and off-hook events, performs digit collection, does directory lookup, and does call setup. The application also has modules for generating dialtone, touch tones, and error tones.

7.3 Directory Implementation

The TOPS directory server is shown in Figure 10. It is organized around a database of user records. Clients can access the directory service using different protocols (the current implementation supports both IP and native ATM). A protocol adapter module produces a directory query from a message (e.g., a UDP datagram or an AAL5 frame) and similarly constructs a response with the results of the query.

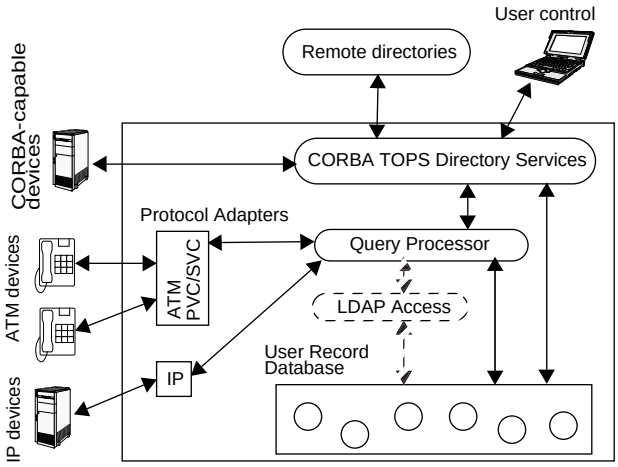


Figure 10: Directory Software Architecture

Every query is passed to a query processor that identifies a user record in the database. It uses the persona information in the query (if present) to select the appropriate QHP to execute. The query processing interface is also accessible as a CORBA service [11] for clients that have this capability.

In order to submit a query, the client uses a simple message protocol (shown in Table 3). Each message is of the form:

cmd_name attrname1=attrval1 attrname2=attrval2 ...,

where cmd_name identifies the action to be taken by the directory. Parameters are specified as attribute name/value pairs.

User profiles can be created and modified through a separate CORBA service. For convenience, a user can modify his profile using a Web browser, that loads a Java applet

Table 3: Directory Command Interface

Command	Supplied Attributes	Description
Query	Callee, Caller, Persona, Terminal Caps.	Submits a query to the directory and returns a set of call appearances that can be used by the terminal to locate the user.
Create Update	ESN, Address, Priority, Timeout, Description	Creates or updates a call appearance record when the user or a terminal moves.
Find	First/Last Name, Email, etc.	Returns a list of DNs from the local database that match the user supplied predicates (white pages service)

from the server that supports an interface for profile modification. Another set of interfaces is used to access remote directories in order to fetch user records, update call appearances, etc.

In the future we plan to integrate the TOPS user database within an X.500 directory service. The main advantage of this is that user records for TOPS can be kept together with information for other services. In this case, the TOPS directory server will access the user record through an X.500 access protocol such as LDAP [9] rather than from local disk.

7.4 Implementation of a PSTN Gateway

To interwork between TOPS phones and PSTN phones, a gateway has been implemented. It uses a PC card [3] that terminates a PSTN connection and moves the voice data to/from an ATM VC that is setup between a packet phone and the gateway.

Telephone numbers can be handled in one of two ways. When calling a packet phone from the PSTN, telephone numbers assigned to packet phones are also assigned to a gateway. Dialing a packet phone from the PSTN results in a call being setup to the gateway. The gateway consults the directory to map the dialed number to an ATM VC, and arranges for the hardware to move the voice data between the telephone circuit and the VC. Alternatively, the gateway itself is assigned a PSTN telephone number. When this number is called from the PSTN, the gateway provides a secondary dialtone to the caller. The caller enters the name of the called party via the telephone keypad. The gateway maps this name to an ATM VC and sets up the voice path as before.

To call out to the PSTN, the initiating terminal connects to the gateway and uses ALS to convey the target PSTN telephone number. The gateway initiates a voice call on the PSTN to the specified number and sets up the voice path as before. The gateway may also provide secondary dialtone, allowing the caller to dial directly.

8. Summary

TOPS enhances existing telephone services. It leverages the combination of packet networking and more sophisticated end-systems to enable a variety of creative new services. TOPS extends traditional directory services to support powerful location management functions. The directory service maps from a user's distinguishing name to one or more terminals, servers or other entities to which a call should be routed. Users have the ability to customize the translation using a profile which allows them to be associated with multiple terminals, and to screen and route calls addressed to them. By using a name as the basis for reaching a user, the TOPS architecture directly supports both user and terminal mobility.

TOPS incorporates an application layer signaling (ALS) protocol that supports call establishment, terminal capability negotiation and session management for the duration of the call. ALS supports multiple simultaneous sessions, which allows traditional services such as three-way calling to be easily implemented. ALS supports multiple media streams to be associated with a call. TOPS uses an encapsulation format that is simple and efficient for real-time transport. A logical channel mechanism allows control and media streams to be multiplexed over an single transport association, and to be managed as a single, application-layer context. For example, one or more control channels may be used to manage the constituent media streams of a single ALS context. This allows the application to be isolated from the characteristics of the underlying transport layer (such as a datagram or connection oriented transport) or network layer. TOPS only assumes that the network supports establishment of end-to-end connectivity between terminals, with the appropriate quality of service to carry the real-time media streams.

TOPS interfaces packet telephony services with the public switched telephone network through a telephone gateway. The gateway performs conversion from packet to circuit, and acts as a signaling gateway between ALS and traditional telephone signaling.

Conferencing of multiple call participants is an important service that must continue to be supported as telephony evolves to packet networks. TOPS supports a variety of conferencing modes. For details see [17].

We believe that mechanisms for authorization and usage recording must be developed that allow service providers to charge users for telephony services. We are presently exploring these issues.

References

[1] W. Almesberger, “ATM over Linux documentation”, http://lrcwww.epfl.ch/linux-atm/doc.html.

[2] O. Kahane and S. Petrack. “Call Management Agent System: Requirements, Function, Architecture and Protocol”, IMTC Voice over IP Forum submission VoIP97-010, 1997.

[3] Dialogics Corp., “More About Computer Telephony”. http://www.dialogic.com/company/3602web.htm.

[4] P. Mockapetris. “Domain names: Concepts and facilities”, IETF RFC 882, 1983.

[5] Onufryk, P. Z. “Euphony: A Signal Processor for ATM,” EE Times, pp. 54-80, January 20 1997.

[6] Onufryk, P. Z. “Euphony: An Embedded RISC Processor for Low Cost ATM Networking and Signal Processing,” Design SuperCon97: Digital Communications Design Conference, pp. C112-1 to C112-16, 1997.

[7] ITU-T. “Recommendation H.323*:* Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-guaranteed Quality of Service*”*, 1996.

[8] Inno-Media Logic Ltd., Building Telephony Switches and Systems from 1,000 to 100,000+ ports with Computer Telephony Chassis linked over an ATM infrastructure. http://www.iml-cti.com/index_doc.htm.

[9] M. Wahl, T. Howes, S. Kille. “Lightweight Directory Access Protocol (v3)”, IETF draft <draft-ietf-asid-ldapv3-protocol-09>, November 1997.

[10] P. Bhagwat, S.K. Tripathi, C. Perkins, “Network Layer Mobility: an Architecture and Survey”, IEEE Personal Communication, 3 (3), June 1996.

[11] Object Management Group, “The Common Object Request Broker Architecture and Specification”, Rev. 1.2, Dec. 1993.

[12] T. Richardson, F. Bennett, G. Mapp, A. Hopper, “Teleporting in an X Window System Environment”, IEEE Personal Communications Magazine, Vol. 1, No. 3, pp 6-12, September 1994.

[13] Open Sound System (OSS/Linux). http://www.4front-tech.com/linux.html.

[14] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. “RTP: A Transport Protocol for Real-Time Applications”, IETF RFC 1889, January 1996.

[15] H. Schulzrinne, “A comprehensive multimedia control architecture for the Internet”, 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 97), May 1997.

[16] M. Handley, H. Schulzrinne, E. Schooler. “SIP: Session Initiation protocol”, IETF draft <draft-ietf-mmusic-sip-04>, November 1997.

[17] N. Anerousis, R. Gopalakrishnan, C.R. Kalmanek, A.E. Kaplan, W.T. Marshall, P.P. Mishra, P.Z. Onufryk, K.K. Ramakrishnan, C.J. Sreenan. “TOPS: An Architecture for Telephony over Packet Networks”, IEEE Journal on Selected Areas in Communications, to appear, 1st quarter 1999.

[18] G. Hjalmtysson and K.K. Ramakrishnan. “UNITE: An Architecture for Lightweight Signaling in ATM Networks”, *Proceedings of the IEEE Infocom ‘98*, San Francisco, CA, April 1998.

[19] ITU-T, “The Directory: Overview of Concepts, Models and Service”, Rec. X.500, 1993.

Routing Traffic with Quality-of-Service Guarantees in Integrated Services Networks

Qingming Ma
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134
qma@cisco.com

Peter Steenkiste
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
prs@cs.cmu.edu

Abstract

Transmission of interactive multimedia streams requires stringent Quality-of-Service (QoS) guarantees in delay, delay jitter, and bandwidth, which imposes strict resource constraints on the paths being used. While the general problem of finding a path that meets multiple QoS constraints is NP complete, we have shown that such a path can be found in polynomial time if the network service disciplines are rate proportional. However, two important issues remain unsolved. One issue is how to select among all feasible paths an efficient one that achieves high network throughput. We identify four optimality criteria and for each of them we propose a polynomial algorithm that selects paths that meet the criterion. Our simulation results show that, while the widest-shortest path performs best for heavy loads, the shortest-delay path has a performance edge for light loads. Surprisingly, the path that requires the reservation of the smallest amount of bandwidth performs the worst for both heavy and light loads. We also evaluate the influence of these routing algorithms on the throughput of best effort traffic. Our results show that, using the shortest-widest path algorithm for routing guaranteed traffic can result in relatively low throughput for best-effort traffic, while the other three algorithms result in comparable performance. The second open issue is that, although these algorithms are polynomial, they have much higher computational complexity than the Bellman-Ford algorithms used in today's Internet. We propose approximation algorithms that have a running time within a constant factor of that of the Bellman-Ford algorithm without sacrificing network throughput. Overall, the approximation algorithms that select the shortest-delay path results in consistently high network throughput.

1 Introduction

The advent of broadband networking technology makes it feasible to support real-time multimedia applications such as video conferencing and Internet telephony. These applications have stringent Quality-of-Service requirements (QoS) in terms of bandwidth, delay, delay jitter, and loss rate. To provide guaranteed QoS, new network service models have been defined in both the IP and ATM community [30, 1] and new resource management mechanisms that support these service models are being developed [6, 4, 37, 24, 35, 33, 10]. QoS routing is one such mechanism. It has two goals: selecting feasible paths that meet QoS constraints and making efficient utilization of the network resources. To achieve these goals, both routing protocols and the routing algorithms must be developed. While QoS routing protocols have received considerable attention in both the ATM forum and the IETF QoS Routing group, QoS routing algorithms are not yet well understood.

Selecting a feasible path that meets multiple QoS constraints (e.g., delay and delay-jitter) is in general computationally intractable [7, 34]. However, this result is obtained under the assumption that QoS constraints (e.g., delay, delay jitter, and bandwidth) are not related and that queueing delay is known *a priori* regardless of the burstiness of the traffic source and the amount of bandwidth to reserve. In our earlier study [22], we observed that that QoS constraints are interrelated in a way that is determined by the network scheduling discipline and that queueing delay is determined by the amount of bandwidth to reserve and the burstiness of the traffic source. By exploiting these dependencies in a network whose service disciplines are rate proportional (e.g., Weighted Fair Queueing), we developed polynomial algorithms for selecting feasible paths with delay, delay jitter, and bandwidth constraints. The resulting routing algorithms iterate the Bellman-Ford (IBF) algorithm over the different bandwidth values of all links in the network.

In this paper, we move further by addressing two important issues that have not been well understood in QoS routing. First, the IBF algorithm presented in [22] finds a feasible path as long as it exists, but more than one feasible path is often available. This raises the question of which feasible path to select to achieve high network throughput. We identify four optimality criteria that can be used when selecting an efficient path. These criteria strike a different balance between two competing goals: minimizing the resource consumption of individual flows and spreading the network load. Based on the IBF algorithm, we propose routing algorithms that select paths with these different optimality criteria. We evaluate these al-

gorithms through an extensive simulation study, using realistic topologies, traffic models, and periodic routing information distribution. Our evaluation uses the call blocking rate as the primary performance metric. However, in a network that supports both guaranteed traffic and best-effort traffic, the choice of a routing algorithm for higher priority guaranteed traffic can influence the throughput of lower priority best-effort traffic, since guaranteed sessions can take away resources used by ongoing best-effort traffic. The throughput for best-effort traffic becomes a more important performance metric when the volume of guaranteed traffic is low and the call blocking rate is not an issue. Our results show that while the widest-shortest path algorithm performs well when the load is heavy because it conserves resources, the shortest-delay path algorithm has a performance edge when the load is light because it distributes the network load more evenly than other algorithms. Surprisingly, the algorithms focusing on minimizing bandwidth reservation perform the worst.

The second issue we address in this paper is that of the computational cost of the IBF-based algorithms: even though they are polynomial, they considerably more complex than the Bellman-Ford algorithm. We develop approximation algoritms with a computational complexity within a constant factor of that of the Bellman-Ford algorithm. Our simulation results show that these approximation algorithms run almost ten times faster than the original algorithms for an MCI Internet backbone topology. Most importantly, this speedup in running time is obtained without sacrificing the network throughput. Overall, the approximation to the shortest delay path algorithm performs consistently well for different topologies and traffic loads.

These results demonstrate that QoS routing is both desirable and feasible. It is desirable because using carefully selected routes can significantly reduce the blocking rate and improve throughput. QoS routing is also feasible because routes that meet QoS constraints can be selected using algorithms with reasonable cost, i.e. within a constant factor of the execution times of the Bellman-Ford algorithm.

In the remainder of this paper, we first briefly review the basic IBF algorithm of [22] that selects feasible paths for guaranteed traffic in polynomial time (Section 2). In Section 3 we identify optimization criteria and present corresponding routing algorithms. Sections 5 and 6 examine the performance of these routing algorithms, and Section 7 present and evaluate our approximation algorithms. Related work is discussed in Section 8 and we summarize in Section 9.

2 Selecting Feasible Paths

As discussed in [22], by exploiting the relationship between QoS constraints introduced by the rate-proportional service discipline, we are able to develop polynomial QoS routing algorithms that select paths subject to delay, delay jitter, and/or buffer space constraints. In this section, we summarize the main results obtained in [22].

2.1 Rate-proportional Service Disciplines

A variety of service disciplines aimed at providing per-flow performance guarantees have been proposed. Examples include Virtual Clock (VC) [36], Weighted Fair Queueing (WFQ) [6, 24], Worst-case Weighted Fair Queueing (WF^{2}Q) [2], Self Clocked Fair Queueing (SCFQ) [9], and Frame-Based Fair Queueing [32]. These service disciplines ensure that flows sharing an output link get their proportional shares of the link capacity. As a result, the end-to-end queueing delay, the delay jitter, and required buffer space in each network node is determined by the bandwidth reserved for the flow and the characteristics of the traffic source.

Given a path $\mathbf{p}$ of n hops with link capacity C_i at hop i, and a traffic source constrained by the leaky bucket $\langle \sigma, b \rangle$, where σ is the average token rate and b is bucket size, provable bound on end-to-end delay, delay jitter, and buffer space requirements at the h-th hop are given by [35, 33]:

$$D(\mathbf{p}, r, b) = \frac{b}{r} + \frac{n \cdot L_{\max}}{r} + \sum_{i=1}^{n} \frac{L_{\max}}{C_i} + \sum_{i=1}^{n} \text{prop}_i \quad (1)$$

$$J(\mathbf{p}, r, b) = \frac{b}{r} + \frac{n \cdot L_{\max}}{r}. \quad (2)$$

$$B(\mathbf{p}, b, j) = b + h \cdot L_{\max}. \quad (3)$$

where r ($r \geq \sigma$) is the amount of bandwidth to be reserved, $L_{\max}$ is the maximal packet size in the network, and prop_i is the propagation delay. These equations relate delay, delay jitter, and buffer space to the flow bandwidth and the leaky bucket characterizing the traffic source.

2.2 Iterative Bellman-Ford Algorithm

The main results of [22] are summarized in the following theorem. We refer the reader to [22] for detailed proofs and algorithms.

Theorem 1 *The QoS routing problem of finding a path with delay, delay-jitter, and/or buffer space constraints is solvable in polynomial time. If the bandwidth to be reserved is known* a priori*, a slightly modified version of the Bellman-Ford algorithm can solve it in $S = O(m \cdot L)$, where m is the number of nodes and L the number of links in the network. If the bandwidth to be reserved is unknown, an algorithm that iterates the modified version of the Bellman-Ford algorithm can solve it in $E \cdot S$, where E is the number of all possible residual bandwidths of links in the network.*

We briefly present the intuition behind the IBF algorithm. The simplest case is when the amount of bandwidth r to reserve is known *a priori*. In that case, a feasible path can be found using any shortest path algorithm by using the following link cost function:

$$d(j) = \frac{L_{\max}}{r} + \frac{L_{\max}}{C_j} + \text{prop}_j \quad (4)$$

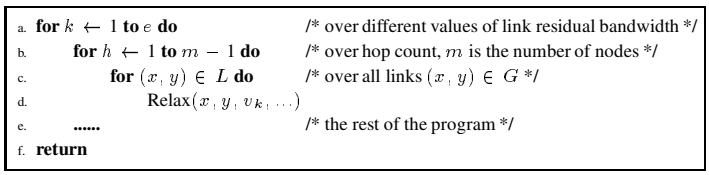


Figure 1: A sketch of the pseudo-code for the IBF algorithm

In practice, the bandwidth r to be reserved is not known *a priori* and should be determined by the routing algorithm. This creates a dependency between r, the delay, and the path being selected, and a simple shortest path algorithm no longer works. However, it is clear that the path with the shortest delay will be feasible, if a feasible path exists. Such a path requires reserving the maximum reservable bandwidth (mrb) on the path

$$\mathbf{mrb_p} = \min\{R_j \mid j \in \mathbf{p}\},$$

to achieve the minimum delay, where **p** is the path and R_j the residual bandwidth, i.e., the amount of reservable bandwidth of link j. Note that the $\mathbf{mrb_p}$ must be equal to R_j for some $j \in \mathbf{p}$,

This observation leads to an algorithm for finding a shortest-delay path, and thus a feasible path if one exists. The algorithm simply iterates over all different residual link bandwidths R in the network. In each iteration, it assumes that R is the amount of bandwidth that needs to be reserved and finds the shortest-delay path using the cost function of Equation 4. If none of these shortest-delay paths found in each iteration is feasible, no feasible path exists. Otherwise a solution can be found. Using Equation 1, the algorithm determines how much bandwidth actually needs to be reserved. If the Bellman-Ford shortest path algorithm is used in each iteration to find the shortest-delay path, the algorithm is called the Iterative Bellman-Ford algorithm, or IBF algorithm. A sketch of the pseudocod for the IBF algorithm is shown in Figure 1, where the procedure Relax$(x, y, v_k, \ldots)$ updates the paths if the distance of the current shortest path from the source to x plus the distance between x and y is smaller than the distance of the current shortest distance path from the source to y. If not only delay, but also jitter and buffer size constraints have to be met, a similar algorithm can be used. The only difference is that hop count constraints have to be added to the algorithm to ensure the jitter and/or buffer size constraints.

We do not consider in this paper the more general case in which a different amount of bandwidth can be reserved on each link. In that case, QoS routing is NP-complete [20].

3 Selecting Efficient Paths

In practice, more than one feasible path is often available. This raises the question of what path to use to achieve higher network throughput (or lower blocking rate). We propose four different optimality criteria that can be used in path selection and present polynomial routing algorithms that select paths with these different optimality properties.

3.1 Optimality Criteria

What path selection criterion will minimize the call blocking rate for guaranteed traffic is an open question. A first goal could be to minimize the resource utilization of selected paths. For guaranteed traffic, this can be achieved by selecting the feasible path with the **minimum hop count** or that requires reserving the **minimal bandwidth**. An alternative goal is to try to distribute the load evenly through the network. Since mrb can be viewed as an indication of the load conditions (for guaranteed traffic), selecting the **minimum load path**, i.e. the path with maximum mrb, would emphasize distributing load in the network. Finally, we can select a path with **minimum end-to-end delay**, if the mrb is reserved (this does not imply that the mrb has to be reserved). This last criterion is motivated by the form of Equation 1: it suggests that a path with low end-to-end delay will in general have few hops and a high mrb, so this criterion strikes a balance between minimizing resource use and distributing load.

Different combinations of these optimality criteria lead to the following path selection criteria:

- **Minimum-bandwidth path**—a feasible path that requires the reservation of the minimum amount of bandwidth. If there is more than one choice, the one with the minimum hop count is selected.
- **Widest-shortest path**—a feasible path with the minimum hop count. If there is more than one path with the minimum hop count, the one with the maximum reservable bandwidth is selected.
- **Shortest-widest path**—a feasible path with the maximum reservable bandwidth. If there are several such paths, the one with the minimum hop count is selected.
- **Shortest-delay path**—a feasible path giving the minimal end-to-end delay if the maximal reservable bandwidth is reserved. If there are several such paths, the one with the minimum hop count is selected.

Clearly other path selection criteria are possible. We selected these algorithms because they represent a broad spec-

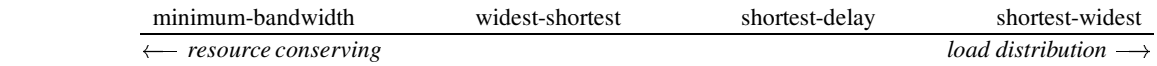


Figure 2: Tradeoffs for path selection criteria

trum of different tradeoffs between resource conservation and network load distribution as is shown in Figure 2.

3.2 Algorithms

We present algorithms that can identify paths that meet the above optimality criteria. Recall that the IBF algorithm has two levels of iterations: the outer loop (line (a) in Figure 1) over different values of link residual bandwidth and the inner loop (line (b) in Figure 1) over the hop count. For every value of link residual bandwidth and every hop count, a feasible path may be found. We store all these paths obtained in an array as is shown below. We will now show that paths corresponding to the different optimal criteria can be found from the feasible paths recorded in the table.

bandwidth value	hop 1	$\cdots$	hop $m-1$
v_1	$\mathbf{p}_{1,v_1}$		$\mathbf{p}_{m-1,v_1}$
$\cdots$			
v_e	$\mathbf{p}_{1,v_e}$		$\mathbf{p}_{m-1,v_e}$

The *shortest-delay* path can be directly found from the table after all iterations have been completed. To find a *minimum-bandwidth path*, we compare all feasible paths in the table, and select the one requiring the reservation of the minimum bandwidth (use Equation 1). We claim that there is no other path requiring less bandwidth that meets the delay bound. Let us assume there is such a path p_x, and let r_r and h be the mrb and hop count of that path. Let us compare p_x with the path p_t in the entry with hop count h and bandwidth r_r. Since both paths have the same hop count and residual bandwidth, they will have identical values for the first two terms in Equation 1. However, since p_t has a lower delay, it will have smaller values for the last two terms than p_x, which means that p_t can meet a given delay bound by reserving less bandwidth r. This means that p_x must be equal to p_t.

To find a *widest-shortest path*, we can search the table column by column in increasing order of hop count, searching each column in decreasing order of bandwidth values. The first feasible path found is a widest-shortest path. Similarly, a *shortest-widest path* can be found by searching the table row by row.

The shortest-delay path and minimum-bandwidth path algorithms require that the entire table is constructed, *i.e.*, all iterations have to be executed. In contrast, the shortest-widest path and the widest-shortest path algorithms can be implement more efficiently by ordering the different bandwidth values in decreasing order. To select a shortest-widest path, the algorithm can terminate when the first feasible path is found. To select a widest-shortest path, in addition to ordering the residual bandwidth values, we can swap the loop order in lines (a) and (b) in Figure 1. With this slight modification, the first feasible path found is a widest-shortest path. Figure 3 gives pseudocode for the widest-shortest path algorithm.

4 Simulator Design

Our evaluation is based on a session-level event-driven simulator, which has also been used in studying routing best-effort traffic [23] and traffic requiring bandwidth guarantees [21]. The simulator simulates the start and termination of sessions, and does route selection, admission control, and resource reservation. To increase the level of realism, the simulator does packet-level simulation of connection setup and tear-down, and of routing information distribution. For connection setup and tear-down this means that the latency of these operations is modeled. For routing information distribution, this means that we include both the effect of routing update overheads and of delayed state information on performance.

The simulator uses link-state routing, and routing information distribution is implemented using a simple reliable flooding protocol similar to that used in PNNI and OSPF. The default period is 30 seconds. It is easy to show that static routing, while simple, can give very poor performance for guaranteed traffic, so we focus on dynamic routing. As a result, the routing information includes both static topology information and dynamic residual bandwidth and buffer space. We assume on-demand routing with a routing cost of 10 msec, and connection set up and tear down costs of 3 msec/how and 1 msec/hop, respectively. Since the connection life times are fairly long, these costs have little impact on the results.

We used two topologies in our simulation: an MCI topology and a local area switched cluster topology (Figure 4). The link propagation delay is calculated according to the physical distance between nodes for the MCI topology.

4.1 Traffic Loads

Our traffic load consists of two classes of traffic: guaranteed sessions and best-effort sessions, which are described below. Clearly, many different types of traffic loads could be considered. Our evaluation shows that low bandwidth connections (e.g. voice or small best-effort data transfers) have little impact on performance, so we focus on higher-bandwidth video connections and high bandwidth best-effort traffic.

The traffic entering the network is split between the two traffic classes according to a predetermined ratio. The traffic

```
Widest_Shortest_Path(G : GRAPH, s : NODE, d : NODE, d : real, j : real, b : NODE → int)
1.  r_vec ← Sorting(L)                          /* sort R_i (i ∈ L) in decreasing order, delete duplicates */
2.  E ← lenght of r_vec                         /* the number of different link residual bandwidth */
3.  for v ∈ V do                                /* for all nodes in G */
4.      N_v ← ⌊(b(v) − b)/L_max⌋                 /* hop count bound for node v by buffer space */
5.      max_hop = max(max_hop, N_v)
6.  m ← min(|V|, max_hop)                       /* |V| is the number of nodes in G */
7.  Initialize-Single-Source(G, s, E, r_vec)    /* initialization*/
8.  for h ← 1 to m − 1 do                       /* iteration over hop count */
9.      for k ← 1 to E do                       /* iteration over different values of link residual bandwidth */
10.         if (n_k ≥ h) then                   /* if h meets the hop count bound set by j */
11.             for (u, v) ∈ L do               /* for all links in G */
12.                 Relax(u, v, k, l_k, h, N_v)
13.         if l_k(d) ≤ d then
14.             return π_k                      /* path found */
15. return FALSE                                /* no feasible path */

Initialize-Single-Source(G, s, E, r_vec)
1.  for k ← 1 to E do                           /* iteration over different values of link residual bandwidth */
2.      n_k ← ⌊(r_vec[k] · j − b)/L_max⌋         /* bound on hop count determined by j */
3.      for v ∈ V do                            /* for all nodes in G */
4.          l_k[v] ← ∞                          /* for r = r_vec[k] */
5.          π_k[v] ← NIL                        /* initialize path */
6.  return

Relax(u, v, k, l_k, h, N_v)
1.  if N_v ≤ h and R_(u,v) ≥ r_vec[k] then      /* check hop count bound and link residual bandwidth */
2.      if l_k[v] > l_k[u] + l_k(u, v) then
3.          l_k[v] ← l_k[u] + l_k(u, v)
4.          π_k[v] ← u                          /* update the path */
5.  return
```

Figure 3: Pseudo-code for the widest-shortest path algorithm

load can be either evenly or unevenly distributed. For evenly distributed loads, a new session selects with equal probability any pair of switches as its source and destination. Even though networks are typically designed to match the expected traffic conditions, the network load can often be unevenly distributed in the sense that the traffic load does not precisely match the expected load, resulting in higher loads in some parts of the network than in others. We simulate such scenarios with unevenly distributed load by having a percentage of the sessions selected a source and destination from a preselected subset of the nodes, while the rest of the sessions can still pick any node as their source and destination.

4.2 Guaranteed Sessions

Sessions arrive according to a Poisson distribution. The arrival rate is determined by the traffic load for all guaranteed sessions, the mean requested bandwidth, and the mean call holding time. The parameters for the leaky bucket $\langle \sigma, b \rangle$ specified by the traffic source are selected as follows. The token rate σ is uniformly distributed between $1 \sim 5$ megabits/second and the bucket size b is uniformly distributed over one of the two intervals: [4KB, 8KB] and [16KB, 20KB]. These distributions model a combination of low and high quality video.

Most studies assume an exponential call holding time distribution but recent studies [3] show that the call holding time distribution for conversations, facsimile, and voice mail connections has a large portion of very short calls and lognormal long-tail distributions. We follow the model suggested in [3]: most of our simulation use a holding time distribution that is a mixture of two normal distributions (F_1 and F_2) on a logarithmic time scale with the mixing probability α

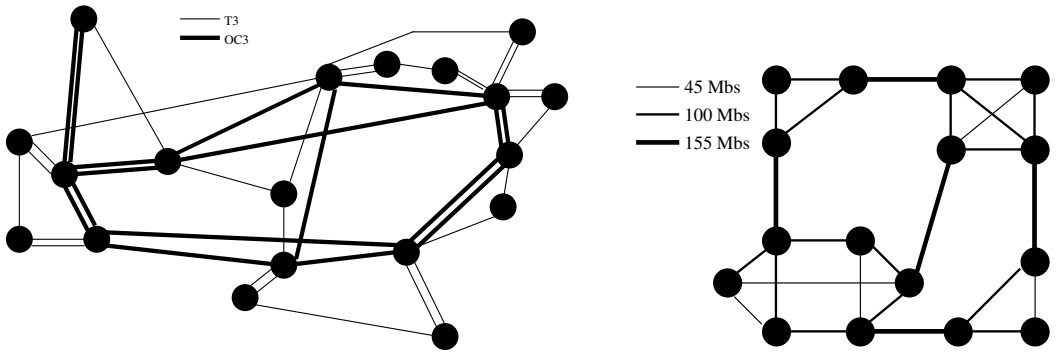
$$F(x) = \alpha \cdot F_1(x) + (1 - \alpha) \cdot F_2(x)$$

Finally, the traffic source of a guaranteed session also needs to specify its delay bound. We assume that the end-to-end delay of a session is either uniformly distributed in the interval [80ms, 120ms] or in [200ms, 240ms]. Tighter end-to-end delay bounds require more bandwidth to be reserved. This allows us to explore the impact of delay bound distribution on the performance of routing algorithms.

The performance metric for guaranteed traffic is the *call blocking rate*, the percentage of sessions being rejected by the network over the total number of arrival sessions:

$$\text{Call blocking rate} = \frac{\text{number of rejected guaranteed sessions}}{\text{number of arrival guaranteed sessions}}.$$

A guaranteed session can be rejected either because no path with sufficient resources can be found by the routing algorithm or because the resource availability on the selected path has changed since the time when the routing decision was made. We do not implement crankback.



(a) MCI topology (b) Switched cluster topology

Figure 4: Topologies used in the simulations

4.3 Best-effort Sessions

We expect that in many networks, best-effort traffic will continue to be the most dominant traffic class. In that case, the call blocking rate can be a misleading performance metric since it does not capture how routing algorithms for guaranteed traffic can influence the performance of best-effort traffic. For this reason we also examine the throughput of best-effort traffic in a network that carries both guaranteed and best-effort traffic.

Following our earlier work in [23], a best-effort session is specified by the number of bytes to be sent. All best-effort sessions share unreserved link capacity according to max-min fair share policy. This sharing policy has been adopted by the ATM ABR service as one of its design goals for congestion control mechanisms and has also been addressed for IP networks [14, 19].

The performance metric for the best-effort sessions is the average throughput of the best-effort sessions, weighted by the size of the data transfer [17]:

$$\text{Average throughput} = \frac{\text{bytes sent by best}-\text{effort sessions}}{\text{time used by best}-\text{effort sessions}}.$$

5 Performance of Guaranteed Sessions

In this section, we examine the performance of the routing algorithms described in Section 3, assuming that the network carries only guaranteed traffic. We consider both scenarios in which the network load is evenly and unevenly distributed.

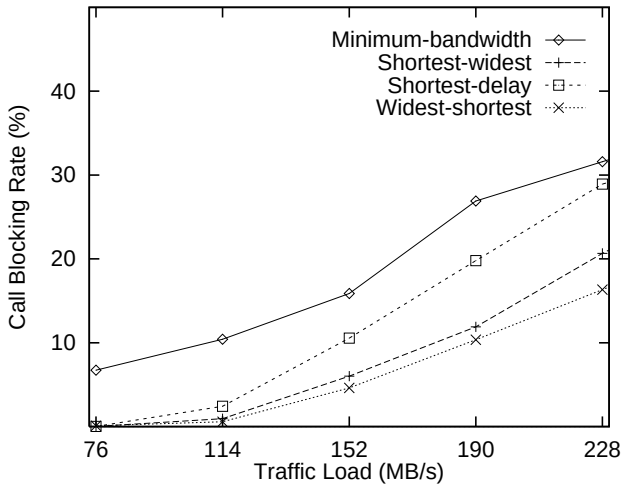
5.1 Evenly Distributed Load

Figure 5 shows the call blocking rate as a function of the network load for the MCI topology for four different combinations of traffic burstiness and delay bounds.

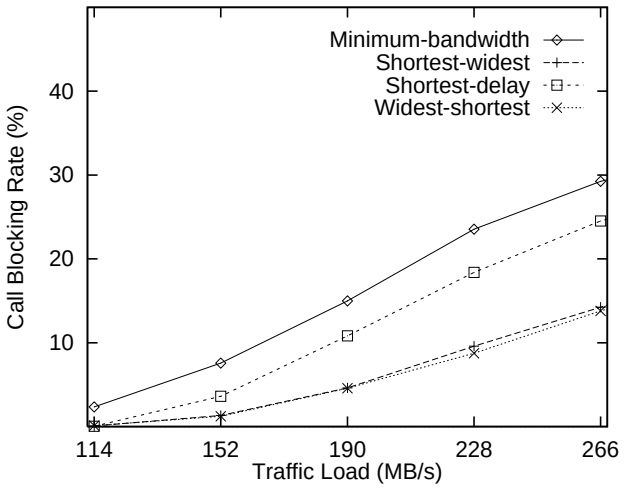
We see that the performance difference between the different routing algorithms can be large: up to a factor of two. We also see (comparing the top two graphs with the bottom two graphs) that the call blocking rate is higher when the delay bound is tighter and when the traffic is more bursty. This is to be expected: more bandwidth has to be reserved to limit the queueing delay. When the load is light, all algorithms except for the minimum-bandwidth path algorithm converge quickly and result in a zero call blocking rate.

Overall, the widest-shortest path algorithm performs better than the other three algorithms and the minimum-bandwidth path algorithm performs the worst. The reason is that the widest-shortest path algorithm selects paths with as few hops as possible to conserve resources. At the same time, it does load balancing by selecting the widest path among all paths with the same hop-count. On the other hand, the minimum-bandwidth path algorithm, which also tries to conserve resources, does not take into account the current load of the path being selected, and may pick a loaded path, thus blocking future arrivals. It is interesting to see that the shortest-widest path performs reasonably well and has a performance similar to the widest-shortest path when the delay bounds are between 200ms and 240ms. This suggests that the load is an important factor when selecting a path. The performance for the shortest-delay path is interesting. When the load is light, its performance converges to that of widest-shortest path. When the load is heavy, its performance moves close to that of minimum-bandwidth path. This can be explained using Equation (1). When the load is heavy, the r is small, which means that n in the equation weights much less than r in determining the final end-to-end delay. This results in selecting a path with higher r but with more hops.

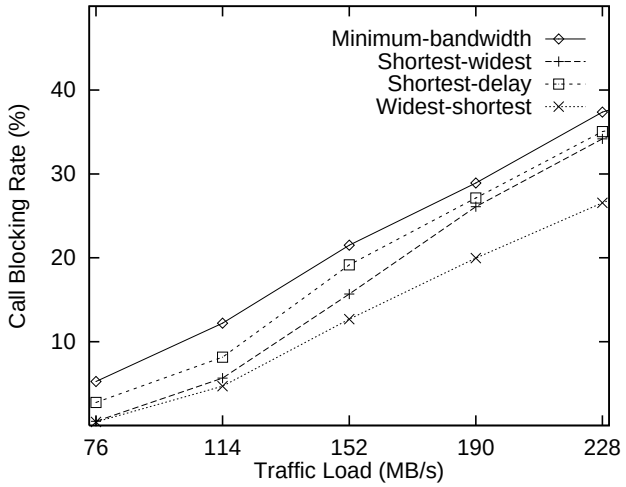
Figure 6 shows the call blocking rate for the switched cluster topology. We observe similar behavior for the different routing algorithms as for the MCI topology. The main difference is that the shortest-widest path does not perform as well as for the MCI topology: it often is worse than the minimum bandwidth path. Thus, the shortest-widest path algorithm is more sensitive to topology changes than other algorithms, which was also observed for best-effort traffic in [23]. The reason is that, for the MCI topology, a shortest-widest path is often a shortest path that consists of only OC3 links in the topology. For the switched cluster topology, a shortest-widest



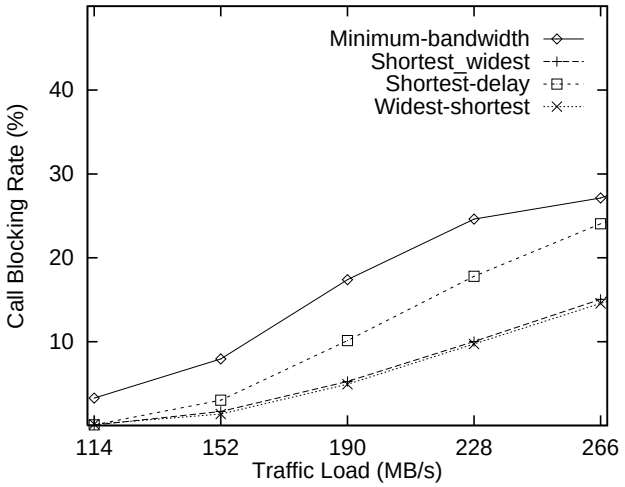
(a) b = [4KB, 8KB] and **d** = [80ms, 120ms]



(b) b = [4KB, 8KB] and **d** = [200ms, 240ms]

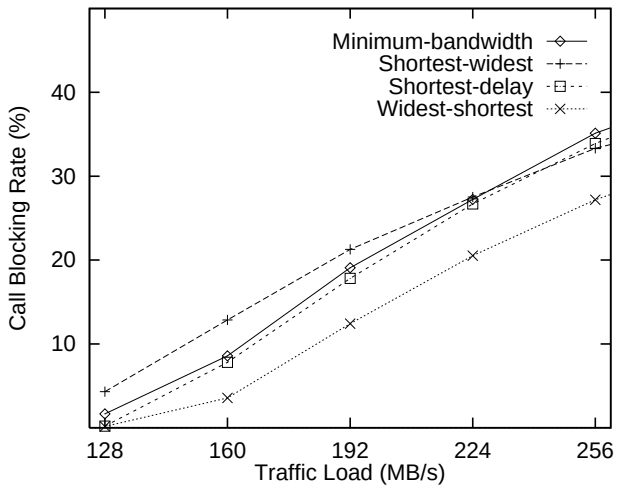


(c) b = [16KB, 20KB] and **d** =[80ms, 120ms]

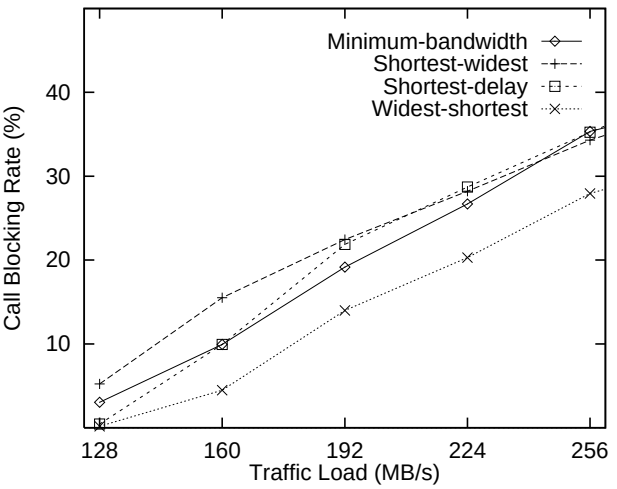


(d) b = [16KB, 20KB] and **d** =[200ms, 240ms]

Figure 5: Call blocking rate as a function of network load: MCI topology, 100% guaranteed sessions, and evenly distributed load



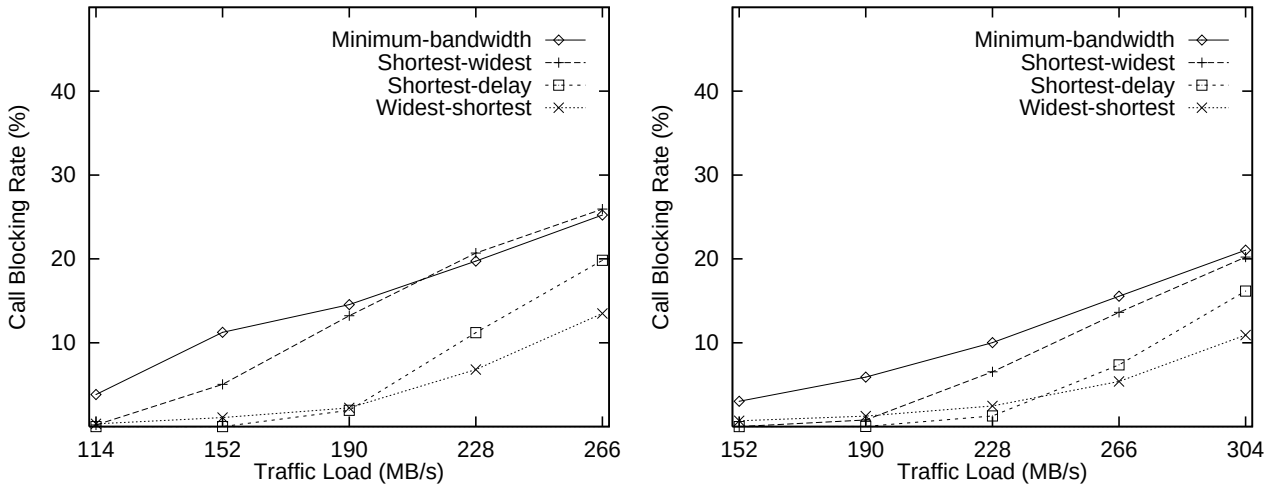
(a) b = [4KB, 8KB] and **d** = [200ms, 240ms]



(b) b = [16KB, 20KB] and **d** =[200ms, 240ms]

Figure 6: Call blocking rate as a function of network load: Switched cluster, 100% guaranteed sessions, and evenly distributed load

path is more dynamic, depending on the load of links, which often contains more hops than a shortest path.



(a) b = [4KB, 8KB] and d = [80ms, 120ms]

(b) b = [16KB, 20KB] and **d** =[200ms, 240ms]

Figure 7: Call blocking rate as a function of network load: MCI topology, 100% guaranteed sessions, and unevenly distributed load

5.2 Unevenly Distributed Load

In Figure 7, the traffic load is unevenly distributed: 50% of the traffic is between the West coast and East coast and the other 50% is evenly distributed. We again see a large performance difference between the different routing algorithms. The widest-shortest path continues to perform well, although in when the load is very light, the shortest-delay path sometimes outperform the widest-shortest path. This suggests that using a path with the minimum hop-count may make it hard to avoid some congested links. It also indicates the importance of selecting an unloaded path rather than a minimum hop path when the load is relative light. The minimum-bandwidth path still performs badly as was the case for even traffic loads. The shortest-widest path keeps performing inconsistently.

In summary, for evenly distributed loads, the widest-shortest path algorithm outperforms the other three algorithms because it tries to minimize resource use. For unevenly distributed loads, balancing load becomes also important. The shortest-delay path algorithm outperforms the other algorithms when the load is light. However, when the load is heavy, conserving resources becomes more important, and the widest-shortest path again outperforms the others. The minimum-bandwidth path algorithm performs poorly because it may select heavily loaded paths, while the performance of the shortest-widest path algorithm is very sensitive to both topology and load distribution.

The result on the widest-shortest path is different from what we observed for best-effort traffic [23], where the shortest-distance path algorithm has a clear performance edge over the other algorithms. This difference is caused by the different sharing policies of the two traffic classes. Heavily loaded links become automatically ineligible for guaranteed sessions, causing any dynamic algorithm to route around them. But these links remain eligible for best-effort traffic so the algorithm has to explicitly avoid them.

6 Performance Impact on Best-effort Traffic

In a network with both guaranteed traffic and best-effort traffic, what path is used for guaranteed traffic can have a significant impact on the performance of best-effort traffic. Thus, it is important to understand this performance impact on best effort traffic before adopting a routing algorithm for guaranteed traffic. For example, when the guaranteed traffic load is relatively low compared to the network capacity, the call blocking rate for guaranteed traffic is likely to be zero, regardless of what routing algorithm is used. In such cases, the throughput for best-effort traffic is the main performance metric distinguishing the routing algorithms for guaranteed traffic.

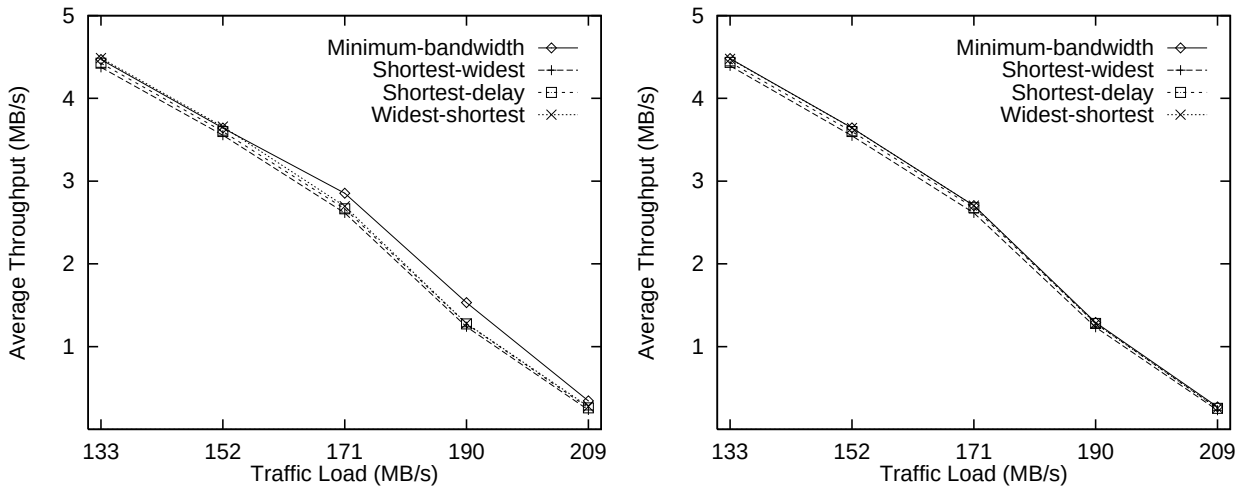
In this section, we examine the throughput of best-effort sessions in networks supporting both guaranteed traffic and best-effort traffic, for different path selection algorithms for guaranteed traffic. We assume that the best-effort sessions share the unreserved link capacity according to the max-min fair sharing policy defined in [16]. The algorithm used for best-effort sessions is the shortest distance path with the distance for a path defined as

$$\text{dist}(P) = \sum_{i=1}^{k} \frac{1}{r_i}$$

where r_i is the max-min fair share rate a new best-effort session will obtain [23].

6.1 Evenly Distributed Load

We first consider the scenarios that the traffic load is evenly distributed. The total traffic load is split between guaranteed sessions and best-effort sessions, with 40% of the traffic for guaranteed sessions. Figure 8 shows the average throughput of the best-effort sessions as a function of the total network load. We assume that up to 90% of the capacity of each link can be



(a) b = [4KB, 8KB] and **d** = [80ms, 120ms]

(b) b = [16KB, 20KB] and **d** =[200ms, 240ms]

Figure 8: Average throughput as a function of network load: MCI topology, 40% guaranteed sessions, evenly distributed load, and 90% reservation rate

reserved for guaranteed sessions. Overall, all four routing algorithms have a similar performance impact on best-effort sessions, although the minimum-bandwidth path performs slightly better than the others. However, this slight performance advantage is achieved at the cost of more guaranteed sessions being blocked. Table 1 shows the call blocking rate for guaranteed sessions. A much higher call blocking rate is observed when the minimum-bandwidth path is used, which is not surprising given the results in the previous section.

Figure 9 shows the throughput for best-effort traffic for the switched cluster topology. For all traffic loads in the figure, no guaranteed sessions are blocked. What is surprising is that the shortest-widest path performs much worse than the three other algorithms. This is suggests that, for a more symmetric network topology, applying the shortest-widest path algorithms for guaranteed sessions can cause much higher resource consumption, therefore leaving fewer resources for best-effort sessions. This again demonstrates that the performance of the shortest-widest path is inconsistent.

6.2 Unevenly Distributed Load

We next look at unevenly distributed loads. Figure 10 shows the impact of different routing algorithms on the performance of the best-effort sessions for the MCI topology, and Table 2 shows the call blocking rate for the guaranteed sessions. We see again that the shortest-widest path algorithm takes away more resource from best-effort sessions, especially when the load is heavy. The minimum-bandwidth path blocks more guaranteed sessions although its performance impact on best-effort sessions is close to that of the shortest-delay path and the widest-shortest path algorithms. Non-zero call blocking rates are also observed for the widest-shortest path algorithm because of its limited ability to balance the network load. The shortest-delay path has overall the best performance: no guaranteed sessions are blocked and best-effort sessions achieve a high average throughput.

In summary, the shortest-widest path algorithm tends to select resource-intensive paths for guaranteed sessions, which reduces the throughput of the best-effort traffic. The other three routing algorithms result in better and similar performance for the best-effort traffic.

7 Approximation Algorithms

In this section, we present approximation algorithms for the routing algorithms discussed in previous sections and evaluate their performance both in terms of running time and in terms of network throughput.

7.1 Algorithms

In the previous sections, we evaluated the performance of four routing algorithms both in terms of the call blocking rate and in terms of the average throughput of the best-effort sessions. These algorithms have a running time $O(m \cdot e \cdot E)$, where m is the number of nodes, e the number of different values of link residual bandwidth, and E the number links in the network. In the worst case, e is equal to E, and the running time is $O(m \cdot E^2)$. We observe in our simulations that this worst case is fairly common, since the residual bandwidth of two links will usually be different, although the difference may be small. The algorithms have to iterate over all these very similar but different bandwidth values. We recall that the original motivation for iterating the Bellman-Ford algorithm over all possible values of the link residual bandwidth was to ensure that no path with the shortest delay is missed, regardless of the amount of bandwidth that has to be reserved.

A first optimization is to use a small set of residual bandwidths, and to approximate the residual bandwidth of each link

Traffic load (MB/s)	133	152	171	190	209
Minimum-bandwidth	0.06	0.13	0.08	1.44	3.68
Shortest-widest	0.00	0.00	0.00	0.01	0.07
Shortest-delay	0.00	0.00	0.00	0.01	0.22
Widest-shortest	0.02	0.03	0.03	0.04	0.13

(a) b = [4KB, 8KB] and **d** =[80ms, 120ms]

Traffic load (MB/s)	133	152	171	190	209
Minimum-bandwidth	0.00	0.00	0.01	0.08	0.78
Shortest-widest	0.00	0.00	0.00	0.00	0.01
Shortest-delay	0.00	0.00	0.00	0.00	0.01
Widest-shortest	0.00	0.00	0.00	0.00	0.04

(b) b = [16KB, 20KB] and **d** =[200ms, 240ms]

Table 1: Call blocking rate in percentage: MCI topology, 40% guaranteed sessions, evenly distributed load, and 90% reservation rate

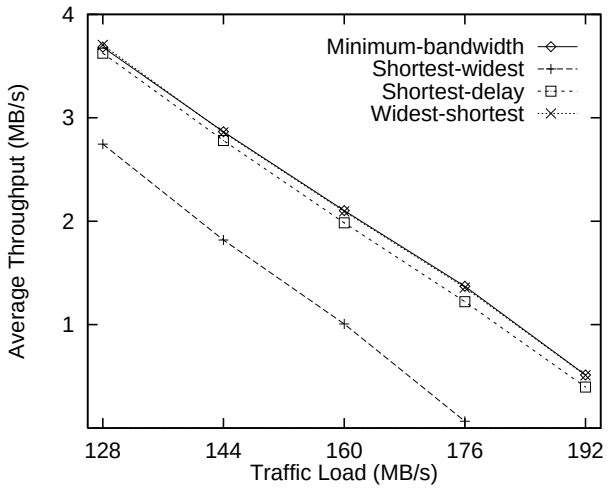


Figure 9: Average throughput as a function of network load: Switched cluster, 60% best-effort, 90% link-capacity reservation, evenly distributed load, b = [16KB, 20KB], and d =[200ms, 240ms]

by rounding down to one of these values. We can then iterate the Bellman-Ford algorithm over this smaller set of values, which enables us to eliminate iterations over close but different residual bandwidth values. A second optimization is to limit the amount of bandwidth that a flow can reserve. In practice, one would expect that for a given token rate, reserving bandwidth higher than a certain threshold will never result in acceptable network performance. For example, if the token rate of a flow is 5 Mb/s, the network should not reserve more than 25 Mb/s of the link bandwidth in order to achieve a delay bound.

Our approximation algorithms select a set of bandwidth values $V = \{v_1, \ldots, v_n\}$ such that $v_1 > \ldots > v_n$. The value v_1 is the maximal bandwidth that can be reserved. For every link, we round its residual capacity v to the largest v_i such that $v_i \leq v$. The Bellman-Ford algorithm is iterated only over the values v_i. For each iteration, it assumes that the value v_i is the reservable bandwidth.

The selection of the set V depends on the link capacity and the distribution of bandwidth requests. In order to select as few values as possible without eliminating many potential feasible paths from consideration, a heuristic is to reduce the distance between two consecutive values as the values decrease. This allows us to mimic logarithmic distance between two consecutive values. In our simulation, we select the following 14 values:

$$V = \{24, 20, 16, 14, 12, 10, 8, 7, 6, 5, 4, 3, 2, 1\}$$

In this example, 24 is the maximum bandwidth considered. Once the values of V are selected, the worst case running time of the iterative Bellman-Ford algorithm is $O(|V| \cdot m \cdot E)$, where $|V|$ is a constant. In other words, the approximation algorithms will have running times that differ from the Bellman-Ford algorithm times by up to a constant factor.

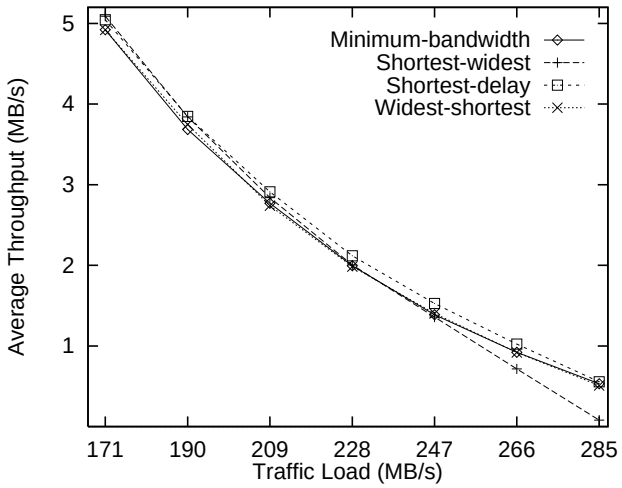


Figure 10: Average throughput as a function of network load: MCI topology, 40% guaranteed sessions, 90% link-capacity reservation, unevenly distributed load, b = [16KB, 20KB], and d =[200ms, 240ms]

Traffic load (MB/s)	209	228	247	266	285
Minimum-bandwidth	0.00	0.00	0.07	0.23	1.17
Shortest-widest	0.00	0.00	0.00	0.00	0.00
Shortest-delay	0.00	0.00	0.00	0.00	0.00
Widest-shortest	0.00	0.00	0.06	0.10	0.19

b = [16KB, 20KB] and **d** =[200ms, 240ms]

Table 2: Call blocking rate: MCI topology, 40% guaranteed sessions, unevenly distributed load, and 90% reservation rate

7.2 Performance on Network Throughput

The simulation results in Sections 5 and 6 show that the shortest-widest path and the minimum-bandwidth path perform inconsistently for different topologies because they put either too much weight on balancing the network load or on bandwidth consumption. Thus we focus our evaluation on the approximation algorithms for the shortest-delay and the widest-shortest path.

Figures 11 and 12 show for the MCI topology the performance difference between the widest-shortest path and the shortest-delay path algorithms, and their approximations, for even and uneven traffic loads respectively. We see that the call blocking rates for the widest-shortest path and its approximation are very similar, while the approximation algorithm for the shortest-delay path performs better than the original "correct" algorithm. This result is somewhat unexpected. Further analysis shows that the improvement is not caused by the limit on the reserved bandwidth, since the exact algorithms rarely allocate high bandwidth anyway. Instead the performance improvement is the result of the use of bandwidth intervals. Recall that the end-to-end delay (see Equation 1) is a function of the hop count and reservable bandwidth. By using bandwidth intervals, we reduce the reservable bandwidth to its closest discrete interval value and equalize the link bandwidths with small differences, which leads the shortest-delay path algorithm to select paths with fewer hops, and therefore, to place more emphasis on conserving network resources. When the load is heavy, the reservable bandwidth is small, and reducing the bandwidth is likely to have a bigger impact. However, the approximation scheme has little impact on the widest-shortest path algorithm. The reason is that it is already very biased towards minimizing resource utilization.

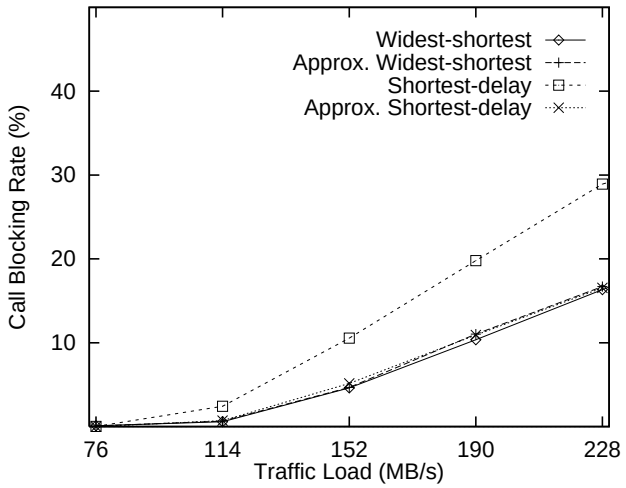
Figure 13 shows similar results for the switched cluster topology. Again, the heuristic for the shortest-delay path algorithm performs better than the original algorithm, although it does not perform as well as the widest-shortest path algorithm for this scenario. The reason is that in this fairly symmetric topology conserving resources is more important than balancing the load, since the load is already fairly evenly distributed.

7.3 Running Time

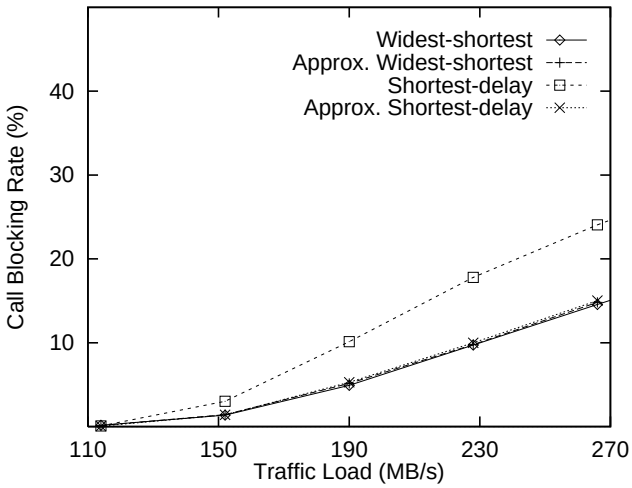
An important performance issue for routing is the running time of the path selection algorithm. Our performance metric is the average algorithm running time in our simulation:

$$\text{Average running time} = \frac{\text{time used by routing algorithm}}{\text{number of flows}}.$$

We compare the average time of the approximation algorithms and the original iterative Bellman-Ford algorithms. The measurements were carried out on a DEC Alpha Station 255. Figure 14 shows the time (in milliseconds) needed for both the

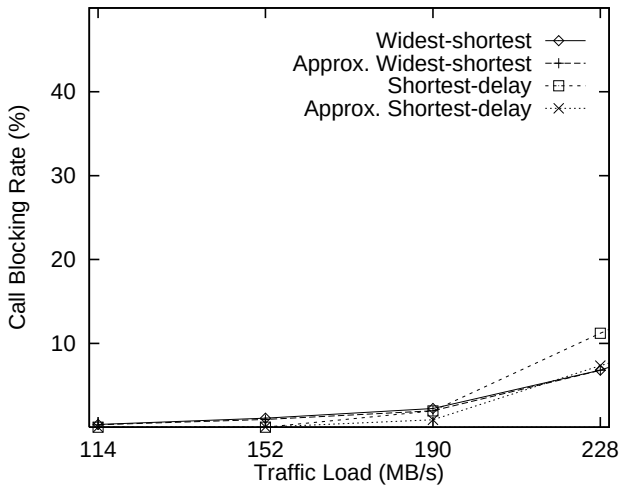


(a) b = [4KB, 8KB] & d = [80ms, 120ms]

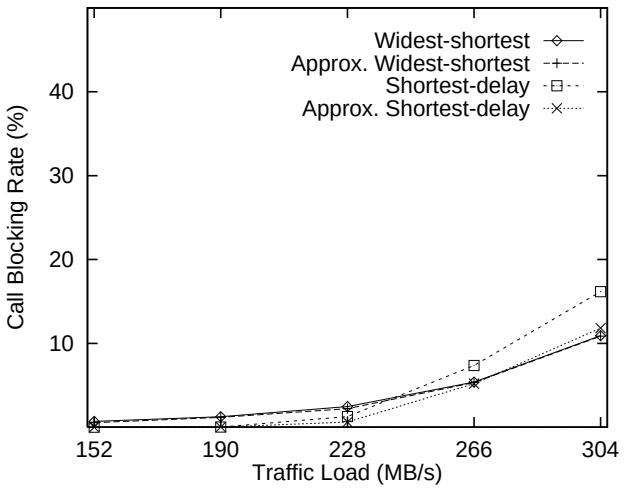


(b) b = [16KB, 20KB] & d = [200ms, 240ms]

Figure 11: Call blocking rate as a function of network load: MCI topology, 100% guaranteed sessions, and evenly distributed load



(a) b = [4KB, 8KB] & d = [80ms, 120ms]



(b) b = [16KB, 20KB] & d = [200ms, 240ms]

Figure 12: Call blocking rate as a function of network load: MCI topology, 100% guaranteed sessions, and unevenly distributed load

original and approximation algorithms for the widest-shortest path and the shortest-delay path. We see that the approximation algorithms outperform the original algorithms by as much as a factor of eight. Moreover, we see that the widest-shortest path algorithm and its approximation run 18% faster than the shortest-delay path and its approximation, respectively. The significant improvement on the running time of our approximation algorithms comes from eliminating iterations over different but similar values of the link residual bandwidth.

8 Related Work

Many studies in the literature have addressed different aspects of QoS routing. A good introduction to QoS routing and existing routing techniques can be found in [18, 31]. Trunk Reservation and Dynamic Alternative Routing [8], and Least Load Routing [13] are some of the algorithms that have been studied in telecommunications networks. In [5], a QoS based routing framework is defined and many challenging questions are raised.

Several studies have investigated routing algorithmic problems associated with QoS routing. In [7, 34], it is shown that the QoS routing problem of finding a path satisfying both delay and delay-jitter constraints is NP-complete. Jaffe and Salama studied [15, 29] heuristics to tackle the NP-problem of routing with two additive constraints. Przygienda suggested in [26] the use of cost vectors to solve the NP-complete by associating a priority to each constraint. Rampal [27] evaluated the performance of several path selection algorithms under the assumption of three different scheduling algorithms. The IBF algorithm that is used in this paper was first described in [22]. Zhao and Tripathi in [38] proposed a similar algorithm, but is-

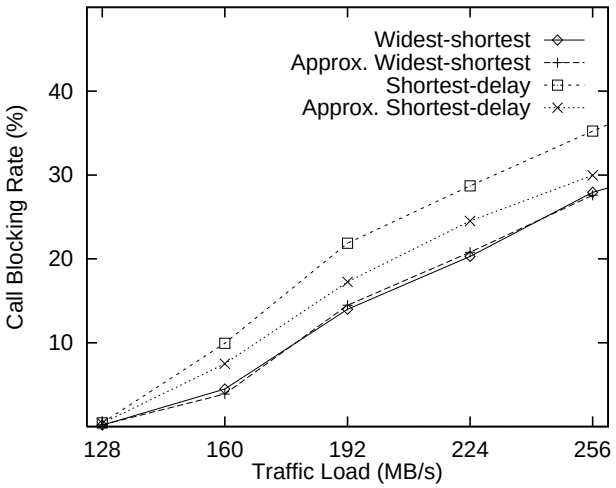


Figure 13: Call blocking rate as a function of network load: Switched cluster, 100% video traffic, evenly distributed load, b = [16KB, 20KB], and d = [200ms, 240ms]

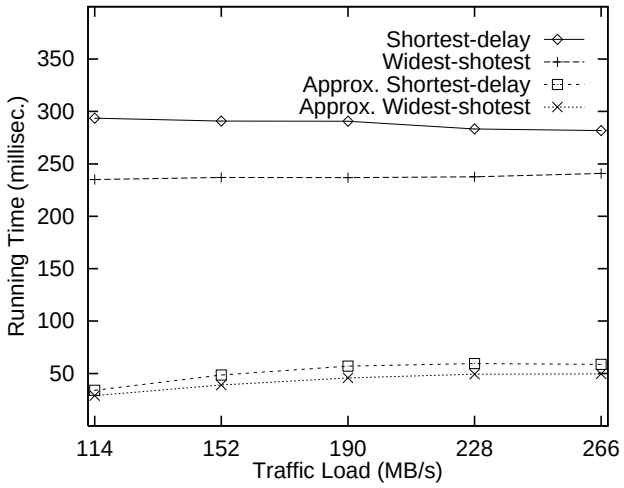


Figure 14: Average running time as a function of network load: MCI topology, 100% guaranteed sessions, 90% link-capacity reservation, unevenly distributed load, b = [16KB, 20KB], and d =[200ms, 240ms]

sues related to selecting efficient paths and resource utilization efficiency are not addressed. Several people have presented algorithms that address a subset of the problems associated with routing guaranteed traffic. In [28], routing algorithms for achieving the shortest delay when moving a burst of bytes are proposed. In [25], QoS routing algorithms are studied for finding the shortest-delay path when the network employs the weighted fair queueing service discipline. In [11], QoS routing algorithms for a network with inaccurate link-state information are explored. Guerin, Orda, and Williams outlined in [12] how OSPF can be extended to QoS routing, using the widest-shortest feasible paths, but no performance evaluation is provided. QoS routing algorithms for traffic requiring bandwidth guarantees are addressed in [21] and related work can be found there.

In contrast to the previous studies, we present not only a complete set of QoS routing algorithms for traffic with delay, delay jitter, and buffer space constraints with a variety of optimality criteria, but also approximation algorithms that run significantly faster than the original algorithms without sacrificing resource utilization efficiency. We also present a detailed performance evaluation of four different algorithms, considering both call blocking rate and impact on best-effort traffic as performance metrics.

9 Conclusion

In order to support applications with stringent QoS requirements such as interactive continuous media applications, both the ATM and IP community have defined "guaranteed" service classes that provide per-flow guarantees for delay, delay-jitter, and loss rate. QoS routing is one of important resource management components that support QoS in integrated services networks.

In this paper we address two important issues related to achieving resource efficiency while selecting feasible paths and the effectiveness and practicality of routing algorithms. To achieve low blocking rates, we identified four optimization criteria that place a different emphasis on minimizing resource utilization and balancing the load in the network. Based on the IBF algorithm, we developed path selection algorithms that select paths with these different criteria. The performance of these routing algorithms is evaluated through an extensive simulation study, using realistic topologies, traffic models, and periodic routing information distribution. Unlike telecommunication networks, multiple classes of service will be supported in data networks. Our performance evaluation considers not only the blocking rate for guaranteed traffic but also the influence on the throughput of best-effort traffic. Our results show that the performance gap for these different algorithms is large and both conserving resources and balancing the network load are necessary to achieve high network throughput. The widest-shortest path algorithm performs well for heavy load because it conserves resources, but the shortest-delay path has a performance edge for light load because it balances the load better. Surprisingly, the minimum-bandwidth path algorithm has the worst performance because it is not very sensitive to load, and can easily saturate some links while other parts of the network are underutilized.

Even though the IBF algorithms are polynomial, their computational complexity is significantly higher than that of the Bellman-Ford algorithm. We develop approximation algorithms that runs almost ten times faster than the exact iterative Bellman-Ford algorithm for an MCI backbone topology. The simulation results show that there is no loss in routing accuracy and network throughput. Overall, the approximation algorithm for the shortest-delay path performs consistently well.

Our results demonstrate that QoS routing is both desirable and feasible. It is desirable because using carefully selected routes can significantly reduce the blocking rate and improve throughput. QoS routing is also feasible because routes that meet QoS constraints can be selected using algorithms with reasonable cost, i.e. within a constant factor of the execution times of the Bellman-Ford algorithm. The routing information needed is link residual bandwidth and link propagation delay.

Acknowledgements

This research was sponsored by the Defense Advanced Research Projects Agency monitored by Naval Command, Control and Ocean Surveillance Center (NCCOSC) under contract number N66001-96-C-8528.

References

[1] ATM Forum Traffic Management Specification Version 4.0, October 1995. ATM Forum/95-0013R8.

[2] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case Fair Weighted Fair Queueing. In *Proceedings of IEEE INFOCOM'96*, May 1996.

[3] V.A. Bolotin. Modeling Call Holding Time Distributions for CCS Network Design and Performance Analysis. *IEEE JSAC*, 12(3):433–438, April 1994.

[4] David Clark, Scott Shenker, and Lixia Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. *ACM SIGCOMM 92*, August 1992.

[5] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for QoS-based Routing in the Internet. *IETF Internet Draft <draft-ietf-qosr-framework-01.txt>*, July 1997.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. *ACM SIGCOMM 89*, 19(4):2–12, August 19-22, 1989.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979. (page 214).

[8] R.J. Gibbens, F.P. Kelley, and P.B. Key. Dynamic Alternative Routing — Modelling and Behaviour. In *Proceedings of the 12th International Teletraffic Congress*, June 1988.

[9] S. Golestani. A Self-Clocked Fair Queueing Scheme for Broadband Applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, Canada, June 1994.

[10] P. Goyal and H.M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. Technical Report Technical Report TR95-30, Department of Computer Science, UT Austin, Texas, 1995.

[11] R. Guerin and A. Orda. QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms. In *Proc. IEEE INFOCOM'97*, 1997.

[12] R. Guerin, A. Orda, and D. Williams. QoS Routing Mechanisms and OSPF Extensions. *IETF Internet Draft <draft-guerin-qos-routing-ospf-00.txt>*, November 1996.

[13] S. Gupta, K.W. Ross, and M. El Zarki. On Routing in ATM Networks. In M. Steenstrup, editor, *Routing in Communications Networks*, pages 49–74. Prentice Hall, 1995.

[14] E. Hahne. Round-Robin Scheduling for MaxMin Fairness in Data Networks. *IEEE Journal on Selected Areas in Communications*, 9(7), September 1991.

[15] J. Jaffe. Algorithms for Finding Paths with Multiple Constraints. *Networks*, 14(1):95–116, Spring 1984.

[16] J.M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, COM-29(7):954–962, July 1981. Correspondence.

[17] R. Jain. *The Art of Computer Performance Analysis*. Wiley, 1991.

[18] W.C. Lee, M.G. Hluchyj, and P.A. Humblet. Routing Subject to Quality of Service Constraints in Integrated Communication Networks. *IEEE Network*, 9(4):14–16, July/August, 1995.

[19] D. Lin and R. Morris. Dynamics of Random Early Detection. In *ACM SIGCOMM '97*, pages 115–126, Cannes, France, Sepetember 1997.

[20] Q. Ma. *Quality of Service Routing in Integrated Services Networks*. PhD thesis, Department of Computer Science, Carnegie Mellon University, 1998.

[21] Q. Ma and P. Steenkiste. On Path Selection for Traffic with Bandwidth Guarantees. In *IEEE International Conference on Network Protocols*, Atlanta, Georgia, October 1997.

[22] Q. Ma and P. Steenkiste. Quality-of-Service Routing for Traffic with Performance Guarantees. In *IFIP Fifth International Workshop on Quality of Service*, pages 115–126, NY, NY, May 1997.

[23] Q. Ma, P. Steenkiste, and H. Zhang. Routing High-Bandwidth Traffic in Max-Min Fair Share Networks. In *ACM SIGCOMM '96*, pages 206–217, Stanford, CA, August 1996.

[24] A. Parekh and R. Gallager. A Generalized Processor Sharing Approach to Flow Control-the Single Node Case. *IEEE/ACM Transactions on Networking*, 3(1):344–357, June, 1993.

[25] C. Pornavalai, N. Shiratori, and G. Chakraborty. QoS Based Routing Algorithm in Integrated Services Packet Networks. In *IEEE International Conference on Network Protocols*, Atlanta, Georgia, October 1997.

[26] A. B. Przygienda. *Link State Routing with QoS in ATM LANs*. PhD thesis, Dipl. Inf. Ing. ETH, 1995.

[27] S. Rampal. *Routing and End-toend Quality of Service in Multimedia Networks*. PhD thesis, North Carolina State University, August, 1995.

[28] J.B. Rosen, S.Z. Sun, and G.L. Xue. Algorithms for the Quickest Path Problem. *Computers and Operations Research*, 18(6):579–584, 1991.

[29] H.F. Salama, D.S. Reeves, and Y. Viniotis. A Distributed Algorithm for Delay-Constrained Routing. In *Proceedings of IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[30] S. Shenker, C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service. *IETF Internet Draft <draft-ietf-intserv-guaranteed-svc-07.txt>*, February 1997.

[31] M. Steenstrup. *Routing In Communications Networks*. Prentice Hall, Englewood Cliffs, NJ 07362, 1995.

[32] D. Stiliadis and A. Varma. Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks. In *ACM SIGMETRICS 96*, Philadelphia, PA, May 1996.

[33] D. Stiliadis and A. Varma. A General Methodology for Designing Efficient Traffic Scheduling and Shaping Algorithms. In *Proceedings of IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[34] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE JSAC*, 14(7):1288–1234, September 1996.

[35] H. Zhang. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10), October 1995.

[36] L. Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. *SIGCOMM 90*, pages 19–29, 1990.

[37] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource Reservation Protocol. *IEEE Communications Magazine*, 31(9):8–18, September 1993.

[38] W. Zhao and S. Tripathi. Routing Guaranteed Quality of Service Connections in Integrated Services Packet Networks. In *IEEE International Conference on Network Protocols*, Atlanta, Georgia, October 1997.

Design and Evaluation of a Platform for Mobile Packet Telephony

Sandesh Goel†, Partho Mishra†, Huzur Saran*, Cormac Sreenan†

† AT&T Labs - Research
Florham Park, NJ 07932

* Department of Computer Sc. and Eng.
Indian Institute of Technology - Delhi

1 Introduction

Advances in packet switching technology have made it feasible to support integrated transport of voice, video and data on a common packet switched network infrastructure. This trend has stimulated research in exploring how integrated transport may be supported over *wireless* networks as well, to provide mobile users access to the same set of services that are available in their desktop environment. Due to spectrum limitations, specially for wide area wireless coverage, it may be practical to initially support only a limited form of integration, for example to enable the transmission of voice and best-effort data on a shared physical channel. However, even this capability makes it possible to design interesting new services. Consider for example, a mobile user who wishes to retrieve personalized directions or traffic updates while driving down a freeway. If the wireless channel supported integrated transport, it becomes feasible to design a service interface in which the user interacts with the appropriate network server using an interactive voice driven interface to specify the information required. This information might then be presented to the user in a variety of styles - as audio, text, images etc. Another example is a repairman dispatched to work at a remote location. Integrated transport might provide him the option of consulting with another repairman at headquarters, while perusing product manuals and transmitting realtime sensor feedback describing the product's current state.

Services that result from voice-data integration may be particularly valuable for cellular/PCS telephony users who constitute the fastest growing segment of the wireless industry. A limited form of wireless integrated access is already possible with so-called *smart phones* like the Nokia communicator [26] and the AT&T PocketNet phone [25] in which the smart phone can function either as a phone or as a (IP) data terminal offering access to e-mail and personalized information on stock quotes, weather reports, etc. These devices use separate wireless access protocols when functioning as a phone or as a data terminal, prohibiting the simultaneous transport of both voice and data. Voice-data integration is underway in the Internet as well with the development of applications centered around Internet telephony, for example click-to-dial and simultaneous voice-data transmission over dial-up (PPP) links.

There are several interesting problems that need to be solved to make it possible to support integrated voice-data access over wireless links [20]. First, it is necessary to design wireless channel scheduling mechanisms that can efficiently satisfy the service quality requirements of both voice and data. Second, network protocols need to be extended to support *handoffs* as mobile terminals change their point of connectivity to the backbone network. Third, forward error correction/packet retransmission mechanisms used over the wireless link need to be be tuned to match the varied loss resilience levels of voice and data. Fourth, integrated voice-data mobile terminals need to be developed that can act as both a phone and information-retrieval device. Finally, there are several interesting issues in providing enhanced voice-data services for mobile-users, for example how to support voice-driven information retrieval and adapt services and data delivery to match the limitations of the terminal and wireless link.

In this paper, we describe an experimental testbed that we have designed to support integrated wireless voice-data access. It consists of off-the-shelf network adapters, PCs as basestations, and handheld mobile terminals. This approach of using commodity hardware allowed us to assemble the testbed relatively quickly, providing us with a platform for prototyping algorithms and applications, and examining performance in a realistic but controlled environment. It also provides a clean upgrade path as new technology becomes available. Our goals were to investigate the network mechanisms required to support integrated voice/data transport to mobile terminals, and explore the design of interesting services/applications for this network environment. The initial focus has been on the design and implementation of a Medium-Access-Control (MAC) and handoff protocols that are suitable for integrated voice/data transport. The MAC protocol provides a simple token-based mechanism to support bandwidth reservation - an essential element for supporting time-sensitive voice traffic. The handoff protocols allow a wireless terminal to move between geographical

areas while maintaining connectivity to the backbone network, and transparently rerouting data to/from its communicating peers. We present experimental results that analyze these protocols in terms of throughput and latency performance. We also measure the effect of handoff interruption on packet audio applications, and specifically its impact on playback buffer operation. We compare the trade off between tolerating a small amount of packet loss during a handoff to the additional delay incurred by buffering packets to avoid losses.

The paper is organized as follows. Section 2 describes the testbed components. Section 3 presents details of our wireless MAC protocol. Section 4 describes our algorithms for doing handoff and data rerouting. Section 5 presents our experimental results using the testbed to provide packet audio communications to a mobile handheld computer. Section 6 reviews the related work. We conclude the paper in Section 7.

2 Mobile Packet Telephony testbed

In creating the testbed, we have assumed a network architecture in which wireless links are used for last-hop access only and *basestations* act as gateways to an (integrated services) packet switched network, i.e. all communication to/from a mobile terminal is funneled through a basestation. We assume that the link layer protocol is ATM, i.e. each mobile terminal is an ATM endpoint and basestations are interconnected using ATM switches. Basestations are logically identical to ATM switches, i.e. they execute a NNI signaling protocol with their neighboring ATM switches, and switch cells (or packets) from one network interface to another. However, transmission on the wireless link uses variable size packets that may be larger than a single ATM cell for more efficient utilization of the wireless link capacity. We assume that connectivity to the Internet and to the PSTN is provided through gateways.

Figure 1 illustrates the current structure of the testbed. A mobile terminal is a PDA / subnotebook / laptop equipped with a wireless interface card. Basestations are PCs equipped with one or more wireless interface cards for communicating with mobile terminals, and with ATM network interface cards for communicating with ATM switches. We use two types of ATM switches - ATML switches that are controlled from an external PC that runs custom signaling/control software that we have designed, and FORE switches that run their own signaling/control software. A Powerhub IP router connects the testbed to external (IP) networks. The testbed also includes several *static* terminals, i.e. regular PCs with ATM network interface adapters that connect to one of the ATM switches. All of the functional entities in the testbed have been realized using off the shelf hardware components, while implementing custom algorithms in software. Moreover, we have tried to use open platforms so that we can piggy-back off the existing software base of public-domain applications, signaling protocols and device drivers.

The wireless link used for communication between a mobile terminal and a basestation is realized using Lucent's WaveLAN wireless LAN adapters. WaveLAN is a popular Wireless LAN adapter that operates in the Industrial, Scientific and Medical (ISM) frequency bands (915 MHz and 2.4 GHz) providing a shared bandwidth of up to 1.2 Mb/s in the coverage area[1] The WaveLAN network interface cards and associated operating system device drivers use a single FIFO queue per host, and implement a Ethernet-style random access MAC protocol to arbitrate access to the shared wireless channel among multiple hosts. A common FIFO queue is clearly undesirable when multiplexing traffic from applications with very different QoS requirements. Random-access MAC protocols can allow a single transmitter that seizes access to a channel to lock out other transmitters for several hundred milliseconds [5]. This is clearly unacceptable for packet telephony. To avoid these problems, we have modified the WaveLAN device driver to implement per-connection queueing and to use a basestation controlled token-passing MAC protocol that supports two distinct classes of service: fair-best-effort and guaranteed-bandwidth [13]. The details of this protocol are described in Section 3.

Basestations are PCs equipped with both wireless and ATM network interface cards. The cell/packet switching functionality is implemented in software (in the Linux kernel). The switching tables in the kernel are manipulated by a signaling module running in user space through a pseudo-device driver. The control protocol used across this interface is the General Switch Management Protocol (GSMP) [24]. GSMP was originally designed to allow external switch controllers the ability to modify the switching tables of a hardware ATM switch. It allows a controller to establish and release connections across the switch; add and delete leaves on a point-to-multipoint connection; manage switch ports; request configuration information; and request statistics. The choice of GSMP allows us to run the same signaling code on a PC and use it to control a hardware ATM switch, as we do with the ATML Virata switch in our testbed.

Each basestation handles communication to/from mobile terminals in a particular geographical area. As the mobile terminal changes its geographical location, it may be "handed-off" from one basestation to another. Since mobile terminals use ATM virtual circuits for communication (at least up-to the nearest IP router), a handoff requires rerouting of these virtual circuits. We have augmented the tradi-

[1] The use of WaveLAN as a wireless link restricts us to primarily indoor coverage.

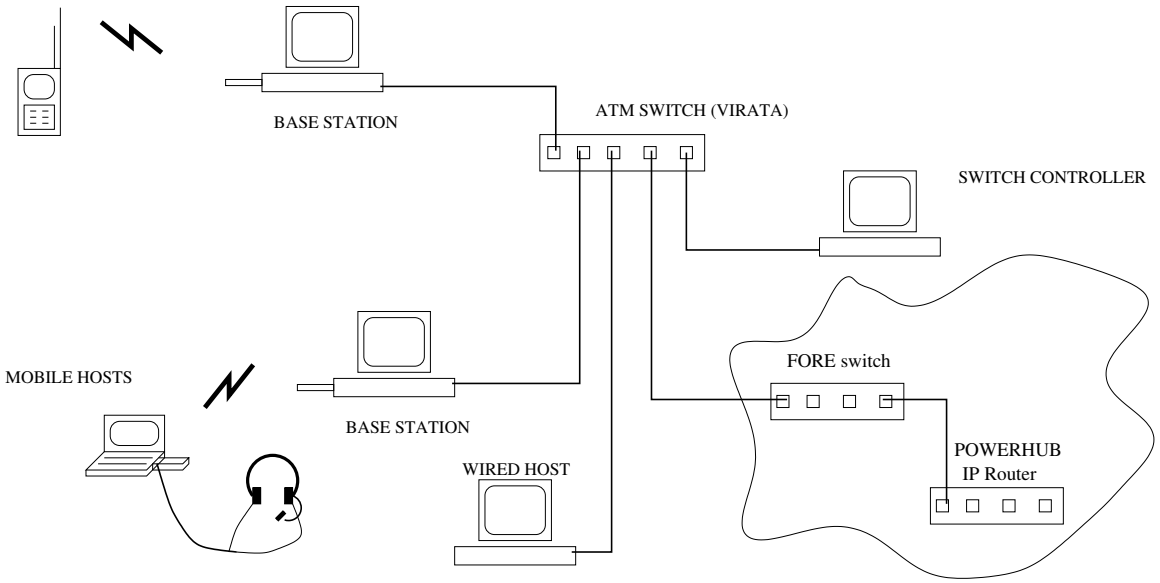


Figure 1. Testbed Configuration

tional ATM UNI/NNI signaling protocols running on mobile terminals and switches to support this rerouting capability - the mechanisms we have implemented allow a very rich set of rerouting policies to be instantiated. The details of this protocol are described in Section 4.

An integrated mobile terminal should ideally be able to function as a phone or a Walkman or a PDA by providing users the ability to access applications such as telephony, networked music play back, email, calendars, address books etc. It should be a small, lightweight device with (wireless) network connectivity. The device would need to support audio processing and a microphone/speakers for audio I/O. It might also need to support one or more user-interfaces such as a keypad, a touch sensitive display or voice-driven I/O. Smart phones such as the Nokia Communicator as well as PDA's such as the 3Com Palm Pilot provide some but not all of these capabilities. We have chosen instead to realize mobile terminals using laptops/palmtops along with the necessary I/O and networking peripherals. One of our mobile terminals is a IBM PC110 palmtop which measures 158x113x33mm and weighs only slightly more than a pound. This device has a 486SX CPU, 20 Mbyte RAM, 52 Mbytes of internal flash memory, a color LCD display (640x480 in a 4.7" screen) and 2 PCMCIA card slots.

All our terminals run the Linux operating system. Linux was chosen because of several reasons. First it allowed us to run, unmodified, applications such as the MBONE audio tool `vat`, and a locally-developed packet telephony application. Second, we already had a working implementation of our custom WaveLAN device driver for Linux. Third, source code availability for both the Linux kernel and various device drivers allows us to perform accurate timing measurements and meet the stringent delay and jitter requirements of packet telephony. Unfortunately, the PC110 has a fairly low quality half-duplex sound card that is unsuitable for packet telephony. Instead, we use a PCMCIA sound card from Communication, Automation & Control Inc for which we have developed a Linux device driver[2]. Figure 2 shows the PC110 equipped with a WaveLAN PC-card and the sound card. Clearly this is somewhat clumsier than would be desirable in a consumer product, but it satisfies our purpose of demonstrating the technology potential and quantifying system performance.

Applications running on a mobile terminal can use either a native ATM or a TCP/IP protocol stack for communicating with applications running on other machines. The native-mode ATM stack is an enhanced version of the IDLINet stack [10, 3] developed by S. Keshav and H. Saran. A client application that wishes to initiate communication with a server application, needs to first send a message to the signaling entity (via UNIX IPC). The signaling entity then initiates the connection setup process by sending out a UNI SETUP message. When the signaling entity at the remote end receives the request it informs the remote server application. If the remote application accepts the call, a virtual circuit is set up between the client and the server. At the time of connection setup, the client describes its service quality requirements to the signaling entity, and this is made available to both local and remote protocol stacks, so

[2]Thanks to Jont Allen of AT&T Research for providing us with an initial version of a device driver for this device and a lot of subsequent help in extending its capabilities.

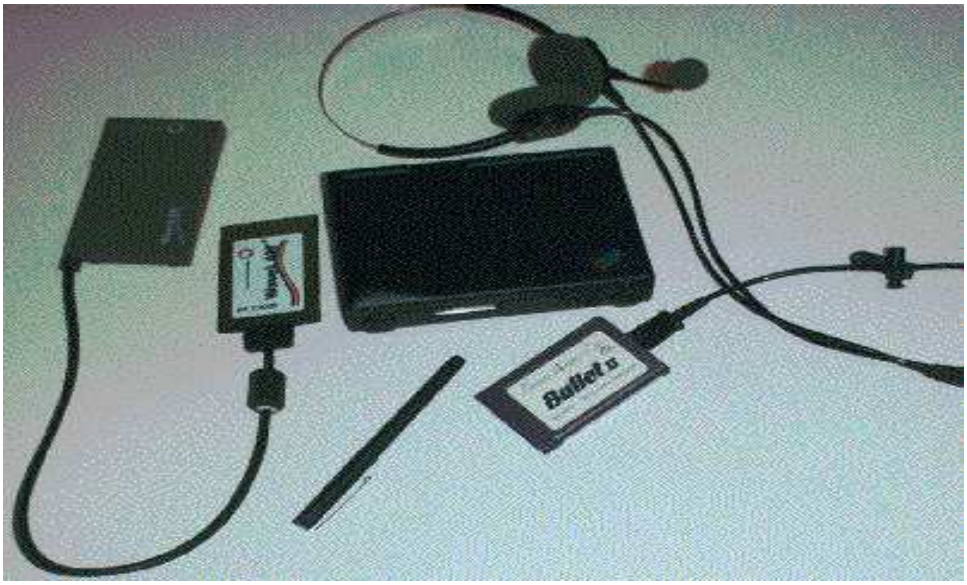


Figure 2. PC-110 with peripherals

that the appropriate control actions can be taken at the end systems as well as by intermediate network elements.

IP-based applications are supported over the ATM network segment, using a Classical IP over ATM implementation [11] in which the entire ATM cloud appears as a single IP subnet to the Powerhub router. We have begun to explore mechanisms to provide QoS-support for IP applications through a mix of pre-configuration and run-time traffic estimation. This allows us to avoid the need to modify existing applications and IP protocol stack implementations.

3 Medium Access Control

The WaveLAN physical layer is implemented using a direct sequence spread spectrum in combination with DQPSK modulation. The raw physical layer data rate is 2 Mb/s. The adapter implements a CSMA/CA medium access scheme. A transmitter that needs to send data first checks to see if the channel is busy. If it not busy, transmission can start immediately. If it is busy, then the transmitter waits until the channel becomes idle and then computes a back-off interval before trying to transmit again. Repeated failures lead to the usual exponential backoff algorithm. Due to the inherent non-determinism, this class of random access protocols are not suitable for telephony-like applications that require guaranteed bandwidth and/or tight control over delays. Many *scheduled access* protocols have been proposed for use in shared-media Local Area Networks that attempt to address some of these problems for example the "point coordination function" of IEEE 802.11. These protocols are typically designed to support multiple classes of traffic with differing service quality requirements.

We chose instead to design a simple scheduled access MAC protocol that supported only two classes: guaranteed bandwidth (voice) and best effort (data). This MAC uses a frame based scheduling policy [9] to control access to the broadcast wireless channel. A frame is a fixed interval of time and is comprised of several slots. Individual connections are provided bandwidth guarantees by allocating them a certain number of slots in each frame. These reservations are setup using the signaling protocols at virtual circuit setup. Any unreserved or unused slots in a frame are allowed to be used for best effort traffic. The MAC protocol operates within a cell, i.e. the geographical area controlled by each basestation. We assume that basestations are spatially separated to minimize interference between independent transmissions in different cells.

We have modified the WaveLAN device driver to implement this scheduled access MAC. In our implementation, a basestation computes the transmission schedule for each frame and based on this transmits a token to a mobile granting it the right to transmit for a certain number of slots. The mobile is expected to hand the token back to the base station after completing the specified transmission. A mobile terminal or a basestation cannot send a packet unless it has a token - this ensures that only one transmitter is active at any time thereby disabling the native CSMA/CA protocol implemented in the WaveLAN adapter. Tokens are piggy-backed onto data transmissions whenever possible for greater efficiency. To facilitate this, the frame computation algorithm attempts to schedule a reception slot from a mobile just after a transmission slot to that mobile. If all of

the mobile terminals had tightly synchronized clocks, it is possible to reduce the overhead of explicit token-passing by having the basestation compute slot allocations and broadcast these at the beginning of each frame. This is not feasible with our current hardware.

Ideally, the slot allocation within a frame for best effort traffic should be fair across all active connections. We use the Start Time fair queueing algorithm to implement fair sharing. This requires the basestation to have knowledge of the amount of pending (best effort) data for each connection at each mobile terminal. This is achieved by piggy-backing this information onto regular data/voice packets (in the link layer protocol) whenever a mobile terminal sends packets to the basestation. Signaling traffic is also carried on the best effort slots but is given a higher weight and so normally gets priority over other best effort traffic. If a reserved slot at a mobile terminal is going unused, it is used to transmit best effort data.

When a mobile terminal moves into the coverage area of a new basestation it needs to discover the identity of the new basestation and register with it. A basestation identifies itself by periodically sending out a beacon. In response, a new mobile sends in a registration message. The MAC supports this by allocating two best effort slots for the beacon/beacon-response pair. When more than one mobile unit enters a coverage area simultaneously and try to respond to the beacon, the collision avoidance mechanisms of the WaveLAN hardware kicks-in and ensures that only one mobile terminal transmits during that slots. Those mobile terminals that failed to send a beacon-response on their first attempt defer their transmission until they get the next beacon; thus only one mobile terminal can successfully register per-frame.

The link layer protocol supports variable sized packet transmission.[3] We leave the decision on packet sizes to individual connections but determine the slot allocation based on a normative packet size for each connection which is made available via the signaling protocol. As such our guarantees hold good only if the connection uses packets of this normative size. Although it is potentially possible to police connections to ensure these packet sizes, we have not yet implemented this option. The choice of optimal packet size for a particular connection depends on application-specific requirements. Large packet sizes lead to more efficient use of the channel bandwidth, since the ratio of the data payload to protocol headers is higher. On the other hand, for wireless telephony applications, the use of large packet sizes may increase the end-to-end delay due to the relatively large time interval spent in assembling a packet. The encapsulation protocol used by the link layer protocol is described in [13].

[3]The WaveLAN hardware allows packet sizes to vary from 64 bytes to 1536 bytes.

4 Handoff policies

As a mobile terminal roams geographically, it may move from the vicinity of one basestation to another. Supporting such mobile terminal roaming requires network protocols to track the location of a mobile terminal and reroute existing virtual circuits to/from the mobile terminal. There have been several proposals for how data rerouting might be accomplished [1, 4, 8, 14, 18, 19]. For example, rerouting may be accomplished by extending a connection from one basestation to another, partially rebuilding a connection and so on. Most of the proposed algorithms make particular optimization choices based on specific assumptions about network parameters and topologies. There has also been a lot of activity in exploring how the ATM UNI and NNI signaling interfaces need to be augmented to support the capability to setup a call to mobile terminals, trigger VC rerouting etc. However, there has only been a limited amount of prototyping or experimentation with any of the proposed mechanisms making it difficult to quantify what kind of performance might be achievable with mobility-enhanced ATM signaling and how it might impact the performance seen by applications, specially telephony.

The UNI (and NNI) signaling protocols implemented in our testbed support terminal mobility using mechanisms described in [15] and [6]. The sequence of events leading up to and following a handoff are as follows. The MAC layer at the basestation periodically transmits beacons. At a mobile terminal, the MAC layer measures the signal strength for beacons received from each basestation. When the signal strength from the current basestation is much less than the signal strength from another basestation, the MAC layer decides to trigger a *handoff*. This causes the mobile to register with the new basestation. Subsequently, the new basestation generates a REBUILD request which triggers rerouting of active connection(s) in the following manner - see Figure 3.

The REBUILD request in initially forwarded to the old basestation. From here, a REBUILD_RELEASE message is generated and propagated along the original path of a connection (like a normal RELEASE message) with each switch on the path of the connection making a determination about whether it should act as a *cross-over-switch*. If not, then a switch processes the message like a normal RELEASE message. If yes, then the cross over switch attempts to setup a VC segment towards the new basestation. Modifying how a cross-over switch is identified, allows the above protocol to be used to implement various rerouting policies such as Extend, Total Rebuild, Partial Rebuild to a Fixed COS etc.

Our implementation incorporates several optimizations to improve performance in the wireless environment and ensure correct operation even if a mobile terminal were to

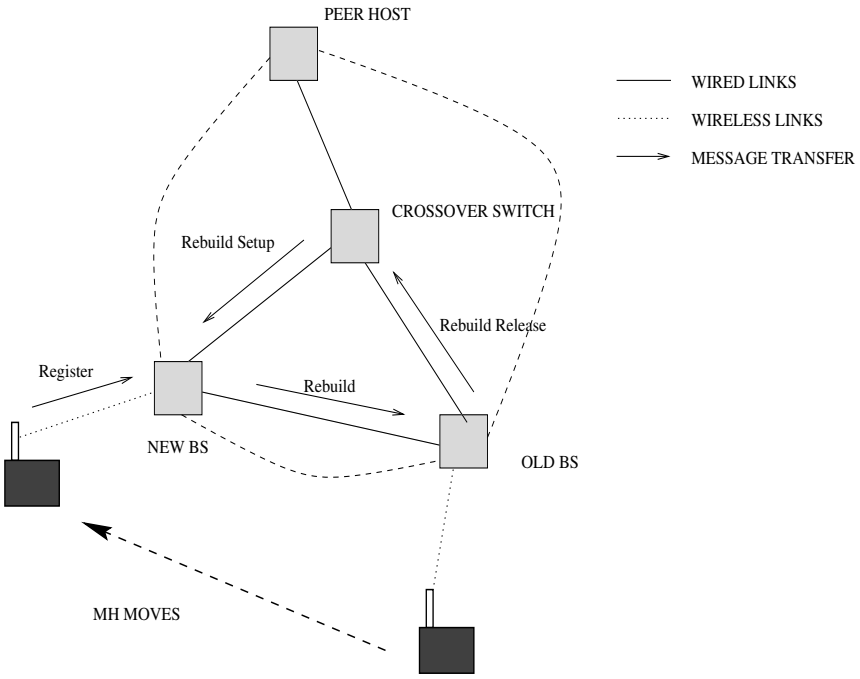


Figure 3. Sequence of steps for VC rerouting

undergo a series of rapid handoffs. For example, the initial registration message carries information about all the virtual circuits terminating at the mobile that need to be rerouted (instead of a separate message for each VC as would be the case with standard ATM signaling). This helps in reducing the number of messages that need to be sent over the wireless link thereby reducing handoff latencies. Also, unlike standard point-to-point ATM signaling, messages over the link between the mobile terminal and the base station do not use SSCOP for ensuring reliable delivery of signaling messages. Instead the signaling protocol is designed to explicitly deal with the possibility of lost messages. The use of SSCOP would require the peer SSCOP entities at the mobile terminal and the new base station to reinitialize their state, following every handoff. This requires an additional *handshake* over the slow wireless link, thereby unnecessarily increasing the handoff latency. Using SSCOP also implies that the signaling layer is unable to immediately detect the successful transmission of a message. This knowledge can be necessary for correct operation in situations where a mobile terminal moves subsequent to transmitting a signaling message [6].

5 Performance Results

We have run a variety of experiments to analyze the performance of the various components of our testbed, including the WaveLAN physical layer, the MAC protocol, the handoff policies etc. In this paper, we present results that attempt to quantify the feasibility of supporting mobile wireless packet telephony. Most of the measurements reported in this paper are with the basestations and terminals located in the same room (approx. 60 feet x 20 feet).

5.1 Performance of MAC protocol

The performance of the scheduled access MAC is determined by both the characteristics of the hardware platform and various implementation artifacts, as well as the parameters chosen for the scheduling slot allocation policy. To isolate these two effects, we have conducted two sets of experiments. First, we quantified the packet transmit/receive latencies and the maximum achievable throughput over a point-to-point WaveLAN link with the scheduling access MAC disabled and with only a single transmitter active at a time[4]. Subsequently, we measure the performance with the scheduled access MAC enabled and with multiple active transmitters. The key performance metrics are the aggregate utilization of the link and the throughput/delay seen by applications.

Table 1 enumerates the results with the scheduled access MAC disabled. The latency measurements are made using an "echo-mode" in the WaveLAN device driver. The mobile terminal initially transmits a packet towards the bases-

[4] i.e. a transmitter follows the CSMA/CA protocol and can transmit a packet instantly, if it senses that the channel is idle.

Packet size (bytes)	Round trip time (RTT) (ms)	Throughput (Mb/s)
98	4.3	0.2
512	11.7	0.6
1482	27.1	1.2

Table 1. WaveLAN link measurements

tation. The basestation immediately reflects the packet back to the terminal, which in turn reflects it, and so on for a large number of iterations. Table 1 enumerates the average *round trip times (rtt)* for various values of the packet size. The minimal round trip latency[5] is around 4.3 ms. On the transmit side, the transmission latency has two principal components: the time taken to transfer a packet from the host to the adapter and the time taken by the adapter to access the channel and transmit the packet over the air. The same is true on the receive side. Measurements indicate that the host to adapter transmit/receive latency is the dominant component of the packet transfer time - for our testbed these values are approximately 2 ms. This arises primarily because the WaveLAN PCMCIA network adapters have a relatively slow host-interface.

The maximum achievable throughput over the link is measured using a block-mode transfer in which a mobile terminal tries to send data as quickly as it can, and without any competing traffic. These values are also enumerated in Table 1. The maximum throughput that is attainable (with large packets) is around 1.2 Mb/s. Since the raw physical data rate that a WaveLAN radio is capable of is 2 Mb/s and the physical layer and MAC headers/trailers add only about 3% overhead for large packets, this throughput limitation also arises primarily from the (slow) speed host interface.

Our next set of experiments attempt to quantify the packet transmit/receive latencies and the maximum achievable throughput over a point-to-point WaveLAN link when using the scheduled access token-passing MAC protocol. Since the token passing MAC uses a frame based scheduling policy, the delays and throughput seen by a connection depend on whether it is a best effort or a guaranteed service connection. For best effort connections, the delay and throughput depend on the volume of traffic contending for the link. For guaranteed service connections, the maximum delay seen by each packet is determined by how often the connection has to wait between successive packet transmissions. This depends on the slot duration, the rate of the connection and may also depend on how multiple slots available to a connection are assigned within a frame. The values of the slot duration and the frame duration also determine the granularity with which bandwidth can be allocated.

As a consequence of implementing the MAC protocol

[5]The WaveLAN hardware enforces a minimum packet size of 64 bytes

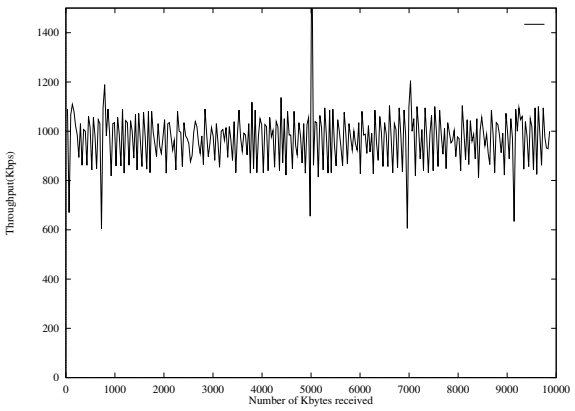


Figure 4. Throughput of single best effort VC

in the WaveLAN device driver running on the host, we are constrained to use relatively large slot durations, e.g. of the order of 5 ms, since this is approximately the empirically measured upper bound on the time taken to grant a token and get it back - see Table 1. For a slot duration of 5 ms and a frame duration of 100 ms, the bandwidth allocation granularity is approximately 64 kb/s but the maximum delay seen by a packet for a 64 kb/s connection may be as much as 100 ms. Increasing the frame duration allows bandwidth to be allocated in smaller quanta but increases the maximum delay, and vice versa. Since, the target end-to-end round trip delay for telephony is about 300 ms, a delay of 100 ms over a wireless link appears to be excessive. However, in practice we get much lower delays on the average as illustrated by the next set of experiments.

These experiments are designed to measure the delay and throughput seen by best-effort[6] and guaranteed service connections using the token-passing MAC for the following three test configurations:

1. single best effort connection from a mobile terminal to basestation

2. three best effort connections from three different mobile terminals to basestation

3. single best effort connection from a mobile terminal to basestation and single 64 kb/s guaranteed bandwidth connection from another mobile terminal to basestation

Figure 4 shows the throughput for a single best effort connection, as measured at the receiver. The average throughput is around 1.1 Mb/s. This is slightly

[6]For a best effort connection, there is an infinite amount of data to send.

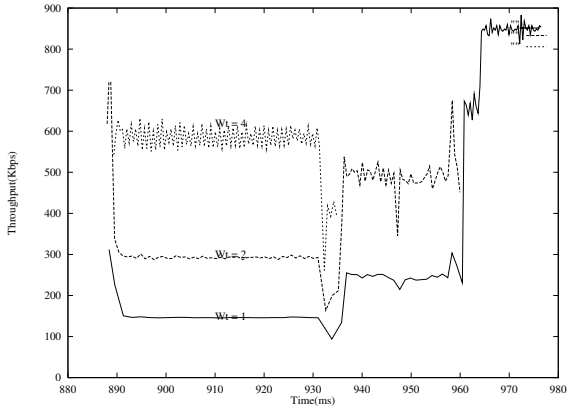


Figure 5. Throughput of three best effort VCs

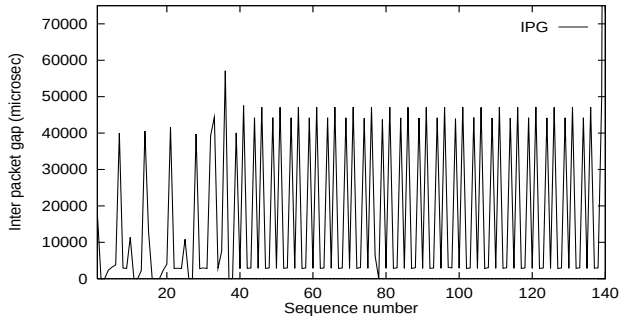


Figure 6. Inter packet delay for a 64 kb/s guaranteed bandwidth connection

lower than the throughput shown in Table 1 with the native WaveLAN MAC because of the overhead due to the token-passing. With three best effort connections (between three different terminals and the basestation) the aggregate throughput stays at about 1.1 Mb/s. In this experiment, the fair-queueing scheduling policy used to determine token-allocation[7] was configured with weights in the ratio 1:2:4 for each of the three VCs. Figure 5 illustrates that the throughput seen by each connection is indeed in proportion to the assigned weights. With a single best effort connection and a single (64 kb/s) guaranteed bandwidth connection, the throughput seen by the best effort connection drops to about 400 kb/s. The delay seen by the voice connection ranges from a few microseconds to several tens of milliseconds - see Figure 6.

5.2 Effect of Handoffs

Our final set of results quantifies the effect of a handoff when a mobile terminal moves from one basestation to another.

[7] We used Start-Time Fair Queueing.

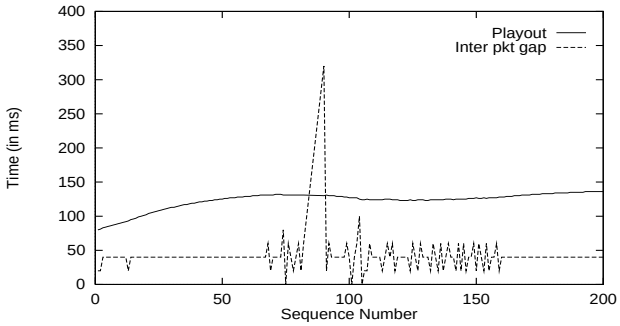


Figure 8. Inter packet delay and playout variation with *vat* during a handoff

Figure 7 illustrates the timeline for a handoff to complete when there is a single active connection. The rerouting policy used here is a FULL REBUILD, i.e. the entire connection is torn down and rebuilt from the peer terminal. For this experiment, the end to end path traverses only 3 links. The total handoff latency is about 32 ms[8]. Out of this about 12 ms is taken up in tearing down the old VC segment and setting up a new VC segment over the wired part of the network. The remaining time is spent in the initial registration handshake and final confirmation of the reroute.

As described earlier, it is possible to try to ensure lossless handoffs, by buffering in-transit data on VC segments being torn down. This reduces the loss rate at the penalty of increasing the delay variance. We attempted to verify whether these options affected audio quality, by running the *vat* telephony application between a mobile terminal and another host, as the mobile terminal underwent handoffs. This experiments was done using PCM encoding and with vat's silence suppression option turned off. vat incorporates an adaptive playback mechanism at a receiver to determines when an incoming packet is ready for presentation. The goal of this playback process is to compensate for delay jitter introduced by the network and restore the inter packet timing pattern as seen at the sender. Packets that arrive after their playout deadline are discarded. The playback time is varied over time using estimates of the mean and variance of the per-packet delay.

Figure 8 plots the inter-packet gap, for packets arriving at the receiver immediately prior to and following the handoff. It also computes the playout time as computed by vat for each arriving packet. Typically, only one or two packets are dropped during a handoff (vat was generating a 355 byte packet every 40 ms) Perceptually, the effect of these losses are barely noticeable with or without lossless hand-

[8] As a point of comparison, handoffs in the cellular network can take several hundred milliseconds.

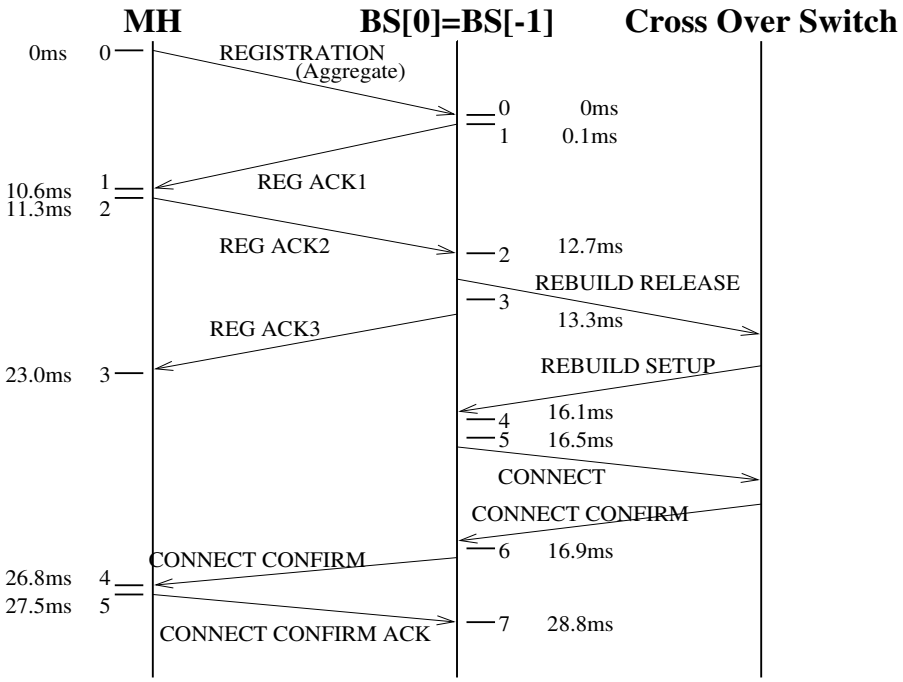


Figure 7. Call Reroute Timings

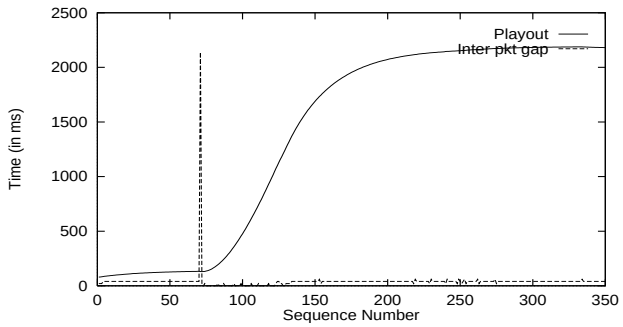


Figure 9. Inter packet delay and playout variation with *vat* forlarge handoff delays

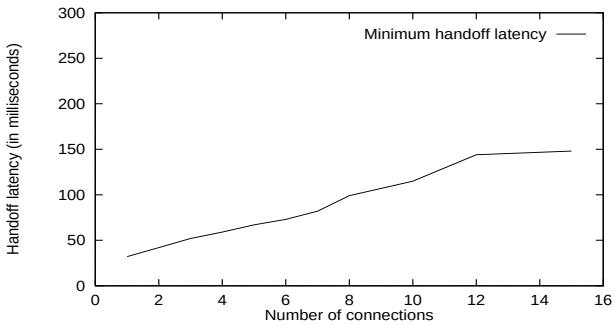


Figure 10. Handoff latency vs. number of connections requiring handoff

off. In the case, where in-transit packets are buffered rather than dropped, the playout value increases slightly due to the increase in the delay variance.

However, if the handoff latency is artificially increased by introducing an additional delay of about 2 seconds, then the use of the lossless handoff option causes the the playout delay computed by vat to increases abruptly following the handoff and to stay elevated, as shown in fig 9.

Another experiment we did was to study how the handoff latency changed as the number of connections terminating at a mobile terminal was increased - see Fig 10. As might be expected, the handoff latency increases with the number of connections requiring rerouting. The primary cause for this increase is because of the *contention* between the signaling messages for each connection, e.g CONNECT_CONFIRM and CONNECT_CONFIRM_ACK for transmission over the slow wireless link. Note that since there is no explicit reservation for signaling messages in the current implementation, contention is possible. In this situation, the CSMA mode of the WaveLAN card becomes active resulting in exponential back-offs after collisions and an increase in the transmission time over the air.

These results suggest that it may be useful to use aggregated signaling for the entire rerouting process, and not just for the initial handshake over the wireless link. Also, this suggests that it may be useful to allow some types of

connections, e.g those used for telephony, to complete their handoffs faster. This may be achieved by allowing signaling messages associated with a connection to use the bandwidth reserved for the data transfer over that connection.

6 Related Work

Providing integrated services to mobile terminals has attracted considerable interest in the research community. Much of this work has involved the design of new QoS-oriented MAC algorithms [7, 23, 17]. Typically, these offer some form of scheduled MAC access, with slots assigned centrally by the base station. As an example, the NEC [23] algorithm provides a comprehensive set of QoS-controlled services. Constant bitrate connections are assigned slots periodically, while available bitrate connections are handled on a burst-by-burst basis. The IEEE 802.11 draft standard for wireless LANs also includes as an option a "point coordination function" which provides a centralized scheduler to support time-sensitive traffic. More recent work has explored adapting wireline fair scheduling algorithms to deal with wireless characteristics such as bursty errors and location-dependent capacity [12].

Efficient solutions for data rerouting and handoff have been explored for both ATM and IP networks. Several research groups [2, 22] have explored the design of wireless ATM LANs, with the goal of extending ATM's QoS properties to the wireless link. There has also been work on specifying appropriate ATM signaling extensions to handle mobility, and comparing techniques for rerouting such as VC extension, rebuild, partial rebuild and loop removal. In IP networks, the Mobile-IP standard [21] defines a procedure for rerouting packets to terminals which are mobile. Mobile-IP was not originally designed to support the delay requirements of telephony. There have been mechanisms proposed to extend mobile-IP to support voice traffic using a hierarchical handoff scheme to avoid noticeable disruption to voice traffic [5].

Several research groups have built systems to explore networking and terminal issues of mobility and wireless access. The SWAN [2] (Seamless Wireless ATM Network) system at Bell Labs used custom-built interface adapters as part of an indoor network extending ATM connectivity to laptops and custom-built personal terminals. The focus was the design and implementation of protocol software to handle wireless to wireline interworking and VC rerouting. Infopad [16] at UC Berkeley built a system to explore indoor delivery of multimedia data to mobile terminals. The focus was designing a fairly dumb terminal called a pad, which was low-power and effectively just performed I/O but no general purpose computing. Also explored was the supporting stationary-side infrastructure, including pad managers, filtering proxies, and servers for handling pen input.

In summary, research on wireless integrated services has addressed an array of interesting issues, primarily yielding new proposals for MAC and rerouting algorithms. Our work builds on these results, adopting a pragmatic view in designing a low-cost experimental testbed and focusing our research effort on supporting packet telephony as part of a wireless integrated service model.

7 Conclusions

In this paper, we have described an experimental testbed designed to support integrated wireless voice-data access. It consists of off-the-shelf network adapters, PCs as basestations, and handheld mobile terminals. A key component of this testbed is a token-based MAC protocol designed to support integrated voice-data transport and implemented using off-the-shelf WaveLAN cards. Another important testbed element are the handoff and VC rerouting protocols that have been optimized to provide predictable and fast handoff - essential features for packet telephony services.

Our experimental results have analyzed the MAC protocols in terms of throughput and latency performance. We have also measured the effect of handoff interruption on packet audio applications, and especially its impact on playback buffer operation. We compared the trade-off between tolerating a small amount of packet loss during a handoff to the additional delay incurred by buffering packets to avoid losses. For telephony our results suggest that the low latency of handoff makes it better to drop packets rather than suffer the increased delay by perturbing playback buffer operation.

The essential components and network protocols for our integrated services wireless testbed are fully operational. Future work will focus mainly on issues related to providing telephony service, terminal design and use, and music delivery. In addition to predictable and low-latency transport, providing telephony service to mobile terminals requires support for location management, call signaling, interworking with wireline packet (and PSTN) telephony, and implementing calling features. The nature of handheld terminals prompts work on speech-based user interfaces for data retrieval and handling telephone calls, as well as using proxies to adapt to variations in the quality of the wireless link and terminal capabilities. Finally, for networked music delivery, latency may be more relaxed, but losses are harder to tolerate due to high compression factors. We have begun to investigate these issues using an audio decoder running on the PC110 and accessing CD-quality audio from a web site.

References

[1] A. Acampora and M. Naghshineh. "An Architecture

and Methodology for Mobile-Executed Handoff in Cellular ATM networks." In IEEE Journal of Selected Area in Communications, vol. 12, no. 8, pp. 1365-1375, Oct. 1994.

[2] P. Agrawal, E. Hyden, P. Krzyzanowski, P.P. Mishra, M.B. Srivastava, J. A. Trotter "SWAN: A Mobile Multimedia Wireless Network" IEEE Personal Communications, pp. 18-33, April 1996.

[3] R.Ahuja, S.Keshav and H.Saran. "Design and Implementation of Native Mode ATM Stack." IEEE Transactions on Networking, August 1995.

[4] S. Biswas and A. Hopper. "A Connection Management Scheme for a Mobile Radio LAN." In Proceedings of the IEEE International Conference on Personal Wireless Communications, Bangalore, India, Aug. 1994.

[5] R. Caceres, V.N. Padmanabhan "Fast and Scalable Handoffs for Wireless Internetworks" Proc. of ACM MobiCom, Nov. 1996.

[6] C. Kalmaner, P. P. Mishra and M.B. Srivastava. "ATM Virtual Circuit Rerouting to support Mobile Host Roaming". ATM_FORUM contribution #97-0155, Feb 1997, San Diego, California.

[7] M.J. Karol, Z. Liu, K.Y. Eng "An efficient demand-assignment multiple access protocol for wireless packet (ATM) networks" ACM/Baltzer Wireless Networks, 1(3):267–279, 1995.

[8] K. Keeton, B. Mah, S. Seshan, R. Katz, and D. Ferrari. "Providing connection-oriented services to mobile hosts." In Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing, pages 83-102, Cambridge, Massachusetts, August 1993.

[9] C.R. Kalmanek, H. Kanakia, S. Keshav "Rate Controlled Servers for Very High-Speed Networks." GlobeCom, San Diego California, Dec 1990.

[10] S.Keshav and H.Saran. "Semantics and Implementation of a Native Mode ATM Protocol Stack." AT&T Bell Labs. Technical Report, 1994.

[11] M. Laubach. "Classical IP and ARP over ATM" NIC, Request For Comments 1577, 1994.

[12] S. Lu, V. Bharghavan, R. Srikant "Fair Scheduling in Wireless Packet Networks" Proc. of ACM SIGCOMM 1997

[13] K. Mahajan and P. S. Garg. "Mobile ATM: QoS over the Wireless Hop." Technical Report, Department of Computer Science and Engineering, IITD, 1997.

[14] P. P. Mishra and M. B. Srivastava. "Call Establishment and Rerouting in Mobile Computing Networks." AT&T Bell Laboratories Technical Memorandum 11384-940906-13TM, September 1994.

[15] Partho P. Mishra and Mani B. Srivastava. "Effect of Virtual Circuit Rerouting on Application Performance." Proceedings of IEEE International Conference on Distributed Computing Systems, Baltimore, Maryland, May 1996.

[16] S. Narayanaswamy, S. Seshan, E. Brewer, R. Brodersen, F. Burghardt, A. Burstein, Y-C. Chang, A. Fox, J. M. Gilbert, R. Han, R. H. Katz, A. C. Long, D. G. Messerschmitt, J. Rabaey "Application and network support for InfoPad" IEEE Personal Comms. Mar 1996

[17] K.S. Natarajan "A hybrid medium access control protocol for wireless LANs" Proc IEEE Int Conf on Selected Topics in Wireless Communications, pp 134-137, June 1992.

[18] B. Rajagopalan. "Mobility Management in Integrated Wireless-ATM Networks." In Proceedings of MobiCom `95, Berkeley, California, November 1995.

[19] S. Seshan et. al. "Experiences with Hand-Offs in the Daeduls System." ACM Baltzer Wireless Networking Journal, December 1995.

[20] Mani B. Srivastava and Partho P. Mishra. "On Quality of Service in Mobile Wireless Networks." Proc. of Network and Operating System Support for Digital Audio and Video (NOSSDAV), St Louis, Missouri, May 1996.

[21] C.E. Perkins, P. Bhagwat "A mobile networking system based on the Internet protocol" IEEE Personal Comms, pp 32–41, First quarter 1994.

[22] J. Porter and A. Hopper "An overview of the ORL Wireless ATM System" Proc. of IEEE ATM Workshop, Washington DC, Sept. 1995.

[23] D. Raychaudhuri "Wireless ATM networks: architecture, system design and prototyping" IEEE Personal Communications pp. 42-49, August 1996.

[24] Network Working Group. RFC on GSMP. Version 1.0.

[25] http://www.attws.com/nohost/data/pocketnet/

[26] http://www.nokia.com/com9000/

A Comparison of SIP and H.323 for Internet Telephony

Henning Schulzrinne
Dept. of Computer Science, Columbia University
New York, NY 10027
hgs@cs.columbia.edu

Jonathan Rosenberg
Bell Laboratories
Holmdel, NJ 07733
jdrosen@bell-labs.com

Abstract—**Two standards have recently emerged for signaling and control for Internet Telephony. One is ITU Recommendation H.323, and the other is the IETF Session Initiation Protocol (SIP). These two protocols represent very different approaches to the same problem: H.323 embraces the more traditional circuit-switched approach to signaling based on the ISDN Q.931 protocol and earlier H-series recommendations, and SIP favors the more lightweight Internet approach based on HTTP. In this paper, we compare SIP and H.323 on complexity, extensibility, scalability, and features.**

I. Introduction

In order to provide useful services, Internet telephony requires a set of control protocols for connection establishment, capabilities exchange, and conference control. Currently, two protocols exist to meet this need. One is ITU-T H.323, and the other is the IETF Session Initiation Protocol (SIP). In this paper, we compare the two protocols on complexity, extensibility, scalability, and services.

The ITU H.323 series of recommendations ("Packet Based Multimedia Communications Systems") defines protocols and procedures for multimedia communications on, among other things, the Internet. It includes H.245 for control, H.225.0 for connection establishment, H.332 for large conferences, H.450.1 H.450.2 and H.450.3 for supplementary services, H.235 for security, and H.246 for interoperability with circuit-switched services. H.323 started out as a protocol for multimedia communication on a LAN segment without QoS guarantees, but has evolved to try and fit the more complex needs of Internet telephony.

H.323 is based heavily on the ITU multimedia protocols which preceded it, including H.320 for ISDN, H.321 for B-ISDN, and H.324 for GSTN terminals. The encoding mechanisms, protocol fields, and basic operation are somewhat simplified versions of the Q.931 ISDN signaling protocol.

The Session Initiation Protocol (SIP) [1], developed in the MMUSIC working group of the IETF, takes a different approach to Internet telephony signaling by reusing many of the header fields, encoding rules, error codes, and authentication mechanisms of HTTP.

In both cases, multimedia data will likely be exchanged via RTP, so that the choice of protocol suite does not influence Internet telephony QOS.

II. Complexity

H.323 is a rather complex protocol. The sum total of the base specifications alone (not including ASN.1 and PER) is 736 pages. SIP, on the other hand, along with its call control extensions and session description protocols totals merely 128 pages. H.323 defines hundreds of elements, while SIP has only 37 headers (32 in the base specification, 5 in the call control extensions), each with a small number of values and parameters, but that contain more information. A basic, but interoperable SIP Internet telephony implementation can get by with four headers (To, From, Call-ID, and CSeq) and three request types (INVITE, ACK, and BYE) and is small enough to be assigned as a homework programming problem. A fully functional SIP client agent, with a graphical user interface, has been implemented in just two man-months.

H.323 uses a binary representation for its messages, based on ASN.1 and the packed encoding rules (PER). ASN.1 generally requires special code-generators to parse. SIP, on the other hand, encodes its messages as text, similar to HTTP [2] and the Real Time Streaming Protocol (RTSP) [3]. This leads to simple parsing and generation, particularly when done with powerful text processing languages such as Perl. The textual encoding also simplifies debugging, allowing manual entry and perusing of messages. Its similarity to HTTP also allows for code-reuse; existing HTTP parsers can be quickly modified for SIP usage.

H.323's complexity also stems from its use of several protocol components. There is no clean separation of these components; many services require interactions between several of them. (Call forward, for example, requires components of H.450, H.225.0, and H.245.) The use of several different protocols also complicates firewall traversal. Firewalls must act as application level proxies [4], parsing the entire message to arrive at the required fields. The operation is stateful since several messages are involved in call setup. SIP, on the other hand, uses a single request that contains all necessary information.

H.323 also provides for an array of options and methods for accomplishing a single task. For example, there are three distinct ways in which H.245 and H.225.0 may be used together: the original H.323v1 approach of separate connections, H.245 tunneling through H.225.0, and FastStart in H.323v2. In the original approach, the call signaling channel is set up first, the H.245 control channel is established, and finally the media channels are opened. This can require many round trips for call setup. FastStart includes the media channel information in the original call invitation, avoiding the need to open the H.245 channel. In H.245 tunnelling, the H.245 channel is still used, but its messages are carried over the call signaling channel. Even though FastStart is much more efficient, H.323 allows any of

the three and thus, firewalls, end systems, gatekeepers, and gateways must support all of them. As with any protocol, large option spaces lead to feature interaction and the need for profiles. (How does encryption of the H.245 channel work when its tunneled through H.225.0, for example?).

An additional aspect of H.323's complexity is its duplication of some of the functionality present in other parts of the protocol. In particular, H.323 makes use of RTP and RTCP. RTCP has been engineered to provide various feedback and conference control functions in a manner which scales from two-party conferences to thousand-party broadcast sessions. H.245, however, provides its own mechanisms for both feedback and simple conference control (such as obtaining the list of conference participants). These H.245 mechanisms are redundant, and have been engineered for small to medium-sized conferences only.

III. Extensibility

Extensibility is a key metric for measuring an IP telephony signaling protocol. Telephony is a tremendously popular, critical service, and Internet telephony is poised to supplant the existing circuit switched infrastructure developed to support it. As with any heavily used service, the features provided evolve over time as new applications are developed. This makes compatibility among versions a complex issue. As the Internet is an open, distributed, and evolving entity, one can expect extensions to IP telephony protocols to be widespread and uncoordinated. This makes it critical to build in powerful extension mechanisms from the outset.

SIP has learned the lessons of HTTP and SMTP (both of which are widely used protocols that have evolved over time), and built in a rich set of extensibility and compatibility functions. By default, unknown headers and values are ignored. Using the Require header, clients can indicate named feature sets that the server must understand. When a request arrives at a server, it checks the list of named features in the Requires header. If any of them are not supported, the server returns an error code and lists the set of features it does understand. The client can then determine the problematic feature and fall back to simpler operation. The feature names are based on a hierarchical namespace, and new feature names can be registered with IANA. This means that any developer can create new features in SIP, and then simply register a name for them. Compatibility is still maintained across different versions.

To further enhance extensibility, numerical error codes are hierarchically organized, as in HTTP. There are six basic classes, each of which is identified by the hundreds digit in the response code. Basic protocol operation is dictated solely by the class, and terminals need only understand the class of the response. The other digits provide additional information, usually useful but not critical. This allows for additional features to be added by defining semantics for the error codes in a class, while achieving compatibility.

The textual encoding means that header fields are self-describing. It is self-evident what the meaning of the To, From, and Subject fields are. As new header fields are added in various different implementations, developers in other corporations can determine usage just from the name, and add support for the field. This kind of distributed, documentation-less standardization has been common in the Simple Mail Transfer Protocol (SMTP), which has evolved tremendously over the years.

As SIP is similar to HTTP, mechanisms being developed for HTTP extensibility can also be used in SIP. Among these are the Protocol Extensions Protocol (PEP), which contains pointers to the documentation for various features within the HTTP messages themselves.

H.323 provides extensibility mechanisms as well. These are generally nonstandardParam fields placed in various locations in the ASN.1. These params contain a vendor code, followed by an opaque value which has meaning only for that vendor. This does allow for different vendors to develop their own extensions. However, it has some limitations. First, extensions are limited only to those places where a non-standard parameter has been added. If a vendor wishes to add a new value to some existing parameter, and there is no placeholder for a nonstandard element, one cannot be added. Secondly, H.323 has no mechanisms for allowing terminals to exchange information about which extensions each supports. As the values in non-standard parameters are not self-describing, this limits interoperability among terminals from different manufacturers.

In addition, H.323 requires full backwards compatibility from each version to the next. As various features come and go, the size of the encodings will only increase. However, SIP allows for older headers and features to gradually disappear as they are no longer needed, keeping the protocol and its encoding clean and concise.

A critical issue for extensibility are audio and video codecs. There are hundreds of codecs that have been developed, many of which are proprietary. SIP uses the Session Description Protocol (SDP) to convey the codecs supported by an endpoint in a session. Codecs are identified by string names, which can be registered by any person or group with IANA, and then used. This means that SIP can work with any codec, and other implementations can determine the name of the codec, and contact information for it, from IANA.

In H.323, each codec must be centrally registered and standardized. Currently, only ITU developed codecs have codepoints. As many of these carry significant intellectual property, there is no free, sub-28.8 kb/s codec which can be used in an H.323 system. This presents a significant barrier to entry for small players and universities.

Furthermore, SIP allows for new services to be defined through a few powerful third-party call control mechanisms. These mechanisms allow a third party to instruct another entity to create and destroy calls to other entities. As the controlled party executes the instructions, status messages are passed back to the controller. This allows the controller to take further actions based on some local program execution. This is much like the IN model in traditional telephony. As there are hundreds of telephony services currently defined, it is unreasonable to attempt to write specifications for each. SIP allows these services to be deployed by basing them on simple, standardized mechanisms. These mechanisms can be used to construct a variety of services, including blind transfer, operator assisted transfer, three-party calling, bridged calling, dial-in bridging, multi-unicast to multicast transitions, ad-hoc bridge invitation and transition, and various forwarding variations [5].

As an example of these extension and service creation mechanisms, the PSTN and Internet Internetworking (pint) working group in IETF is defining a simple SIP extension for click-to-call type of services. In this scenario, a user at a web page clicks on a button, and a PSTN entity connects the user's telephone to a customer service rep. This requires a control protocol between the web server and a PSTN-enabled device. SIP is being used as this protocol.

H.323 does provide some basic mechanisms along this line. The FACILITY message allows a callee to direct a caller to contact a different party (basically, a blind transfer). Another is the H.245 CommunicationModeCommand, which allows the MC to change the media encodings for a conference for the various participants. The former is fairly limited in scope, and the latter can only be executed by the MC for the call. Neither provide generic third party control mechanisms needed for building complex services.

Another aspect of extensibility is modularity. Internet telephony requires a large number of different functions; these include basic signaling, conference control, quality of service, directory access, service discovery, etc. One can be certain that mechanisms for accomplishing these functions will evolve over time (especially with regards to QoS). This makes it critical to apportion these functions to seperate, modular, orthogonal components, which can be swapped in and out over time. It is also critical to use seperate, general protocols for each of these functions. This allows for the function to be duplicated in other applications with ease. For example, it is more efficient to have a single QoS mechanism which is application independent, rather than invent a new QoS protocol or mechanism for each application.

SIP is reasonably modular. It encompasses basic call signaling, user location, and registration. Advanced signaling is part of SIP, but within a single extension. Quality of service, directory accesses, service discovery, session content description, and conference control are all orthogonal, and reside in separate protocols. For example, it is possible to use the H.245 capability description elements in SIP, with no changes to SIP at all.

H.323 is less modular. It defines a vertically integrated protocol suite for a single application. The mix of services provided by the H.323 components encompass capability exchange, conference control, maintenance operations, basic signaling, quality of service, registration, and service discovery. Furthermore, these are intertwined within the various sub-protocols within H.323.

SIP's modularity allows it to be used in conjunction with H.323. A user can use SIP to locate another user, taking advantage of its rich multi-hop search facilities. When the user is finally located, they can use a redirect response to an H.323 URL, indicating that the actual communication should take place with H.323.

IV. Scalability

We also find that H.323 and SIP differ in terms of scalability. We can observe scalability on a number of different levels:

Large Numbers of Domains: H.323 was originally conceived for use on a single LAN. Issues such as wide area addressing and user location were not a concern. The newest version defines the concept of a zone, and defines procedures for user location across zones for email names. However, for large numbers of domains, and complex location operations, H.323 has scalability problems. It provides no easy way to perform loop detection in complex multi-domain searches (it can be done statefully by storing messages, which is not scalable). SIP, however, uses a loop detection algorithm similar to the one used in BGP, which can be performed in a stateless manner.

Server Processing: In an H.323 system, both telephony gateways and gatekeepers will be required to handle calls from a multitude of users. Similarly, SIP servers and gateways will need to handle many calls. For large, backbone IP telephony providers, the number of calls being handled by a large server can be significant.

In SIP, a transaction through several servers and gateways can be either stateful or stateless. In the stateless model, a server receives a call request, performs some operation, forwards the request, and completely forgets about it. SIP messages contain sufficient state to allow for the response to be forwarded correctly. Furthermore, SIP can be carried on either TCP or UDP. In the case of UDP, no connection state is required. This means that large, backbone servers can be based on UDP and operate in a stateless fashion, reducing signficantly the memory requirements and improving scalability.

H.323, on the other hand, requires gatekeepers (when they are in the call loop), to be stateful. They must keep call state for the entire duration of a call. Furthermore, the connections are TCP based, which means a gatekeeper must hold its TCP connections for the entire duration of a call. This can pose serious scalability problems for large gatekeepers.

Furthermore, a gateway or gatekeeper will need to process the signaling messages for each call. The simpler the signaling, the faster it can be processed, and the more calls a gateway or gatekeeper can support. As SIP is simpler to process than H.323, SIP should allow more calls per second to be handled on particular box than H.323. [1]

Conference Sizes: H.323 supports multiparty conferences with multicast data distribution. However, it requires a central control point (called an MC) for processing all signaling, for even the smallest conferences. This presents several difficulties. Firstly, should the user providing the MC functionality leave the conference, and exit their application, the entire conference terminates. In addition, since MC and gatekeeper functionality is optional, H.323 cannot support even three party conferences in some cases. We note that the MC is a bottleneck for larger conferences. To alleviate this, the latest version of H.323 has defined the concept of cascaded MC's, allowing for a very limited application layer multicast distribution tree of control messaging. This improves scaling somewhat, but for even larger conferences, the H.332 protocol defines additional procedures. This means that three distinct mechanisms exist to support conferences of different sizes. SIP, however, scales to all different conference sizes. There is no requirement for a central MC; conference coordination is fully distributed. This improves scalability and complexity. Furthermore, as it can use UDP as well as TCP, SIP supports native multicast signaling, allowing a single

[1] The authors are not aware of any study measuring the processing overhead of SIP and H.323. however.

Feature	SIP	H.323
Blind Transfer	Yes	Yes
Operator Assisted Transfer	Yes	No
Hold	Yes; through SDP	Not yet
Multicast Conferences	Yes	Yes
Multi-unicast Conferences	Yes	Yes
Bridged Conferences	Yes	Yes
Forward	Yes	Yes
Call Park	Yes	No
Directed Call Pickup	Yes	No

TABLE I
SIP AND H.323 CALL CONTROL FEATURE COMPARISON

protocol to scale from sessions with two to millions of members. *Feedback:* H.245 defines procedures that allow receivers to control media encodings, transmission rates, and error recovery. This kind of feedback makes sense in point-to-point scenarios, but ceases to be functional in multipoint conferencing. SIP, instead, relies on RTCP for providing feedback on reception quality (and also for obtaining group membership lists). RTCP, like SIP, operates in a fully distributed fashion. The feedback it provides automatically scales from a two person point to point conference to huge broadcast style conferences with millions of participants.

V. SERVICES

H.323 and SIP offer roughly equivalent services. Some of the call control services are listed in Table 1.

As can be seen from the chart, SIP and H.323 support similar services. A comparison in these dimensions is somewhat difficult, as new services are always being added to both SIP and H.323. We expect that the above table will be different upon printing of this paper.

In addition to call control services, both SIP (when used with SDP) and H.323, provide capabilities exchange services. In this regard, H.323 provides a much richer set of functionality. Terminals can express their ability to perform various encodings and decodings based on parameters of the codec, and based on which other codecs are in use. However, most implementations don't require (or implement) these, and the basic receiver-capability indication supported by SIP ("choose any subset of these encodings for this list of media streams") seems sufficient and equivalent to current H.323 capabilities actually implemented.

SIP provides rich support for personal mobility services, however. When a caller contacts the callee, the callee can redirect the caller to a number of different locations. Each of these locations can be an arbitrary URL, and contains additional information about the terminal at that location. Information on language spoken, business or home, mobile phone or fixed, and a list of callee priorities, can be conveyed for each location. This allows the caller flexibility in choosing which location to talk to. For non-interactive terminals, the original call setup can convey caller preferences about the nature of the terminal to be contacted. This allows network proxies to forward the call based on these preferences.

SIP also supports multi-hop "searches" for a user. When a call request is made to some particular address, a SIP server is contacted at that address. As this SIP server may not be the machine that the callee is currently residing at, the server can proxy the request to one or more additional servers. These servers, in turn, may further proxy the request until the party is contacted. A server can actually proxy the request to multiple servers in parallel. This allows the search for the user to operate more rapidly. SIP also allows multiple branches of the search to accept the call, passing the responses back to the caller. The caller can then decide which party to speak to. This would allow a call for `j.doe@company.com` to be picked up by both Mr. Doe, his wife, and an answering machine. The caller can then hang up with the answering machine and continue with a three party call, if they so desire.

H.323's support for this kind of mobility is more limited. The facility message can redirect a caller to try several other addresses (much like 300 class response codes in SIP). However, it cannot be used to express preferences, nor can the caller express preferences in the original call invitation. H.323 wasn't engineered for wide area operation; it does support forwarding of call requests among servers, but has no mechanisms for loop detection. H.323 doesn't allow a gatekeeper to proxy a request to multiple servers either.

H.323 supports various conference control services, including chair selection, "mike passing", and conference participant determination. SIP does not provide conference control, relying instead on other protocols for this service. Some simple forms of conference control (such as sending notes around, and obtaining a conference participant listing), are available through RTCP, however.

VI. CONCLUSION

In this paper, we have compared SIP and H.323 in terms of complexity, extensibility, scalability, and services. We have found that SIP provides a similar set of services to H.323, but provides far lower complexity, rich extensibility, and better scalability. Future work is to more fully evaluate the protocols, and examine quantitative performance metrics to characterize these differences.

REFERENCES

[1] M. Handley, H. Schulzrinne, and E. Schooler, "SIP: session initiation protocol," Internet Draft, Internet Engineering Task Force, May 1998, Work in progress.
[2] R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," Request for Comments (Proposed Standard) 2068, Internet Engineering Task Force, Jan. 1997.
[3] H. Schulzrinne, R. Lanphier, and A. Rao, "Real time streaming protocol (RTSP)," Request for Comments (Proposed Standard) 2326, Internet Engineering Task Force, Apr. 1998.
[4] Anonymous, "H.323 and firewalls: The problems and pitfalls of getting H.323 safely through firewalls," Developer note, Intel Corporation, Apr. 1997.
[5] Henning Schulzrinne and Jonathan Rosenberg, "Signaling for internet telephony," Technical Report CUCS-005-98, Columbia University, New York, New York, Feb. 1998.

A new audio device driver abstraction

Dickon Reed
Computer Laboratory,
University of Cambridge,
Cambridge CB2 3QG, U.K
Dickon.Reed@cl.cam.ac.uk

Abstract

Traditional audio device drivers are unable to provide low latency audio services and often do not provide audio mixing services. High quality audio has, compared to other media types, relatively low bandwidth, but is very sensitive to latency and errors. This paper presents a device driver abstraction that exploits the low bandwidth and constant bit rate of audio to provide multiple channels of low latency audio efficiently. It provides applications with a great deal of flexibility with regards to latency, blocking behaviour and mixing.

1 Motivation

Until recently, those operating systems that provided support for audio tended to do so in a rather inflexible manner. Typically, only one client could use the audio service at once and that client was often given little information about the timeliness of the audio delivery. Audio is important in modern multimedia systems; there may be several applications wishing to play audio simultaneously – an MPEG audio decoder playing music, a game, user interface sound effects and a video conference, for instance. Similarly, multiple clients may wish to record; a video conference, speech recognition for the user interface and a volume meter, for example. Some of these applications, such as most computer games, play several effects at once and so have to perform their own mixing on the various audio channels they play.

Some applications are very sensitive to latency. Holding a conversation with audio round trip time of greater than approximately 100ms is distracting. The difference of an attack of 10ms and 50ms is perceptible in music. The delay between a sound off to one side of a human head reaching one ear and reaching the other is about 1ms, yet the brain takes this in to account when locating sound sources. General purpose operating systems are typically unable to handle audio with low latency and jitter and so are not usually used for latency critical audio applications such as musical signal processing in real time.

When an application is processing an audio stream from one audio device with one clock source and playing it on one with another clock source, as will typically be the case in an audio conference, we have observed that the two clocks are not generally running at exactly the same rate. Even on identical hardware, operating temperature differences can have a significant effect. So, if the application has a buffer between the two sources then that buffer will tend to either overflow or underflow. There is no portable way for an application to query the exact playback speed or playout queue length, so there is no portable way of correcting clock rate drift.

2 Approaches to audio device drivers

One of the better documented kernel audio device driver schemes is Sun's [2] which has become a *de facto* standard for Unix. It treats the audio device as a number of special files, which makes it easy to program to but enforces certain restrictions; there will typically only ever be one process which has the audio device open at a time. Furthermore, on audio playback, the programmer is typically unaware of how much audio data has been placed in the file already, and so is unaware of exactly when any particular sample is played. Since Unix programmers are contractually unaware of the passage of real time anyway, this is not too much of an issue in the kind of applications written for Sun's audio system.

Early Microsoft operating systems such as MS-DOS typically did not provide any support for audio, leaving it up to applications to drive sound cards themselves. Therefore, various ad-hoc approaches were taken. These were generally acceptable, but necessarily suffered from the limitations of not using an operating system service. At first, this meant that configuration of the audio device had to be handled per program using audio. Later, as the Windows en-

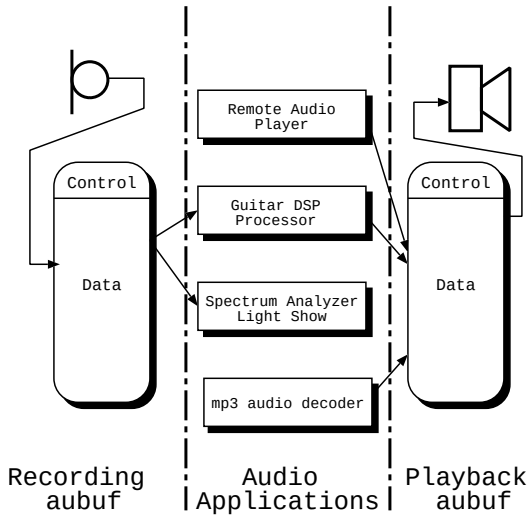


Figure 1. *aubufs* **in action**

vironment was bolted on top of MS-DOS and multitasking was retrofitted, it became clear that operating system support was necessary. Microsoft's first attempts at this were largely ignored by the programmers who had been writing MS-DOS programs with sound support; they still preferred to stay with third party sound driver libraries. Lately, with DirectX 5.0 [1], Microsoft have provided an abstraction that can be used for most applications. The DirectSound abstraction provides support for mixing, in some cases with hardware assistance. Applications may each have multiple channels of audio played simultaneously. Most PC games recently published use this architecture. It does not address latency issues very precisely, however, nor does it appear to provide any indication of the exact play-out rate of the device.

Most systems research dealing with audio have employed a fragment based scheme, sometimes providing direct DMA buffer access. Much work has been performed on higher level synchronisation issues in distributed systems ([6, 5, 3, 7]) which could be applied on top of the abstraction described below.

3 Shared Audio Buffers

Audio buffers, or *aubufs* have one master client and many slave clients, and shared between all of them is one control record belonging to the master but readable by the slaves and one data record readable and writable by all clients. The control record consists of a timestamp, a count and a description of the format of the data stored in the data record. The master reads or writes from the buffer continuously, touching the timestamp and advancing the counter in the control record as it does so. The control block also contains the time at which the master began reading or writing.

For playback, an aubuf works in *reader master mode*; the server reads from the buffer and maintains the control record. Clients use the contents of the control record to calculate where the master is currently in the buffer, then perform scaled add operations to the buffer just after that point. The reader zeroes the buffer as it reads it. For recording, an aubuf works in *writer master mode* in a similar fashion; the server now writes to buffer and still maintains the control record. Clients still use the control record to calculate the position of the master, then read from the data record just before that point.

Slaves do not allow themselves to work more than a certain distance away from the point where they have calculated the master to be at work. The clients are free to use the thread that was playing audio for other purposes, or to block the thread until, for instance, they calculate that half of the data they have placed in the buffer has been used. They then can top up the buffer as they see fit. This provides the backpressure mechanism, without enforcing a blocking policy on applications. The distance that clients work ahead is equivalent to their latency for that audio device and so some applications will wish to minimise the distance. The master never waits for the slaves; if the slaves are not given enough CPU time to operate on the buffer their data will be lost. This scheme therefore requires an operating system capable of giving sufficient time to applications regularly. Hard real time systems should provide this, but more usefully general purpose soft real-time systems providing quality of service guarantees on CPU time should also be applicable, such as Nemesis [4].

The aubuf start up time and counter are used to measure the rate at which the master is operating on the buffer. In cases where audio is being recorded and replayed in real time using audio devices with different clocks, this can be used to provide data for sample rate micro-adjustment, so buffer sizes can be kept under control.

4 Implementation on Nemesis

The above scheme has been implemented on the Nemesis OS running on conventional PCs with Creative Labs Sound Blaster 16 cards, so that timely CPU guarantees can be made for audio applications. Originally, a fragment (i.e. packet) based driver based on the Linux kernel source was used on Nemesis, but the latency was unacceptably high.

Under the new scheme, for playback, the sound card is programmed to repeatedly play CD quality stereo audio from a DMA buffer, issuing an interrupt each time it loops back to the start of the buffer. The device driver has a thread that clears parts of the buffer after the Sound Blaster has played it, because there is no hardware facility to do this. On each interrupt, the aubuf control record's timestamp and

counter are updated. The device driver calculates the observed sample playback rate of the sound card and stores this in the aubuf control record.

When a client wishes to play audio, it reads the aubuf control record. The current position of the audio device in the playback buffer is then calculated by subtracting the aubuf timestamp from the current system time and converting that time into a number of samples using the exact play out rate data stored in the aubuf. The maximum of the sound hardware packet size and the number of samples played in the scheduling period of the client is then added on to the calculated position, giving a position in the buffer the client can safely start writing to. Nemesis is able to guarantee that the applications will be given time to do their processing regularly, independent of whatever else is happening on the system. Note that some clients can be configured to run with higher latency and then can be scheduled less stringently without breaking up.

Clients are allowed to write directly into the data buffer of the aubuf. They perform only addition operations (which are atomic on x86 machines) on the buffer. A number of protection mechanisms could be deployed, such as the secure opcode facility provided on Nemesis for framebuffer access. However, the extra expense would not be worthwhile, because clients could still damage audio playback by placing malicious data in to the buffer. Note that the clients cannot crash the server or the machine by writing bad data in the buffer, so system stability is not affected.

Recording is easier to implement. Any number of clients can easily read from the same aubuf data record, using the aubuf control record to calculate where to start reading from. The sound card runs in a mode whereby it repeatedly records into a buffer, issuing an interrupt that records a timestamp and increments a counter when it passes the top of the buffer, much as before. There are no protection issues with recording, however, as it is not necessary for the clients to have write access to anything. Multiple aubufs may be used by the same application because the aubuf abstraction never requires a client to block.

Applications typically use a library providing implementations of two client-side playing objects– a synchronous and an asynchronous version. The synchronous version never works ahead of the sound card by more than the total size of the DMA buffer. A thread is used by the asynchronous version to perform the actual mixing in to the buffer; the client invokes operations on a proxy object that adds audio data to the potentially long inter-thread queue. The asynchronous version involves context switching between threads, but simplifies construction of some applications. The interface exported by the objects is the same in both cases. However, a `Put` operation is provided to add data to the buffer, that may optionally block or only place whatever data will fit in to the buffer. Query operations are provided to query the play out rate of the master and the current queue length of the client.

5 Preliminary results

In order to measure the achieved latency, a test application writes directly to a PC parallel port in order to trigger a digital storage oscilloscope, then begins to play audio. The delay between the parallel port trigger and the wave form emerging from the sound card is then clearly visible on the oscilloscope and can be measured to about one part in 500. Given the order of magnitude of the latency results obtained, the delay in writing to the parallel port of less than one microsecond can be neglected.

Experiments were performed on a 150Mhz Pentium machine with 16 megabytes of 60ns EDO RAM and a motherboard audio Sound Blaster 16 compatible sound card. The first port of the Linux kernel 2.0 series sound driver to Nemesis achieved a latency of 8.82ms, with some break up, when configured to a fragment size of 64 bytes at CD quality. It was necessary to run with a fragment size of 1024 bytes to obtain reliable audio, with a latency of 35ms. In both cases, the driver was configured to queue 4 outstanding fragments. With other queue lengths, the latency varied as would be expected, but 35ms was the lowest latency that could be achieved reliably.

The same code running native on Linux, configured with a fragment size of 8192 bytes and 3 fragments queue length, was stable with a latency of 139ms. The device driver can be configured down to a fragment size of 16 bytes with 2 fragments queued, but was unstable (i.e. in excess of ten percent of fragments were corrupted) when configured below 100ms. There is probably room for optimisation of these results on Linux.

When testing the new device driver at 44100Hz, latency was 440μs. Figure 2 shows latency results for different sample periods. The Sound Blaster hardware contains a hardware FIFO of undocumented length, which these results tend to indicate has an effective length of 18 stereo samples. This seems to indicate a *bus latency* (i.e. delay between application sending and sound card receiving and queuing the data internally) of around 55μs- i.e. between 2 and 3 samples, but more accurate experimental work is necessary to validate this figure. If the application is set to play any closer to the position of the sound card than this, then the first sample or more is lost. This was observed by having the application produce a sine wave peaking at 25% of maximum volume, but with the first sample being a transient to 75% of maximum volume. The transient is clearly visible on an oscilloscope trace when it is present.

An early, preliminary test of an aubuf application receiving ATM AAL5 CD quality audio from a Fore Systems AVA 200 across two Fore Systems ASX 200 switches, with clock

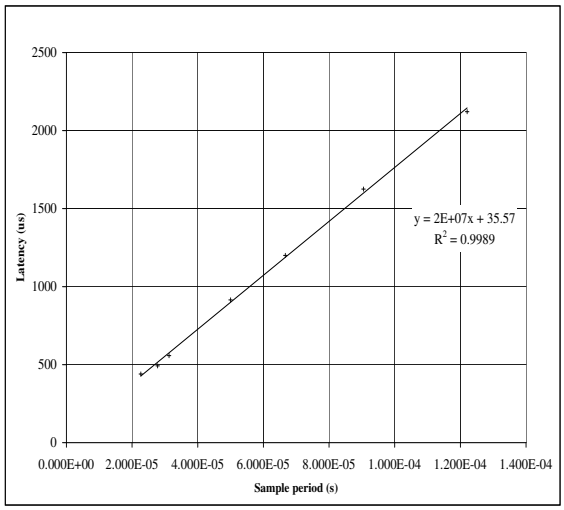


Figure 2. aubuf driver latency

	Time (μs)
Duration of one sample	22.7
Est. bus latency of aubuf driver	36
Est. latency of card FIFO at CD quality	404
Measured latency using aubufs	440
Audio across network to aubuf	4300
Linux driver on Nemesis	35000
Linux driver on Linux	139000

Table 1. Timings for CD quality audio

rate matching, achieved a latency of 4.3ms from a distinctive waveform entering the AVA 200 to the waveform leaving the sound card. (This result was taken before optimisation and several system improvements). Table 1 summarises these work in progress results.

6 Future work

Work is currently in progress to examine achievable end to end latencies using this scheme in various forms of network audio. It is anticipated that very low latencies can be achieved at the expense of forcing the operating system to context switch to the very low latency application frequently. It will be interesting to measure both the effective latencies achievable and the impact in overall system performance of executing some low latency audio clients. Lipsync between an audio stream and a video stream from the same source shall also be examined, as well as synchronisation between different audio streams (for instance for multiple device digital audio hard disk recording and playback). Experiments are also planned to measure latency in audio processing (i.e. recording and playback in the same system simultaneously) and the effect on latency of software signal processing in path. The device driver and client library could be ported to other systems providing accurate timing and CPU guarantees and to other kinds of sound card. More work on measuring and detecting and correcting clock drift is planned. Aubufs may be useful for internal purposes within applications- they may prove a useful in a software synthesiser, for instance. Finally, this device driver abstraction suggests a different design for sound cards, which should mean that protection, mixing and timing can be provided in hardware. For such a piece of hardware, the device driver would only be responsible for initialisation and channel creation.

7 Conclusion

Several clients have been ported to use playback aubufs, including an MPEG audio layer 3 decoder, a software MIDI synthesiser, a computer game, a guitar software digital signal processor and an ATM network audio player. Of these, synthesiser, game and signal processor benefitted from using aubufs particularly, by eliminating a stage of software mixing and a copy they previously had to perform. Aubufs provide a useful abstraction for flexible, efficient and subsample accurate audio processing for operating systems where CPU guarantees can be made to applications. They provide low latency audio services and are well suited to flexible recording as well as playback on standard hardware, because audio packetisation delays are eliminated.

References

[1] Microsoft Developer Network; Direct X 5 Software Development Kit.

[2] SunOS 5.5.1 `audio` man page, section 7.

[3] G. Coulson and G. S. Blair. Meeting the real-time synchronisation requirements of multimedia in open distributed processing. *Distributed Systems Engineering Journal*, 1994.

[4] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas In Communications*, 14(7):1280–1297, September 1996. Article describes state in May 1995.

[5] T. D. C. Little and A. Ghafoor. Synchronisation properties and storage models for multimedia objects. *IEEE Journal on Selected Areas In Communications*, 8(3):413–427, April 1990.

[6] C. Nicolaou. An architecture for real-time multimedia communication systems. *IEEE Journal on Selected Areas In Communications*, 8(3):391–400, April 1990.

[7] M. Papathomas, G. S. Blair, G. Coulson, and P. Robin. Addressing the real-time synchronisation requiements of multimedia in an object-orientated framework. *Multimedia Computing and Networking*, 1995.

Adaptive Resource Management in a Multimedia Operating System

Don Oparah
Computer Laboratory,
University of Cambridge,
Cambridge CB2 3QG, UK
Email: doo20@cl.cam.ac.uk

Abstract

This paper highlights the need for adaptive resource management within the endsystem, in addition to the provision of basic QoS guarantees, in order to maximize the users' perceived performance of their applications at any given time within the limits of the physical resources available. A prototype system developed within the Nemesis multimedia operating system [7] is described. The system provides support for the dynamic adaptation of applications according to user-controlled resource allocation policies. The paper also identifies interesting directions for further work in future.

1. Introduction

To support the ever increasing number of multimedia applications being developed, some work has been done on providing the Quality of Service (QoS) required by such applications in both the network and the endsystem. Operating systems work such as [9] and Nemesis [7] (developed locally) provides support for meaningful QoS guarantees. However we make the following key observations

- QoS work in the endsystem has focussed on moving away from the unpredictability of systems like UNIX by providing a predictable level of service to applications via a QoS guarantee. However most multimedia applications are able to adapt in one or more directions (e.g via frame rate, frame size) and operate in a number of different valid modes. Hence the goal should be to provide a predictable level of service *for the current mode of operation*.
- Hardware improvements notwithstanding, application writers design and users inevitably want to run, a mix of applications whose total resource requirements if run in their highest mode of operation would overload the system.

These observations motivate the need for a resource management system integrated with an operating system such as Nemesis to

- Peform a *balancing act* by dynamically reallocating QoS guarantees for resources amongst adaptive applications in such a way that the current combination of applications, operating at different modes, corresponds to the best possible performance *as perceived by the user*.
- Provide a mechanism which allows the user to express resource allocation *policies* or preferences. This relieves the user of the complicated task of manually attempting to degrade other applications in order to support a new application or an application upgrade while still allowing the user to influence the manner in which the reallocations take place.
- Provide support for adaptive applications.

Our view of the progression of research in endsystem QoS is illustrated in Figure 1. We go on to describe the current state of a resource management architecture designed

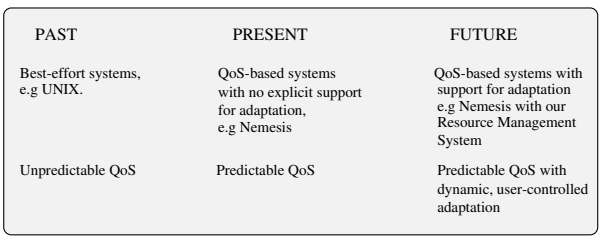


Figure 1. Endsystem QoS research

to provide such support and focus on a prototype which illustrates the main ideas and provides a basis for further discussion.

2. Architectural overview

A simplified view of our resource management architecture is shown in Figure 2. At the centre of the system is the

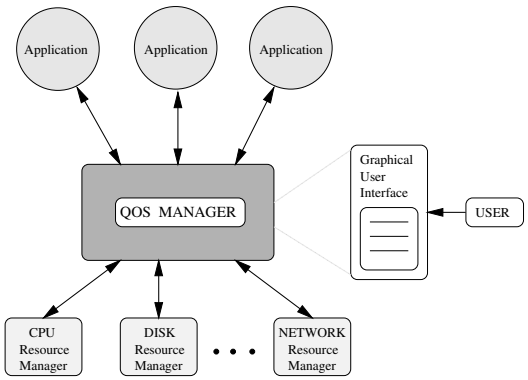


Figure 2. Resource management architecture

QoS Manager (QM) which is a system service responsible for co-ordinating the distribution of shared operating system resources amongst all the domains (Nemesis equivalent of a process) depending on availability and resource allocation policy. In order to get a QoS guarantee for a certain resource or to re-negotiate an old guarantee, an application must make a request to the QM. The QM indicates its allocation decisions to the appropriate Resource Manager (RM) which is the entity that ensures that QoS guarantees are actually enforced via appropriate scheduling.

Adaptive applications can register a number of valid *modes* of operation with the QM at the start of their execution. During situations of resource shortage or unexpected availability (e.g when an application dies and frees up resources), the QM will use this information combined with a knowledge of the user's preferences, expressed via the GUI, to reallocate resources in such a way as to maximize the user's perceived performance or utility. Section 3 describes these mechanisms in more detail via an example.

3. Prototype

The current prototype implementation only fully supports the CPU resource (extensions to support the disk and network resources are being implemented at the timing of writing) and applications only adapt in one direction (i.e via frame rate). However we believe that the system still incorporates enough ideas to show the potential of our work and stimulate discussion. A screen shot from a workstation running our resource management system within the Nemesis operating system is shown in Figure 3.

In the example, there are 4 video display applications each configured to run in one of 3 possible modes 0 to 2 corresponding to 25, 15 and 5 frames per second. The video sources are Fore Systems AVA devices [4] and are connected to a Pentium PC via an ATM newtork. The applications are receiving video via separate ATM VCs (the duplication in pictures was due to a shortage of video cameras at the time the screen shot was taken). The user can request a change in mode by clicking the appropriate mouse button in the bottom control bar of the video window.

The GUI provided by the QM to the user displays the domains currently running in order of *user importance*. A domain's importance can be adjusted by highlighting it and using the `UP/DOWN` buttons. The current CPU allocation for each domain, expressed as a guaranteed slice of CPU time within a given period as well as as a percentage, is also shown. The user can dynamically control the resource allocation policies used for resource revocation and redistribution via the `Edit Policy` pulldown menu.

We assume that the current situation is as shown in Figure 3 with domains RV1 and RV2 in mode 0 (i.e 25fps), RV4 in mode 2 and RV5 in mode 1. The CPU requirements for modes 0 to 2 are 5%, 4% and 2% respectively. These values were obtained by giving the video domain a large CPU allocation and measuring its requirements at different modes (we are currently working on a better method for deriving application requirements). The total CPU allocation is 94% which is the system set limit for the CPU. Hence to support any further requests, the QM would have to consider revoking previously allocated resources.

If the user then requests a mode increase in RV4 up to mode 1 (i.e 15fps), an extra 2% of the CPU must be found to satisfy this request. The resource allocation process then proceeds as follows.

Step 1 Obtain a candidate set, S, of domains for resource revocation. This set would correspond to domains which are less important than RV4. In this case the set is RV2, RV1 and RV5 (see Figure 3).

Step 2 Derive a set of all the possible combinations of mode reductions to the domains in S which would yield the required 2%. In this case there are 4 possibilities as summarized in Table 1. We are assuming that the number of domains involved is small enough to make this calculation feasible in real time. We believe this is a reasonable assumption for a typical workstation environment.

Step 3 Choose one option based on the currently defined user policy. The currently implemented revocation

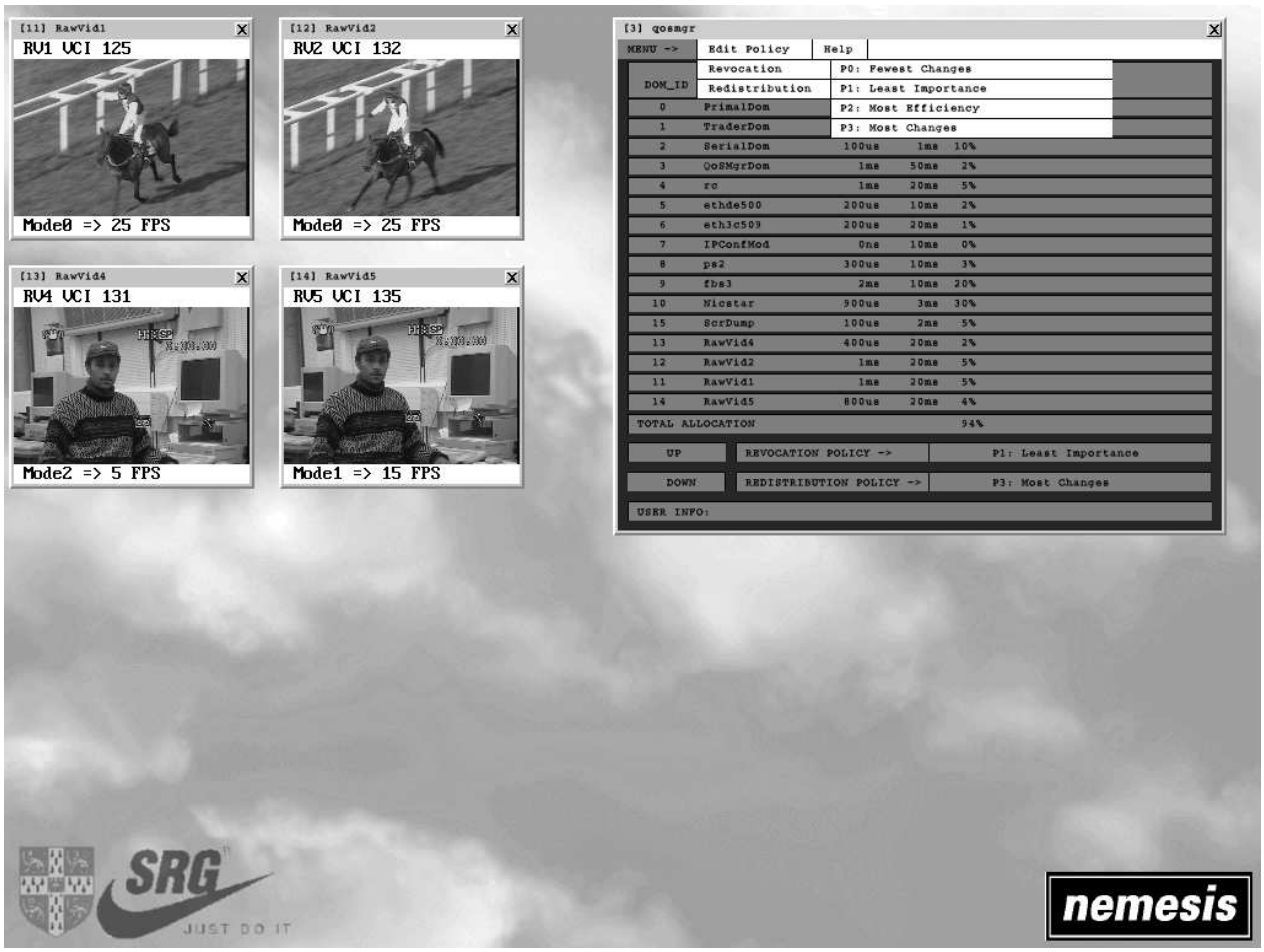


Figure 3. Screen shot of the prototype system

Table 1. Resource revocation options

Option	Domain	Fps change	CPU %
A	RV1	25 > 5fps	3%
B	RV2	25 > 5fps	3%
	RV1	25 > 15fps	1%
	RV2	25 > 15fps	1%
C		Total	2%
D	RV5	15 > 5fps	2%

policies are

- **P0: Fewest Changes**. Pick the option involving changes to as few domains as possible, leading to as little disruption as possible. Options A, B and D would equally qualify in this example and D would then picked because it involves the least important domain.
- **P1: Least Importance**. Pick the option involving the least important domains. This would correspond to option D.
- **P2: Most Efficiency**. Pick the option whose total CPU value is closest to the target value. Options C & D qualify and D would be picked.
- **P3: Most Changes**. Pick the option involving changes to as many domains as possible, leading to a more even distribution of the revocations. Option C would be picked.

The current revocation policy is P1 and hence option D is chosen. A similiar algorithm is used to redistribute resources amongst domains when additional resources become available.

Thus the system gives the user dynamic control over both the relative importance between applications (via the importance list) and the *manner* in which resources are allocated

(via the policy options) instead of constraining adaptation to occur based on some built-in system policy which may not always lead to desirable results as far as the user is concerned.

4. Future work

Future work will address the following

- Extension of the implementation to support multiple resources.
- A generic model for adaptive applications to support adaptation in any number of directions, e.g frame size, compression factor and colour depth.
- A generic way of expressing resource allocation policies. We believe that the simple examples of policy described here leave a lot of scope for further investigation. We intend to develop more precise measures of the user's utility, building on work such as [2] and [3], and we feel that due to the user-oriented nature of the work, having a running prototype will assist greatly with the derivation of realistic solutions.
- The integration of this work into a distributed environment, which may in turn lead to new possibilities.

5. Related work

[8] describes a system similiar to ours where *QoS states* are equivalent to our modes. However the kinds of policies defined are different. The idea of *QoS levels* used in [1] is also similiar to the modes concept. The AQUA system [6] supports adaptation within an application via an adaptation function and [5] identifies some of the problems described here but neither of them address policies for allocating resources between applications.

Acknowledgements

The author would like to thank the members of the SRG group, both past and present, for their support during this work. Special thanks to Dr. Ian Leslie for commenting on an earlier draft of this paper.

References

[1] T. Abdelzaher & K. Shin. *End-host Architecture for QoS-Adaptive Communication.* To appear in IEEE Real-Time Technology and Applications Symposium, June 1998.

[2] S. Chatterjee, J. Sydir, B. Sabata & T. Lawerence. *Modelling Applications for Adaptive QoS-based Resource Management.* The 2nd IEEE High Assurance Systems Engineering Workshop, August 1997.

[3] K. Fukuda, N. Wakamiya, M. Murata and H. Miyahara. *QoS Mapping between User's Preference and Bandwidth Control for Video Transport.* 5th International Workshop on Quality of Service, May 1997.

[4] Fore Systems Research. *AVA Owner's Manual* April 1996.

[5] M. Jones, P. Leach, R. Draves & J. Barrera. *Modular Real-Time Resource Management in the Rialto Operating System.* 5th Workshop on Hot Topics in Operating Systems, 1995.

[6] K. Lakshman, R. Yavatkar & R. Finkel *Integrated CPU and Network-I/O QoS Management in an Endsystem.* 5th International Workshop on Quality of Service, 1997.

[7] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns & E. Hyden. *The design and implementation of an operating system to support distributed multimedia applications.* IEEE Journal on Selected Areas In Communications 14, 7, 1996.

[8] N. Nishio & H. Tokuda. *Simplified Method for Session Coordination Using Multi-level QOS Specification and Translation.* 5th International Workshop on Quality of Service, 1997.

[9] R. Rajkumar, K. Juvva, A. Molano & S. Oikawa. *Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems.* The SPIE/ACM Conference on Multimedia Computing and Networking, 1998.

On Using Intelligent Network Interface Cards to support Multimedia Applications

Marc E. Fiuczynski[†] Richard P. Martin[‡] Tsutomu Owa[*] Brian N. Bershad[†]

[†] Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195

[‡] Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720

[*] Toshiba, Japan. Visiting Researcher at Dept. of Comp. Sci. and Eng., University of Washington, Seattle, WA 98195

Abstract: The emergence of fast, cheap embedded processors presents the opportunity for inexpensive processing to occur on the network interface. We are investigating how a system design incorporating such an *intelligent* network interface can be used to support streaming multimedia applications. We are developing an extensible execution environment, called SPINE, that enables applications to compute directly on the network interface and communicate with other applications executing on the host CPU, peer devices, and remote nodes. Using SPINE, we have implemented a video client that executes on the network interface, and transfers video data arriving from the network directly to the region of frame buffer memory representing the applications window. As a result of this system structure the video data is transferred only once over the I/O bus and places no load on the host CPU to display video at aggregate rates exceeding 80 Mbps.

1 Introduction

Multimedia applications often move large amounts of data between devices (e.g., between network and screen, disk and network, or disk and screen), and therefore place high I/O demands on both the host operating system and the underlying I/O subsystem. We observe that the exponential growth of processor speed relative to the rest of the I/O system presents the opportunity for application-specific processing to occur directly on intelligent I/O devices. Several network interface cards, such as the Myrinet, Alteon's Gigabit Ethernet ACEnic, and I_2O systems, provide the infrastructure to compute on the device itself. With the technology trend of cheap, fast embedded processors (e.g., StrongARM, PowerPC, MIPS) used by intelligent network interface cards, the challenge is not so much in the hardware design as in a redesign of the software architecture needed to match the capabilities of the raw hardware. We intend to apply technology from the SPIN extensible operating system (use of a safe language, extensible interfaces, and resource management) to the network interface, to allow applications to customize their interaction with the network. The motivation is to move application-specific functionality directly to the network interface, thereby reducing I/O related data and control transfers to the host system.

Multimedia applications we believe will benefit from this architecture range from high-definition video clients, multimedia editors, Video-on-Demand servers (e.g., [6]), and 3D distributed real-time rendering (e.g., [7]). Other applications that may benefit, though not directly related to multimedia, range from packet filtering (e.g., Lazy Receive Processing [2]), cluster based storage management (e.g., Petal [3]), to distributed memory management systems (e.g., Global Memory [4]).

Our strategy is to provide an extensible runtime environment, called SPINE [9], for programmable network interface cards. Extensibility is important, as we cannot predict the types of applications that may want to process directly on the network interface. SPINE extends the fundamental ideas in SPIN [1] -- type safe code (called extensions) downloaded into a trusted execution environment -- to the network interface. Specifically, SPINE has three properties that are key to the construction of application-specific solutions:

- **Runtime adaptation and Extensibility.** An application, regardless of its privilege level, may define a SPINE extension and dynamically load it onto an intelligent I/O card.
- **Performance.** SPINE extensions run in the same address space as the firmware, with low-overhead access to services and hardware, and don't require the host CPU to be interrupted. Extensions may share data using peer-to-peer DMA and communicate via message queues. Finally, overall system performance can be improved by reducing I/O related control and data transfers.
- **Safety and Fault Isolation.** The use of a SPINE extension does not compromise the safety of other extensions, the firmware, or the host operating system. Extensions are isolated from the system and one another through the use of a type-safe language, defensive interface design, and resource management.

The rest of this paper describes the software architecture of SPINE and the video application we have built using it.

2 SPINE Software Architecture

SPINE is a safe execution environment that enables applications, which require tight integration with the network, to specialize the network interface for its performance and functionality requirements. We do not assume that the processor architecture of programmable network interfaces provides support for multiple address spaces. A safe language enables extensions and SPINE to run in the context of a single address space. Our design provides safe interfaces to the underlying hardware and allows extensions to provide their own extensible services. In the next two subsections we describe the SPINE runtime and the SPINE communication layer in more detail.

2.1 SPINE Runtime

The SPINE runtime consists of a small Modula-3 runtime that provides support for threads, synchronization, and memory management. It is split across the network adapter and the host into I/O and kernel runtime components, respectively. The basic system structure is shown in Figure 1. Applications define SPINE I/O extensions in Modula-3, which is a Pascal-like type safe programming language. A

SPINE extension may be dynamically loaded onto the network interface card (see the arrow labeled 1 in Figure 1.) where they link with the SPINE I/O runtime. We envision that for some applications it will be necessary to tightly couple both kernel and I/O extensions, and our goal is to support this scenario using a similar extension mechanism. Thus, optionally, applications may define SPINE kernel extension, also in Modula-3, that are loaded into the kernel's virtual address space (see arrow labeled 2 in Figure 1.) where they link with the SPINE kernel runtime.

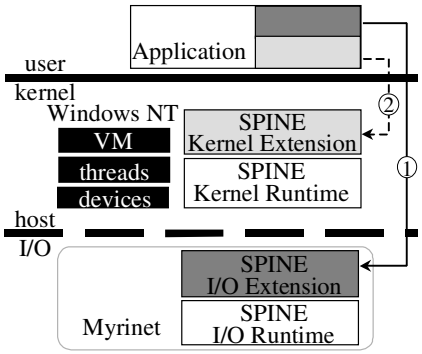


Figure 1. SPINE System Structure.

The SPINE *kernel runtime* provides host-side services to the SPINE *I/O runtime*, such as initializing the intelligent NIC and loading extensions onto the card. Additionally, it provides access to the operating systems thread, device and virtual memory subsystems for SPINE kernel extensions.

The *SPINE I/O runtim*e, which is also implemented in Modula-3, is the execution environment that enables application to compute on the NIC. It exports internal and external interfaces. Internal interfaces are used by extensions that are loaded onto the network interface, which consists of the standard Modula-3 interface, "plain old kernel services", and safe access to the underlying hardware (such as access to DMA engines). The external interface consists of a message FIFO that enables user-level applications, peer devices, and kernel modules to communicate with extensions on the network interface using an active message communication layer. The SPINE I/O runtime manages resources on the network interface. For example, the buffers used as the source/sink of DMA operations are one such resource. These buffers are accessed using unforgable references (i.e., an extension cannot create an arbitrary memory address as a DMA source or destination address), which may refer to local-, peer device-, or host-memory.

2.2 SPINE Communication Layer

The SPINE communication layer is a variation of Active Messages [5] used by the NOW project at UC-Berkeley. It differs from other active message layer implementations in that it can execute shared-memory operations in the context of the network interface. This message layer has the flexibility to execute extension code either on the network interface or the host CPU. The SPINE I/O runtime implements an active message dispatcher that determines whether the message should be pushed to the host or to invoke the handler of a local SPINE extension. SPINE extensions register active message handlers with the I/O runtime, which are invoked when a message arrives from the network, the host, or possibly an intelligent peer device. All handlers are invoked with exactly two arguments: a context variable that contains information associated with the handler at installation time; and, a pointer to the message that contains the data as well as control information specifying the source and type of the message. There are two types of messages: small messages and bulk messages. Small messages are currently 64 bytes total. Bulk messages are similar to small messages, but contain a reference to a data buffer managed by the SPINE I/O runtime. Finally, a handler associated with the active message is invoked after all of the data for the message has arrived.

3 Video Client Application

We use Windows NT 4.0 running on a Pentium based PC as the host system. As the intelligent network interface we use Myricom's Myrinet card on the PCI bus, containing 256 KB SRAM card memory, a 33 MHz "LANai" processor, with a network wire rate of 160MB/s. The LANai processor is a general-purpose processor that can be programmed with specialized control programs and plays a key role in allowing us to experiment with moving application-specific functionality onto the network interface. We have developed our own firmware for the Myrinet upon which we layer the SPINE I/O runtime.

Using SPINE we have implemented a video client application that defines an application-specific *video extension*, which transfers video data arriving from the network directly to the frame buffer. The video client runs as a regular application on Windows NT. It is responsible for creating the framing window that will be used to display the video and informing the video extension of the window coordinates. The video extension on the network interface maintains window coordinate and size information, and DMA transfers video data arriving from the network to the region of frame buffer memory representing the application's window. The video client application catches window movement events and informs the video extension of the new window coordinates.

The implementation of the video extension running on the network interface is simple. It is roughly 250 lines of code, which consists of functions to a) instantiate per-window metadata for window coordinates, size, etc., b) update the metadata after a window event occurs (e.g., window movement), and c) DMA transfer data to the frame buffer. These functions are registered as active message handlers with the SPINE I/O runtime, and are invoked when a message arrives either from the host or the network.

Figure 2 depicts the overall structure in more detail. The numbered arrows have the following meaning:

1. NetVideo application loads video extension onto card.
2. Packet containing video data arrives from the network.
3. SPINE I/O runtime dispatches packet to video extension.
4. Video extension transfers data directly to the frame buffer, and the video image appears on the user's screen.

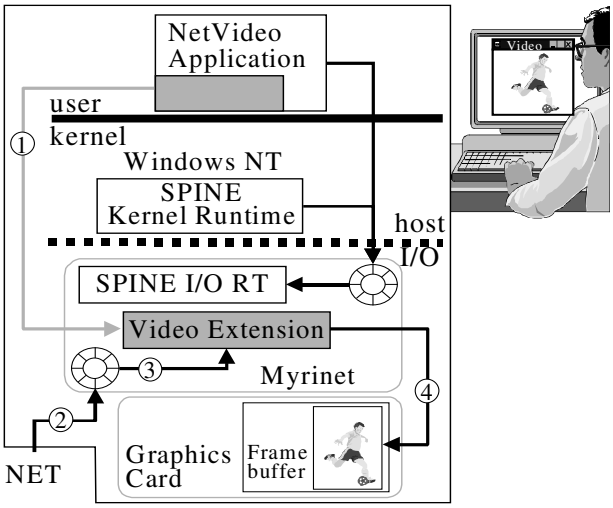


Figure 2 Network video. The Video Extension transfers data from the network directly to the frame buffer.

Using this system structure the host processor is not used during the common case operation of displaying video to the screen. In our prototype system we've been able to support several video clients, each at sustained data rates of up to 40 Mbps, with a host CPU utilization of zero percent for the user-level video application. Thus, regardless of the operating systems I/O services and APIs, we can achieve high-performance video delivery.

The caveat in our current video client experiment is that, although the Myrinet's DMA engines can move large quantities of data, the LANai processor is to slow to decode video data on the fly. The LANai is roughly equivalent to a SPARC-1 processor, i.e., it represents roughly 1989 processor technology. Consequently, our video server takes on the brunt of the work and converts MPEG to raw bitmaps, which it then sends to the video client. Thus the video extension essentially acts as an application-specific data pump; taking data from the network and directly transfering it to the right location of the frame buffer. We expect fast, embedded processors to be built into future NICs that will enable on-the-fly video decoding, or one could use a graphics card that supports video decoding in hardware to avoid decoding on the NIC.

4 Summary

Commercial operating systems that we know, love, and use on a daily basis, may be slow to adopt new techniques to support digital audio/video. Intelligent NICs make it possible to implement application-specific functionality below the operating system, thereby enabling new techniques to be incorporated into commercial systems. A hardware design using a "front-end" I/O processor is not new, but traditionally has been relegated to special purpose machines (e.g., Auspex NFS server), mainframes (e.g., IBM 390 with channel controllers), or supercomputer designs (e.g., Cray Y-MP). The challenge is not so much in the hardware design as in a redesign of the software architecture needed to match the capabilities of the raw hardware. We are designing a flexible I/O execution environment, called SPINE, to enable applications to execute on the network interface. The video client is one example that leverages the SPINE software architecture to efficiently move data through the system. Other application-specific techniques, which may not be easily adopted by commercial operating systems (if at all), could be implemented on the NIC as well. For example, the video client could be improved with Jeffay's [10] buffer management techniques to reduce jitter in a congested network.

Intelligent network cards are particularly compelling when the processor used on an intelligent NIC tracks the relative performance of the host processor. We conjecture that link bandwidth of NICs used in conventional workstations will not exceed one gigabit over the next five years, while the performance of processors continues on its current growth path. Therefore, we expect that future intelligent NICs will be fast enough for a variety of application-specific extensions. For example, Alteon's ACEnic gigabit Ethernet interface has two 100MHz R4000 processors to allow for "… the opportunity to implement more intelligent, value-added features…" [8].

Finally, there are various additional issues to be considered when using this type of hardware design and software architecture. Here are some of the general questions that we intend to examine in the long term with our research:

- When does it pay off to move system functionality to the NIC? Is there enough capacity on the NIC to support interesting [multimedia] applications?
- How does one split applications across the host/IO boundary? Can the split be hidden by "middleware"? How do user-level applications and extensions synchronize?
- What level of OS support is required on the NIC? What types of real-time or QoS support do extensions require?

References

[1] B.N. Bershad, S. Savage, P. Pardyak, E.G. Sirer, M.E. Fiuczynski, D. Becker, S. Eggers, C. Chambers. Extensibility, Safety and Performance in the SPIN Operating System. Proc. SOSP-15, Dec. 1995.

[2] P. Druschel and B.Gaurav. *Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems*. Proc. OSDI'96, Oct. 1996.

[3] E.K. Lee and C.A. Thekkath. *Petal: Distributed Virtual Disks*. Proc. ASPLOS-7, Oct. 1996.

[4] M.J. Feeley, W.E. Morgan, F. Pighin, A. Karlin, and H.M. Levy. *Implementing Global Memory Management in a Workstation Cluster*. Proc. SOSP-15. Dec. 1995.

[5] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. *Active Messages: A Mechanism for Integrated Communication and Computation*. Proc. ISCA-19, May 1992.

[6] W.J. Bolosky, J.S. Barrera III, R.P. Draves, R.P. Fitzgerald, G.A. Gibson, M.B. Jones, S.P. Levi, N.P. Myhrvold, R.F. Rashid. *The Tiger Video Fileserver*. Proc. NOSSDAV'96, Apr. 1996.

[7] Thu D. Nguyen and John Zahorjan. *Scheduling Policies to Support Distributed 3D Multimedia Applications*. To Appear Proc. SIGMETRICS '98, Jun. 1998.

[8] Alteon Networks. Gigabit Ethernet Technology Brief. Available from: http://www.alteon.com/techbr.html

[9] M.E. Fiuczynski. SPINE: A Safe Programmable and Integrated Network Environment. Overview Information available from: http://www.cs.washington.edu/homes/mef/research/spine

[10] K. Jeffay, D.L. Stone, F.D. Smith. Transport and Display Mechanisms for Multimedia Conferencing Across Packet-Switched Networks. Computer Networks and ISDN Systems. Jul. 1994.

Supporting Continuous Media in the Piglet OS

Steve Muir, Jonathan Smith[1]
Distributed Systems Laboratory, University of Pennsylvania
{sjmuir,jms}@dsl.cis.upenn.edu

Abstract

Specializing operating system (OS) functions on a per-processor basis can provide efficient support for continuous media, providing better management of resources as well as a model with which parallelism can be applied to data streams. The Piglet OS uses this model.

We describe how Piglet can provide many of the benefits of other OS proposals such as vertically-structured operating systems without the need to change the system model seen by end-users. Our implementation of Piglet provides guaranteed network bandwidth using the Virtual Clock algorithm, and supports parallel (pipelined) MPEG decoding using a virtual device interface.

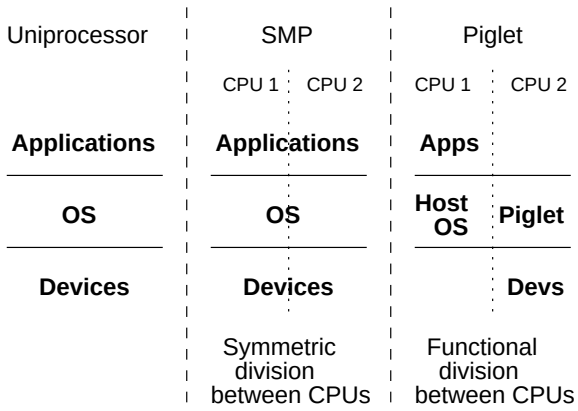


Figure 1: Asymmetry in Piglet

1 Introduction

Efficiently supporting continuous media places new requirements upon networks and operating systems. Firstly, resources must be managed so that the demands of multiple independent applications can be accommodated. Secondly, the complex encodings of many forms of continuous media require extensive computational power to decode them in a timely manner.

Unfortunately, traditional OS structures cannot meet these requirements due to the assumptions inherent in their design. Their abstractions are too general, designed to present application writers with high-level interfaces to devices e.g. Berkeley sockets. Unfortunately this abstraction makes it difficult for applications to make best use of the underlying hardware.

Traditional operating systems also make it a considerable challenge to the programmer to obtain maximum benefit from parallelism available in the system hardware. For example, writing an MPEG player which makes efficient use of dual processors in an SMP configuration is decidedly non-trivial.

One model which has been proposed and appears to offer solutions to these problems is the vertically structured operating system e.g. the University of Cambridge's Nemesis project [1], and MIT's Exokernel [2].

These operating systems provide only very low-level services in the OS kernel (typically safe multiplexing of hardware between applications) implementing all other functionality e.g. scheduling, in user-space, possibly by shared libraries. This allows applications to easily tailor resource management policies to their own needs.

While the vertical structure promises to solve many of the problems described above, it requires application designers to become familiar with a whole new system model. We believe that many of the benefits of vertical operating systems can be provided to applications without imposing a new system model. We have designed and implemented an asymmetric multiprocessor OS named Piglet to investigate these ideas.

2 The Piglet architecture

Piglet uses multiple processors, possibly of different architectures and capabilities (e.g. per-NIC processors and the central processor(s) of a switch or router), in an asymmetric manner as shown in Figure 1.

The system CPUs are partitioned into *functional*

[1] This research was supported by DARPA under Contracts #N66001-96-C-852, #MDA972-95-1-0013, and #DABT63-95-C-0073. Additional support was provided by the AT&T Foundation, and the Hewlett-Packard and Intel Corporations.

groups, each consisting of one or more processors. The host processor group runs the host OS e.g. Linux, and all user applications. Additional functional groups are then dedicated to the management of a certain class of devices, such as network interfaces. The host OS and applications (which have direct user-space access to devices in the Piglet architecture) then access those devices through one or more *Virtual Device Interfaces, VDIs*.

Each processor within a functional group runs the *Lightweight Device Kernel, LDK*, a kernel-mode process which manages the associated devices and implementations of those functions of the VDIs which are not directly available in hardware.

For example, we may provide a VDI which performs TCP/IP checksumming of network data for the user application. If the NIC which data will be sent on provides this functionality in hardware then we can make use of that functionality. If, however, this function is not available, then the LDK can calculate the checksum, thus maintaining a uniform VDI. This means that new devices can be rapidly brought into the Piglet architecture without changing the system interface.

The system can be dynamically switched between different divisions of the CPUs — if computational power is needed then all the CPUs can be placed into the host processor group, giving a traditional SMP structure. Only when it becomes advantageous to do so are CPUs dedicated to a functional group.

3 Resource management using Virtual Clock

While many network-oriented applications do not have specific resource requirements e.g. FTP, e-mail, browsing the World Wide Web, those applications which process continuous media, such as digital audio and video, need to be guaranteed certain resources if they are to provide the end-user with an acceptable rendering of their data.

A bursty application, such as a Web browser, can wreak havoc on the transmission of digital video over the network if the video application is not guaranteed the bandwidth it requires to sustain an acceptable quality display.

Traditional operating systems cannot easily provide the kind of resource guarantee required by video transmission because their multiplexing and demultiplexing of network data is done at too high a level. For example, the TCP/IP stack usually multiplexes all the TCP connections associated with a single interface onto one IP stack.

Vertical operating systems solve this problem by multiplexing[2] network streams onto the network interface at the very lowest level, each application having its own TCP/IP stack.

Piglet provides the same low-level multiplexing as vertical operating systems, each application being allocated its own network queue within the VDI. If applications access these queues directly then the structure is very similar to a vertical OS. Even if applications still access queues through the kernel, thus using only a single IP stack, we can still provide many of the benefits by scheduling the movement of packets to and from this single stack.

Piglet uses Virtual Clock[3] to schedule the delivery of packets from queues to the network interface and vice-versa. It allows an application to specify its required throughput and traffic burstiness in terms of the amount of data to be sent per time period. In addition, priority is given to those applications which have specified resource requirements over those which have not. Queues associated with the latter class of application are only serviced if none of the guaranteed-bandwidth queues have pending data.

Our experiments, described in more detail in [4], show that this mechanism can provide applications with guaranteed network bandwidth[3] in the face of applications which try to use as much bandwidth as possible.

4 Pipelined Protocol Stacks

While the level at which multiplexing is performed in a protocol stack affects the ability of the OS to provide resource guarantees, the protocol stack structure affects its throughput. Traditional operating systems have highly serial implementations of protocol stacks corresponding to their layered architectures, as shown in Figure 2.

While layering provides benefits in terms of data abstraction, this implementation scheme fails to take advantage of the concurrency available in the protocol stack (it can be viewed as a type of dataflow). Each operation is typically independent of every other, and a packet can also be handled without reference to any other. These properties mean that many functions of a protocol stack may operate in an overlapped or pipelined fashion, as shown in Figure 3.

[2] From the purposes of this discussion the same arguments apply to demultiplexing.

[3] On the sending host at least; local mechanisms do not

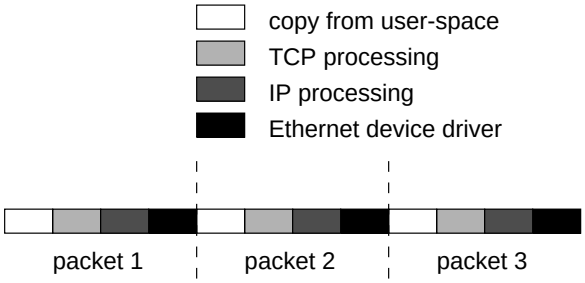


Figure 2: Serial TCP/IP stack implementation

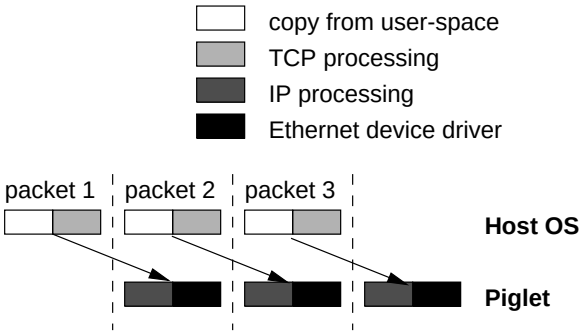


Figure 3: Pipelined TCP/IP stack implementation

Although such a protocol stack could be implemented in a traditional SMP operating system, working within the context of existing protection boundaries and thread/process scheduling would be challenges; the functionally-divided structure of Piglet allows us to more easily move part of the protocol stack into the LDK. The Virtual Device Interface uses shared-memory queues to move data between the application (or host OS) and Piglet, thus providing a clean abstraction and concurrency control between the two 'functional units' (by analogy with a pipelined CPU).

It is also relatively straightforward to split the protocol stack at different boundaries and provide multiple VDIs to the LDK, allowing the application to select the most appropriate functional split. However, splitting the protocol stack in the manner shown in Figure 3 is attractive in that IP processing and lower levels need never examine the packet payload, thus eliminating the need to move that payload between the two processors' D-caches.

5 Piglet Services

We can further extend this model of pipelined protocol stacks by adding application-specific *services* to the protocol stack. If we regard the application's data-processing functions as extra layers in the protocol stack then the Piglet pipelined structure can be extended into the application; Piglet *services* provide various functions to applications in this manner.

5.1 Service Structure

Each network flow within Piglet may be associated with a service which performs processing of packets sent and received on that flow. A Piglet service is implemented in three parts: a transmit function, a receive function, and one or more threads:

- The transmit function takes packets from the flow's queue and performs some service-specific processing on those packets before passing them to the appropriate network interface.
- The receive function performs the reverse processing, except that because it runs in the receive interrupt handler it must be fast. Hence, the receive function passes the packet to the service's threads which perform the necessary processing.

Since services are currently statically compiled into the Piglet LDK, we do not believe that it is necessary to enforce protection barriers between the threads and LDK. Threads execute in the same address-space and at the same privilege level as the LDK, and are non-preemptible (although they can be interrupted). This makes context switching between threads very fast (only 10 instructions on the current x86 implementation) and simplifies concurrency control.

5.2 The Piglet MPEG Decoder

As an example of a non-trivial service which extends the pipelined protocol stack structure to an application we have implemented an MPEG-1 decoder as a Piglet service. The decoder is based upon the Berkeley *mpeg_play* application [5] which displays MPEG-1 streams in an X window. We modified mpeg_play to transmit MPEG frames over the network using a simple UDP-based protocol.

The MPEG-1 decoding process (as implemented by mpeg_play) can be conveniently split into two phases: decoding of MPEG-1 frames into a (Y, U, V) representation of an image, and conversion from (Y, U, V) format to (R, G, B) for display (this phase may also include dithering of the image to reduce color depth).

help deal with 'unruly' hosts.

Decoder	Thorax (fps)	Links (fps)
Application (mpeg_play)	17.5	22.5
Piglet	19.2	25.0

Table 1: Reported frame-rates for MPEG decoding

Since both phases are computationally intensive, there is obvious benefit to be obtained by performing them in parallel. Additionally, the structure of the two phases makes them ideally suited to pipelining. Hence, the Piglet MPEG-decoding service performs the decoding of MPEG-1 frames into the (Y, U, V) representation, leaving the application performs conversion to (R, G, B) and dithering.

In addition to increasing concurrency, this permits the substitution of different video coding protocols without requiring that the application be changed, and also allows Piglet to control the rate at which frames are made available to the application (either for synchronisation/clocking purposes or for resource management).

If each image (typically about 112kB for 320×240 image) was passed to the application as a sequence of data packets then the overhead of processing these packets, including copying of data, would far outweigh the benefits obtained from pipelining. Instead, Piglet's memory buffers into which images are decoded are directly accessible by the application, eliminating most of these overheads.

5.3 Decoder Performance

To illustrate the feasibility of this approach, the Piglet decoder was tested on two video streams: *thorax* [6] and *links* [7]. The former consists of 1356 frames totalling 5.05MB, while the latter is 80 frames/155kB. The average frame-rate reported by mpeg_play is shown in Table 1.

The first row of the table gives the frame rates for the standard mpeg_play application modified to transmit the MPEG stream over the network. Piglet plays no part in this situation, the application performing decoding, conversion, dithering and rendering.

The second row shows the frame rate when Piglet performs the MPEG decoding and the application only performs conversion, dithering and rendering. We see that the frame-rate for both streams is about 10% higher. We believe that with an MPEG player designed specifically as a front-end to a Piglet decoder, rather than modified from a standard player, this figure could be further increased.

6 Conclusions

We believe that Piglet presents an attractive architecture for an OS designed to support networked continuous media for the following reasons:

1. Resource guarantees can be made to applications without changing the host OS. Virtual Clock allows us to guarantee network bandwidth to applications, regardless of other applications.
2. A greater degree of concurrency is achievable more simply than in a traditional OS, thus making more efficient use of multiple processors.

Our implementation of an MPEG decoder as a Piglet service demonstrates that this greater degree of concurrency can provide quantitative benefits to applications processing continuous media.

References

[1] Ian Leslie et al.; *"The Design and Implementation of an Operating System to Support Distributed Multimedia Applications"*, IEEE Journal on Selected Areas in Communications, Volume 14, Number 7 (September 1996), pp.1280–1297.

[2] Dawson R. Engler et al., *"Exokernel: An Operating System Architecture for Application-Level Resource Management"*, Proceedings of the 15th ACM Symposium on Operating Systems Principles (December 1995).

[3] Lixia Zhang; *"Virtual clock: A new traffic control algorithm for packet switching networks"*, ACM Transactions on Computer Systems, Volume 9, Number 2 (May 1991), pp.101–124.

[4] Steve Muir and Jonathan Smith; *"Functional divisions in the Piglet multiprocessor operating system"*, to be published at SIGOPS European Workshop, Sintra, Portugal (September 1998).

[5] Lawrence A. Rowe et al., *mpeg_play (MPEG Video Software Decoder)*; Berkeley Multimedia Research Center, 1995 **http://bmrc.berkeley.edu/projects/mpeg/**

[6] Kiernan Holland, *"Rise of the Thorax"*; 1995 **http://www.unm.edu/~kholland/ /khsplitC3.html**.

[7] Author unknown,*links* animation, **http://www.mpeg1.de/movies/links.m1v**

Multimedia on Multiprocessors: Where's the OS When You Really Need It?

Jason Nieh[1,2] and Monica S. Lam[1]
[1]*Computer Systems Laboratory, Stanford University*
[2]*Sun Microsystems Laboratories*

Abstract

Due to the limitations of current operating systems in supporting multimedia applications, much work has been done to provide resource management mechanisms to address this problem. As processor cycles are often the most oversubscribed and critical resource, most of this work has focused on uniprocessor scheduling. However, hardware platforms are moving to multiprocessor systems, and little work has been done to address the problem of supporting multimedia applications in a multiprocessor context. This paper proposes a new multiprocessor scheduler designed to meet the requirements of multimedia applications. We present an overview of the scheduling algorithm, describe its implementation in a commercial operating system, and discuss directions for future work.

1 Introduction

We are at the dawn of an era in which many everyday applications will need high performance computing. Tomorrow's multimedia applications will do much more than just playback pre-recorded audio and video; we expect that such applications will employ sophisticated image processing techniques and integrate complex computer graphics, all delivered with high interactivity and real-time response. To meet these computing demands, hardware platforms are evolving from single processor systems to multiprocessor systems. We already see today a proliferation of commercial multiprocessor solutions, from desktop workstations to high-end servers. By ganging together commodity microprocessors in varying multiprocessor configurations, these machines can leverage continuing improvements in microprocessor technology and offer the promise of scalable performance. The inherent parallelism found in many multimedia applications makes them particularly amenable to multiprocessor solutions.

While hardware technology has advanced, software technology has lagged behind in effectively supporting multimedia. A distinguishing characteristic of multimedia applications is that they often have real-time requirements associated with their execution. Recognizing the limitations of current operating systems in supporting the real-time requirements of multimedia applications, much work has been done to provide resource management mechanisms to address this problem [1, 3, 8, 9, 10, 11, 13, 15, 16, 18]. As processor cycles are often the most oversubscribed and critical resource, most of this work has focused on uniprocessor scheduling. However, hardware platforms are moving to multiprocessor systems and little work has been done to address the software problem of supporting multimedia applications in a multiprocessor context. Commercial operating systems such as UNIX SVR4 [17] and Windows NT [2] attempt to address this problem by providing a real-time static priority scheduler to be used in conjunction with a traditional timesharing scheduler, but experimental results have demonstrated that these approaches can result in pathological behaviors in which runaway real-time activities can take over the system and even prevent it from accepting user input [14].

To support the requirements of multimedia applications in a multiprocessor environment, operating systems must not only make efficient use of multiple processors in meeting the real-time requirements of these applications, but they must allow such real-time activities to co-exist with non-real-time (conventional) activities already found in traditional interactive and batch applications. While some previous work has attempted to address the problem of real-time multiprocessor scheduling [4, 6, 7], little work has been done to address the problem of how to allow both real-time and conventional applications to share resources and co-exist together in a multiprocessor environment. In addition, different users may have different preferences for the behavior of a particular application mix. To allow users to bias the allocation of resources according to those preferences, the operating system must also provide flexible user controls over the allocation of resources across both real-time and conventional applications.

Scheduling multimedia applications on multiprocessors poses challenges that do not arise in scheduling single processor systems. A single dispatch queue from which

tasks are scheduled is sufficient for the uniprocessor case. However, experience with commercial operating systems suggests that scheduling multiple processors with a centralized dispatch queue is a synchronization bottleneck that can limit the scalability of multiprocessor systems [5, 19]. If a dispatch queue should be associated with each processor, how should tasks be assigned to those dispatch queues in the first place? In particular, the operating system must effectively balance the load across multiple processors. As multimedia application workloads often have dynamically varying resource demands, the operating system must decide when to migrate tasks from one processor to another for load balancing. On the other hand, the operating system may want to reduce task migration through some form of cache affinity to reduce cache misses that occur when a task migrates among processors.

This paper proposes a new multiprocessor scheduler designed to meet the requirements of multimedia applications. Some of the key features of our solution are: (1) decouples the assignment of which processor to use to run a given task (processor task assignment) from the scheduling of tasks already assigned to a processor (per processor scheduling), (2) accounts for cache effects in the processor task assignment, (3) explicitly accounts for application time constraints in both the processor task assignment and the per processor scheduling to make efficient use of resources in meeting real-time requirements, (4) provides flexible prioritized and proportional resource sharing across both real-time and conventional activities. We describe the usage model of our system, present an overview of the scheduling algorithm, and discuss its implementation in the Solaris UNIX operating system, Sun's commercial operating system.

2 Usage model

Our scheduler provides explicit time constraints for real-time applications and supports the notions of priority and proportional sharing across real-time and conventional tasks, even when the system is overloaded. Without additional information from the user, the system provides fair resource allocation across all tasks by default. The system can deliver a rich set of behavior with just a little more information from users and applications:

- **Time constraints**. An application can provide time constraints on a computation, and the scheduler will try to schedule earlier deadline computations before computations that can tolerate more delay. A time constraint for a block of application code consists of a deadline by which the code should execute and an estimate of the processing time required for the code.
- **Resource allocation: priority and proportional sharing**. Real-time tasks should not always be allowed to run first because they may starve out important conventional tasks, such as those that keep the system running. The user can prioritize all the tasks as desired, regardless of whether they are real-time or conventional. Each task can be assigned a priority for this purpose. Among tasks of equal priority, the user can also decide that certain applications should share a processor in some given proportion, which is maintained adaptively as the loading condition changes. Each task can be assigned a share for this purpose.
- **Inform applications of resource availability**. In support of adaptive real-time applications, our scheduler allows applications to decide their own degradation policies. The scheduler notifies applications via an upcall when their time constraints cannot be met and informs them of what resources are available. The upcall invokes an application-level handler which can decide its own policy for dealing with the missed time constraint. For instance, a video application can decide whether to skip a video frame or show a lower quality image when the frame cannot be fully displayed in a timely fashion.

In summary, each task can be parameterized based on its time constraint, consisting of a deadline and a processing time estimate, and its priority and share, which determine the overall resource allocation of the task.

3 Scheduling algorithm

The approach that we take in our scheduler design is to reduce the multiprocessor scheduling problem into two decoupled scheduling decisions: (1) selecting the processor to which to assign a set of tasks, and (2) scheduling the set of tasks assigned to a given processor. By decoupling these decisions, each processor can schedule its own set of tasks in an independent manner. This distributed scheduling model avoids excessive synchronization with other processors and provides performance that can scale with multiprocessor machines with larger numbers of processors.

The basic idea behind how our scheduler decides which processor to assign a set of tasks is as follows. Given a real-time task, find the best processor to use to execute the given task such that all real-time tasks assigned to that processor can meet their respective deadlines. As a result, the algorithm finds excess slack in the system and uses it effectively for meeting deadlines. If no such processor exists, assign the task in question to the processor with the least important tasks, where importance is measured based

on the priorities and shares of tasks. As a result, the algorithm finds processors that are executing less important tasks and defers them to ensure that more important tasks can obtain their desired resource allocations.

In selecting a processor to which to assign a real-time task, the scheduling algorithm determines whether the given real-time task can be run on the processor such that all real-time tasks on that processor can meet their deadlines. If more than one processor satisfies this criterion, the algorithm selects the processor whose excess slack is the best fit for the computational requirements of the given real-time task. This packing process allows real-time tasks with less stringent time constraints to share a given processor while ensuring that processors with large amounts of slack remain available for running real-time tasks with more stringent time constraints. In particular, a processor with conventional tasks assigned to it is considered to have zero excess slack. As a result, real-time tasks will tend to be assigned to processors with other real-time tasks and no conventional tasks. Since real-time tasks, such as those that process video, have poor cache behavior, their execution on the same processor does not result in caching problems. At the same time, they also do not cause cache pollution on processors that are executing conventional tasks with good cache behavior.

In selecting a processor to which to assign a task based on importance, the scheduling algorithm uses priority as the primary measure of importance. It calculates the processor priority for each processor based on the highest priority task assigned to the processor, and selects the processor with the lowest processor priority. For processors of the same priority, the scheduling algorithm sums the total number of shares of all tasks at the given priority and selects the processor with the smallest share sum.

Note that in selecting a processor to which to assign a conventional task, the scheduling algorithm does not use the slack of each processor. Instead, it simply uses the importance-based selection. Once a conventional task is assigned to a given processor, the algorithm may migrate the task to another processor for load balancing reasons, but will avoid doing so for a predefined interval of time. This helps keep the working set of the task in the respective cache and reduce cache misses that result from switching processors. Cache affinity considerations are less of an issue for real-time tasks since they exhibit poor cache behavior.

Once tasks have been assigned to a given processor, we use the SMART scheduling algorithm [15] to perform the per processor scheduling because of its demonstrated performance benefits over other uniprocessor approaches. The algorithm examines both real-time and conventional tasks and performs a feasibility test to determine whether or not the deadlines of all real-time tasks can be met while ensuring that conventional tasks receive their desired resource allocations. When not all deadlines can be met, less important tasks are deferred to meet the requirements of more important tasks, where importance is measured based on the priorities and shares of tasks. A detailed discussion of the SMART scheduling algorithm is beyond the scope of this paper, but such a discussion presented in [15].

4 Implementation status and future work

We have implemented our scheduling algorithm in version 2.5.1 of Sun's Solaris UNIX operating system, a commercial UNIX SVR4 multithreaded operating system. The Solaris operating system is already designed to run on a shared-memory multiprocessor system, which aided our implementation effort. In particular, the Solaris scheduling framework provides a separate dispatch queue for each processor and each task is assigned to one dispatch queue from which it can execute. Separate scheduling locks are associated with each dispatch queue. Scheduling operations proceed in a sequence of steps, obtaining and releasing necessary scheduling locks. Such incremental steps allow scheduling operations to proceed in parallel on different processors, providing more scalable performance.

The original UNIX SVR4 scheduler is a two-level scheduler with a set of scheduling class policies and an underlying priority scheduler. The scheduling class policies include a real-time static priority class, a system priority class, and a timesharing class. The various scheduling policies are unified by mapping each of them onto a range of global priorities. The global priorities are then used by the underlying priority scheduler to decide which task to execute.

To implement our scheduling algorithm, we replaced the underlying UNIX SVR4 priority scheduler with our scheduling algorithm. In particular, our decoupling of processor task assignment and per processor scheduling mapped well to the existing Solaris scheduling framework of per processor dispatch queues. We also introduced a new scheduling policy that exposes the features of our scheduling algorithm to applications and users to allow them to take advantage of time constraints, priorities, and shares. One of the benefits of our approach is that our scheduler subsumes basic priority scheduling and is thus completely backwards compatible with standard UNIX SVR4, allowing all scheduling class policies to preserve their existing functionality without any modification.

Some modifications were also made to the UNIX SVR4 scheduling framework to enable the system to more precisely support the timing requirements of multimedia applications. In particular, the standard UNIX SVR4 scheduling framework, upon which the Solaris operating

system is based, employs a periodic 10 ms clock tick. It is at this granularity that scheduling events can occur, which can be quite limiting in supporting real-time computations with audio and video that have time constraints of the same order of magnitude. To allow a much finer resolution for scheduling events, we added a high resolution timeout mechanism to the kernel and reduced the time scale at which timer based interrupts can occur. The exact resolution allowed is hardware dependent, but is typically 1 ms or less.

Our initial experiences in running mixes of multimedia audio and video applications with our prototype scheduler implementation have shown promising qualitative improvements over standard UNIX SVR4 scheduling on Sun multiprocessor configurations of up to six processors. While effective multiprocessor scheduling is crucial to support multimedia applications, the processors are just one set of components in an overall system. Other resources that require effective resource management include I/O bandwidth, memory, and the network and network/host interface. Meeting the demands of future multiprocessor multimedia applications will require coordinated resource management across all critical resources in the system. We believe that the ideas discussed here for multiprocessor scheduling will serve as a basis for future work in addressing the larger problem of managing system-wide resources to support multimedia applications.

References

1. G. Bollella, K. Jeffay, "Support for Real-Time Computing Within General Purpose Operating Systems: Supporting Co-Resident Operating Systems", *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, Chicago, IL, pp. 4-14, May 1995.
2. H. Custer, *Inside Windows NT*, Microsoft Press, Redmond, WA, 1993.
3. A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", *Proceedings of SIGCOMM '89*, pp. 1-12, Sept. 1989.
4. M. Dertouzos, A. Mok, "Multiprocessor On-line Scheduling of Hard-Real-Time Tasks", *IEEE Transactions on Software Engineering*, 15(12), pp. 1497-1506, Dec. 1989.
5. J. R. Eykholt, S. R. Kleiman, S. Barton, R. Faulkner, A. Shivalingiah, M. Smith, D. Stein, J. Voll, M. Weeks, D. Williams, "Beyond Multiprocessing...Multithreading the SunOS Kernel", *Proceedings of the 1992 Summer USENIX Conference*, San Antonio, TX, pp. 11-18, June 1992.
6. H. Forbes, K. Schwan, "Rapid – A Multiprocessor Scheduler for Dynamic Real-Time Applications", Technical Report GIT-CC-94-23, College of Computing, Georgia Institute of Technology, Apr. 1994.
7. M. R. Garey, D. S. Johnson, "Scheduling Tasks with Nonuniform Deadlines on Two Processors", *JACM*, 23(3), pp. 461-467, July 1976.
8. D. B. Golub, "Operating System Support for Coexistence of Real-Time and Conventional Scheduling", Technical Report CMU-CS-94-212, School of Computer Science, Carnegie Mellon University, Nov. 1994.
9. P. Goyal, X. Guo, H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems", *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, Seattle, WA, pp. 107-122, Oct. 1996.
10. M. B. Jones, D. Rosu, M-C. Rosu, "CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities", *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, St. Malo, France, pp. 198-211, Oct. 1997.
11. I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, E. Hyden, "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications", *IEEE JSAC*, 14(7), pp. 1280-1297, Sept. 1996.
12. C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *JACM*, 20(1), pp. 46-61, Jan. 1973.
13. C. W. Mercer, S. Savage, H. Tokuda, "Processor Capacity Reserves: Operating System Support for Multimedia Applications", *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Boston, MA, pp. 90-99, May 1994.
14. J. Nieh, J. G. Hanko, J. D. Northcutt, G. A. Wall, "SVR4 UNIX Scheduler Unacceptable for Multimedia Applications", *Proceedings of the Fourth International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Nov. 1993.
15. J. Nieh, M. S. Lam, "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications", *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, St. Malo, France, pp. 184-197, Oct. 1997.
16. I. Stoica, H. Abdel-Wahab, K. Jeffay, "On the Duality between Resource Reservation and Proportional Share Resource Allocation", *Multimedia Computing and Networking Proceedings, SPIE Proceedings Series*, Vol. 3020, San Jose, CA, pp. 207-214, Feb. 1997.
17. *UNIX System V Release 4 Internals Student Guide*, Vol. I, Unit 2.4.2., AT&T, 1990.
18. C. A. Waldspurger, "Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management", Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Sept. 1995.
19. J. Zolnowsky, "Realtime Dispatch in SunOS: an Update", *SunSoft TechConf '96*, Apr. 1996.

Issues in Using Commodity Operating Systems for Time-Dependent Tasks: Experiences from a Study of Windows NT

Michael B. Jones

Microsoft Research, Microsoft Corporation
One Microsoft Way, Building 9s/1
Redmond, WA 98052, USA

mbj@microsoft.com
http://research.microsoft.com/~mbj/

John Regehr*

Department of Computer Science, Thornton Hall
University of Virginia
Charlottesville, VA 22903-2242, USA

regehr@virginia.edu
http://www.cs.virginia.edu/~jdr8d/

Abstract

This paper presents a snapshot of early results from a study of Windows NT aimed at understanding and improving its limitations when used for time-dependent tasks, such as those that arise for audio and video processing.

Clearly there are time scales for which it can achieve effectively perfect reliability, such as the one-second deadlines present in the Tiger Video Filesystem. Other time scales, such as reliable sub-millisecond scheduling of periodic tasks in user space, are clearly out of reach. Yet, there is an interesting middle ground between these time scales in which deadlines may be met, but will not always be. This study focuses on system and application behaviors in this region with the short-term goals of understanding and improving the real-time responsiveness of applications using Windows NT 5.0 and a longer-term goal of prototyping and recommending possible scheduling and resource management enhancements to future Microsoft systems products.

Finally, while this paper primarily contains examples and results from Windows NT, we believe that the kinds of limitations and artifacts identified may also apply to other commodity systems such as the many UNIX variants. Indeed, this paper is primarily intended to provide a starting point for fruitful discussions along these lines at the workshop and not as a record of completed work.

1. Introduction

Windows NT and other commonly available general-purpose operating systems such as Solaris and Linux are increasingly being used to run time-dependent tasks such as those that arise for audio and video processing, despite good arguments against doing so [Nieh et al. 93]. This is the case even though many such systems, and Windows NT [Solomon 98] in particular, were designed primarily to maximize aggregate throughput and to achieve approximately fair sharing of resources rather than to provide low-latency response to events, predictable time-based scheduling, or explicit resource allocation mechanisms. Nonetheless, since these systems are being used for time-dependent tasks, it is important to understand both their capabilities and limitations for such applications.

We are in the early stages of a study of Windows NT aimed at understanding and improving its limitations when used for time-dependent tasks. Clearly there are time scales for which it can achieve effectively perfect reliability, such as for the one-second deadlines present in the Tiger Video Filesystem [Bolosky et al. 97]. Other time scales, such as reliable sub-millisecond scheduling of periodic tasks, are clearly out of reach without resorting to special tricks. But there is an interesting range in the middle where tasks can often be executed, but not always. Many practical multimedia activities, such as fine-grained real-time audio synthesis, fall into this middle range.

This paper is intended to serve as a catalyst for discussions on the effectiveness of and problems with using commodity operating systems for time-dependent tasks. While it does provide a snapshot of some of the early findings from our study of Windows NT, it is not a record of completed work. Rather, it is intended to provide some concrete starting points for the discussion at the workshop based on real data.

2. Windows NT Background

Windows NT contains few of the systems facilities and design features that are commonly accepted as providing effective underpinnings for time-dependent applications. Features not found include deadline-based scheduling, explicit CPU or resource management [Mercer et al. 94, Nieh & Lam 97, Jones et al. 97], priority inheritance, fine-granularity clock and timer services [Jones et al. 96], and bounded response time for essential system services. Features that it does have include high-priority real-time thread priorities, interrupt routines that typically re-enable interrupts very quickly, and periodic callback routines [Solomon 98].

Windows NT schedules threads based on their priority and processor affinity. The priorities are divided into three ranges: real-time (16-31), normal (1-15), and idle (0). Priorities of threads in the normal range are boosted following I/O completions and decreased when the thread's time quantum runs out, as is often done in time-sharing systems. The system never adjusts the priorities of threads in the real-time range. The scheduler essentially selects the first thread of the highest runnable

* John Regehr is a research intern at Microsoft Research during the summer of 1998.

priority and runs it for its quantum, then places it at the tail of its priority list. For more details on processor affinity and related issues, see [Solomon 98].

Under Windows NT, not all CPU time is controlled by the scheduler. Of course, time spent in interrupt handling is unscheduled, although the system is designed to minimize hardware interrupt latencies by doing as little work as possible at interrupt level. Instead, much driver-related work occurs in *Deferred Procedure Calls* (DPCs). DPCs are routines executed within the kernel in no particular thread context in response to queued requests for their execution. For example, DPCs check the timer queues for expired timers and process the completion of I/O requests. Hardware interrupt latency is reduced by having interrupt handlers queue DPCs to finish the work associated with them. All queued DPCs are executed whenever a thread is selected for execution just prior to starting the selected thread. While good for interrupt latencies, DPCs can be bad for thread scheduling latencies, as they can potentially result in an unbounded amount of work running before a thread is scheduled.

To enhance system portability, Windows NT uses a loadable *Hardware Abstraction Layer* (HAL) module that isolates the kernel and drivers from low-level hardware details such as I/O interfaces, interrupt controllers, and multiprocessor communication mechanisms. The system clock is one service provided by each HAL. The HAL generates periodic clock interrupts for the kernel. The HAL interface contains no means of requesting a single interrupt at a particular time.

The Win32 interface contains a facility called *Multimedia Timers* supporting periodic execution of application code at a frequency specified by the application. The period is specified in 1ms increments. By default, the Windows NT kernel receives a clock interrupt every 10 to 15ms; to permit more accurate timing, multimedia timers internally use a Windows NT system call that allows the timer interrupt frequency to be adjusted within the range permitted by the HAL (typically 1-15ms).

Multimedia timers are implemented by spawning a high-priority thread that sets a kernel timer and then blocks. Upon awakening the thread executes a callback routine provided by the user, schedules its next wakeup, and then goes back to sleep.

3. Baseline Performance Measurements

Given that Multimedia Timers are the primary mechanism available for applications to request timely execution of code, it is important for time-sensitive applications to understand how well it works in practice. We designed a set of experiments to determine this.

We wrote a test application that sets the clock frequency to the smallest period supported by the HAL (~1ms for all HALs used in these tests) and requests callbacks every 1ms. Each callback just records the Pentium cycle counter value at which it occurred in pinned memory. The application runs at the highest real-time priority. Note that the application is blocked waiting for a callback nearly 100% of the time, and so imposes no significant load on the system. The core of the application is as follows:

```
int main(…) {
    timeGetDevCaps(&TimeCap, …);
    timeBeginPeriod(TimeCap.wPeriodMin);
        // Set clock period to min supported
    TimerID = timeSetEvent(
        // Start periodic callback
        1,  // period (in milliseconds)
        0,  // resolution (0 = maximum)
        CallBack,   // callback function
        0,  // no user data
        TIME_PERIODIC);  // periodic timer
}
void Callback(…) {
    TimeStamp [i++] = ReadTimeStamp();
        // Record Pentium cycle counter value
}
```

On an ideal computer system dedicated to this program the callbacks would occur exactly 1ms apart. Actual runs allow us to determine how close real versions of Windows NT running on real hardware come to this.

Performance measurements were made on two different machines:

- a Pentium Pro 200MHz uniprocessor, with both an Intel EtherExpress 16 ISA Ethernet card and a DEC 21140 DC21x4-based PCI Fast Ethernet card, running uniprocessor kernels, using the standard uniprocessor PC HAL, HALX86.
- a Pentium 2 333MHz uniprocessor (but with a dual-processor motherboard) with an Intel EtherExpress Pro PCI Ethernet card, running multiprocessor kernels, using the standard multiprocessor PC HAL, HALMPS.

NT4 measurements were made under Windows NT 4.0, Service Pack 3. NT5 measurements were made under Windows NT 5.0, build 1805 (a developer build between Beta 1 and Beta 2). All measurements were made while attached to the network.

3.1 Supported Clock Rates

The standard uniprocessor HAL advertises support for clock rates in the range 1003µs to 14995µs. The actual rate observed during our tests was equal to the minimum, 1003µs. This was true for both NT4 and NT5.

The standard multiprocessor HAL advertises support for clock rates in the range 1000µs to 15625µs. The actual rate observed during our tests, however, was 976µs – less than the advertised minimum. See Section 4.1 for some of the implications of this fact. Once again, these observations were consistent across NT4 and NT5.

Finally, note that some HALs do not even support variable clock rates. This limits Multimedia Timer resolution to a constant clock rate chosen by the HAL.

3.2 Times Between Timer Callbacks

Table 1 gives statistics for typical 10-second runs of the test application on both test machines for both operating system versions.

Times Between Callbacks	PPro, NT4	PPro, NT5	P2, NT4	P2, NT5
Minimum µs	31	31	20	33
Maximum µs	2384	18114	2144	2396
Average µs	999	999	999	999
Std Dev µs	70	211	955	941

Table 1: Statistics about Times Between Callbacks

All provide an average time between callbacks of 999µs, but the similarities end there. Note, for instance, that the standard deviation for the Pentium 2 runs is around 950µs – nearly equal to the mean! Also, notice that there was at least one instance on the Pentium Pro under NT5 when no callback occurred for over 18ms.

The statistics do not come close to telling the full story. Table 2 is a histogram of the actual times between callbacks for these same runs, quantized into 100µs bins.

# Times Between Callbacks Falling Within Interval	PPro, NT4	PPro, NT5	P2, NT4	P2, NT5
0-100µs	34	62	4880	4880
100-200µs	1			
300-400µs		1		
500-600µs	4	2		
600-700µs	6	1		
700-800µs	22			
800-900µs	150	10		
900-1000µs	571	1281		
1000-1100µs	9014	8627		
1100-1200µs	161	10		
1200-1300µs	28	1		
1300-1400µs	6	1		
1400-1500µs	1	1		2
1700-1800µs			2	5
1800-1900µs			9	91
1900-2000µs			5107	5014
2000-2100µs				4
2100-2200µs			2	2
2300-2400µs	2			2
7700-7800µs		2		
18100-18200µs		1		

Table 2: Histogram of Times Between Callbacks

Now, the reason for the high standard deviation for the Pentium 2 runs is clear – no callbacks occurred with spacings anywhere close to the desired 1ms apart. Instead, about half occurred close to 0ms apart and half occurred about 2ms apart!

Also, for the Pentium Pro NT5 run, note that twice callbacks occurred about 7.7ms apart and once over 18ms apart. In fact, this is not atypical. On this configuration, there are *always* two samples around 7-8ms apart and one around 18ms apart.

Indeed, the point of our study is to try to learn what is causing anomalies such as these, and to fix them!

4. Early Results

This section presents two snapshots of the kinds of problems we discovered (and one of which we already fixed!) during preliminary latency debugging of applications using Multimedia Timers.

4.1 HAL Timing Differences

Because the HAL virtualizes the hardware timer interface, HAL writers may implement timers in different ways. For example, HALX86 uses the 8254 clock chip to generate clock interrupts on IRQ1, but HALMPS uses the Real Time Clock (RTC) to generate interrupts on IRQ8.

Upon receiving a clock interrupt, the HAL calls up to the Windows NT kernel, which (among other things) compares the current time to the expiration time of any pending timers, and dequeues and processes those timers whose expiration times have passed.

As we have seen, multimedia timers are able to meet 1ms deadlines most of the time on machines running HALX86. To understand why 1ms timers do not work on machines running HALMPS, we next examine the timer implementation in more detail.

A periodic multimedia timer always knows the time at which it should next fire; every time it does fire, it increments this value by the timer interval. If the next firing time is ever in the past, the timer repeatedly fires until the time has moved into the future. The next firing time is rounded to the nearest millisecond. This interacts poorly with HALMPS, which approximates 1ms clock interrupts by firing at 1024Hz, or every 976µs. (The RTC only supports power-of-2 frequencies.)

Because the interrupt frequency is slightly higher than the timer frequency, we would expect to occasionally wait almost 2ms for a callback when the 976µs interrupt interval happens to be contained within the 1000µs timer interval. Unfortunately, rounding the firing time ensures that this worst case becomes the common case. Since it never asks to wait less than 1ms, it always waits nearly 2ms before expiring, then fires again immediately to catch up, hence the observed behavior.

We fixed this error by modifying the timer implementation to compute the next firing time more precisely, allowing it to request wakeups less than 1ms in the future. (An alternative fix would have been to use periodic kernel timers, rather than repeatedly setting one-shot timers.) The results of our fix can be seen in Table 3.

As expected, approximately 2.4% of the wakeups occur near 2ms, since clock interrupts arrive 2.4% faster than timers. As a number of HALs besides HALMPS use the RTC, this fix should be generally useful.

# Times Between Callbacks Falling Within Interval	**P2, NT5**	**P2, NT5 fixed**
0-100µs	4880	1
500-600µs		1
600-700µs		3
700-800µs		2
800-900µs		7
900-1000µs		9609
1000-1100µs		127
1100-1200µs		4
1200-1300µs		2
1300-1400µs		2
1400-1500µs	2	2
1700-1800µs	5	
1800-1900µs	91	
1900-2000µs	5014	240
2000-2100µs	4	
2100-2200µs	2	
2300-2400µs	2	

Table 3: Histogram Showing Results of Timer Fix

4.2 Long-Running DPCs

DPCs are not scheduled by the thread scheduler; they take precedence over any thread. Therefore, long-running DPCs are an obstacle to precise scheduling of user code.

At the time of this writing, we can definitely attribute the 18.1ms and 7.7ms delays on the Pentium Pro to DPC activity. We know, for instance, that the 7ms DPC is in dc21x4.sys – the driver for the DEC Fast Ethernet card. And the 18ms are spent within the NDIS subsystem in a DPC that is intended to check for stalled miniport drivers. Both of these DPCs bear further study in order to identify the exact problem and propose a solution.

5. Methodology

Our primary method of discovering and diagnosing timing problems is to produce instrumented versions of applications, the kernel, and relevant drivers that record timing information in physical memory buffers. After runs in which interesting anomalies occur, a combination of perl scripts and human eyeballing are used to condense and correlate the voluminous timing logs to extract the relevant bits of information from them.

Typically, after a successful run and log analysis, the conclusion is that more data is needed to understand the behavior. So additional instrumentation is added, usually to the kernel, thus unfortunately the edit/compile/debug cycle often gets a reboot step added to it. This approach works but we would be open to ways to improve it.

6. Future Work

Improving predictability of the existing Windows NT features used by time-dependent programs is clearly important, but without better scheduling and resource management support, this can only help so much. In addition to continuing to study and improve the real-time performance of the existing features, we also plan to prototype better underpinnings for real-time applications.

7. Conclusions

While the essential structure of Windows NT is capable of providing low-latency response to events, obvious (and often easy to fix!) problems we have seen, such as the poor interaction between multimedia timers and HALMPS and occasional long DPC execution times, keep current versions of Windows NT from guaranteeing timely response to real-time events below thresholds in the tens of milliseconds. Bottom line – the system is clearly not being actively developed or tested for real-time responsiveness. We are working to change that!

While the details of this paper are obviously drawn from Windows NT, we believe that similar problems for time-dependent tasks will also be found in other general-purpose commodity systems for similar reasons. We look forward to discussing this at the workshop.

Acknowledgments

The authors wish to thank Patricia Jones for her editorial assistance in the preparation of this manuscript.

References

[Bolosky et al. 97] William J. Bolosky, Robert P. Fitzgerald, and John R. Douceur. Distributed Schedule Management in the Tiger Video Fileserver. In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, St-Malo, France, pp. 212-223, Oct. 1997.

[Jones et al. 96] Michael B. Jones, Joseph S. Barrera III, Alessandro Forin, Paul J. Leach, Daniela Roşu, Marcel-Cătălin Roşu. An Overview of the Rialto Real-Time Architecture. In *Proc. of the Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, pp. 249-256, Sep. 1996.

[Jones et al. 97] Michael B. Jones, Daniela Roşu, Marcel-Cătălin Roşu, CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities, In *Proc. of the 16th ACM Symposium on Operating System Principles*, St-Malo, France, pp. 198-211, Oct. 1997.

[Mercer et al. 94] Clifford W. Mercer, Stefan Savage, Hideyuki Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. In *Proc. of the IEEE International Conference on Multimedia Computing and Systems,* May 1994.

[Nieh et al. 93] Jason Nieh, James G. Hanko, J. Duane Northcutt, and Gerald Wall. SVR4 UNIX Scheduler Unacceptable for Multimedia Applications. In *Proc. of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*. Lancaster, U.K., Nov. 1993.

[Nieh & Lam 97] Jason Nieh and Monica S. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, St-Malo, France, pp. 184-197, Oct. 1997.

[Solomon 98] David A. Solomon. *Inside Windows NT, Second Edition.* Microsoft Press, 1998.

An Efficient End-Host Protocol Processing Architecture for Real-Time Audio and Video Traffic

Khawar M. Zuberi and Kang G. Shin
Real-Time Computing Laboratory, The University of Michigan
{*zuberi,kgshin*}*@eecs.umich.edu*

Abstract

Information appliances (IAs) are mass-produced, low-cost devices for specialized computation/communication tasks. Examples include webTV, smart cellular phones, and web phones. Since low cost and low power-consumption are important requirements in the design of these devices, IAs use relatively slow/cheap CPUs (usually 20–40 MHz). Real-time audio and video communication over the Internet is an integral part of many IAs which means that despite slow hardware, the communication subsystem within the OS must be able to efficiently handle heavy network traffic. This paper presents a protocol architecture which reduces I-cache miss overheads (benefiting short audio messages) and also enables use of single-copy without any hardware support or restrictions on network APIs (benefiting long video messages). We implemented UDP/IP to evaluate our architecture and measurements show that overheads for short messages are reduced about 20% while overheads for long messages are reduced 15–22%.

1 Introduction

Information appliances (IAs) [1] are single-user devices with Internet connectivity, used for specialized communication and information retrieval purposes. IAs include devices such as webTVs, smart cellular phones with e-mail, PDAs, and web video phones. With annual production volume of IAs expected to reach 48 million units by year 2001 [2], IAs are becoming an important class of computation devices.

Since IAs are mass produced, keeping per-unit costs low is a primary design objective. As a result, IAs use simple, low-cost hardware. For example, the current generation of *personal information managers* (PIMs) use processors running at 16–44 MHz [1]. Since audio/video communication is a primary function performed by IAs, the communication subsystem within the OS must be highly efficient to work well with the low-cost, slow hardware of IAs for both short audio and long video messages. Different overheads come into play depending on whether short or long messages are being processed. *Data-touching* overheads (which include data copying and checksum overheads) tend to dominate when dealing with long messages. For short (audio) messages, copying overheads are not important, but messages are sent once every 10–30ms [3]. With messages arriving with such high frequency, *non-data-touching* overheads (context switching, interrupt handling, I-cache miss overheads, etc.) become an important part of protocol processing.

Studies have shown that receive-side protocol processing is more complicated and has higher overhead than the send-side [4,5] and this is what limits throughput; so, here we focus on improving receive-side overhead. For reducing non-data-touching overheads, we present *layer bypass* which uses application-specific knowledge to safely bypass select layers within the protocol stack, completely avoiding all I-cache misses associated with those layers. Regarding data-touching overheads, we exploit the periodic nature of video applications. We show that the single-copy scheme presented for non-real-time systems in [6] — which requires specialized network adapter hardware to be feasible for non-real-time systems — works well without any hardware support for multimedia applications because of their periodic nature.

For evaluation, we implemented UDP/IP using our protocol architecture within the EMERALDS real-time operating system [7]. EMERALDS is designed for use in small embedded systems such as digital cellular phones. We chose UDP as the protocol to implement since it is commonly used for audio and video applications.

The next section gives an overview of protocol processing overheads. Section 3 presents our schemes for reducing these overheads which are then evaluated in Section 4. The paper concludes with Section 5.

2 Protocol Architecture Issues

Following is an overview of I-cache and data-copying overheads, schemes proposed by other researchers to reduce these overheads, and shortcomings of these schemes.

2.1 Efficient I-Cache Usage

Communication protocols are designed to accommodate varying communication patterns and error conditions. As a result, a large portion of the protocol code is devoted to checking for rarely-occurring errors or special message formats. These checks are usually coded as shown in Figure 1. Most of the time, there are no errors so that the bodies of the `if` statements never execute. However, code is still fetched into the I-cache, causing replacement misses. Moreover, repeated branches can cause CPU pipeline stalls. For relatively slow CPUs such as those used in IAs, this results in significant non-data-touching overhead. Researchers have proposed techniques such as *outlining/cloning* [8] and *incremental specialization* [9] to reduce I-cache misses, but these schemes involve low-level optimization of frequently-executed code in the

```
if (check1) { ... }
if (check2) { ... }
if (check3) { ... }
```

Figure 1: Typical structure of error checks in protocol code.

protocol stack, and this entails considerable effort on part of the programmer. This indicates a need to develop an easier-to-use scheme to reduce I-cache misses.

2.2 Single-Copy Architectures

The single-copy network architecture was proposed by network adapter (NA) designers [6] to reduce data-touching overheads. The idea is to design a NA with enough buffer space so that on transmission, data is copied once from user-space directly to the NA, while on reception, data stays in the NA until the application makes a receive system call and then data is copied directly to user-space.

To work well for general-purpose systems, single-copy schemes require the NA to have "flexible" buffers. The Afterburner NA [6] uses linked lists to manage NA buffers, so it can continue to receive packets as long as some free buffers are available. This is a more complicated and expensive NA design than common NAs such as LANCE which uses circular queues of buffers. For reception, LANCE fills buffers in the receive ring until it reaches a filled buffer at which point it starts dropping packets (even if other buffers are free down the ring). But, as we will show later, for real-time applications, the single-copy architecture is feasible *when combined with a real-time task scheduler*, and it works even with cheap NAs such as LANCE (which is essential to keep costs low in IAs).

3 Protocol Architecture

The next subsection describes the basic structure we chose for our protocol architecture, followed by a description of our protocol processing optimizations.

3.1 Basic Structure

We chose *lazy receiver processing* (LRP) [10] (Figure 2) as our basic architecture. Under LRP, the packet filter [11] tries to forward real-time packets directly to queues associated with the destination thread where packets stay unprocessed until the application makes a receive system call. This is possible for real-time messages since the application threads are usually periodic so that packets are always processed within a known time interval. Non-real-time packets are forwarded to a special network thread which performs protocol processing and keeps the message until the final destination thread makes a receive call. LRP minimizes *priority inversion* [12] and also saves one context switch for real-time messages compared to architectures which always use intermediate network threads for protocol processing, which is why we use LRP.

Next, we describe our optimizations for protocol processing. Note that these optimizations do not depend on LRP and would work equally well with other protocol architectures.

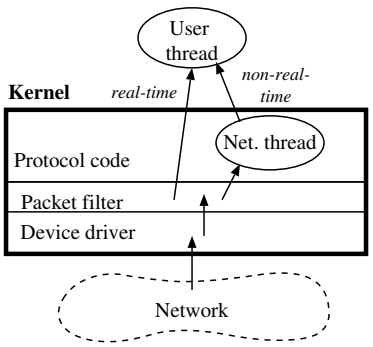


Figure 2: Lazy receiver processing.

3.2 Reducing Non-Data-Touching Overheads

For lowering non-data-touching overheads — especially those related to I-cache misses — we present a new scheme called *layer bypass*. It is easier to apply than low-level optimizations like outlining and specialization and does not require any in-depth analysis of protocol code.

Layer bypass relies on the observation that most of the functionality implemented by various protocol layers is simply not needed when processing short messages. The few operations that are needed are either already duplicated in the packet filter or can be easily migrated there. This allows various protocol layers to be bypassed, completely avoiding all I-cache misses these layers may have caused.

Layer bypass can be applied as follows. When writing the packet filter, the protocol specification for various protocol layers is first studied to determine which aspects of the protocol are not needed for a given message stream (such as live audio messages). Those layers are identified which perform little or no functions. The few useful operations these layers do perform can all be placed together in the packet filter (or some other such small module appropriately inserted in the protocol stack). Then the filter is programmed to detect messages belonging to certain message streams (like audio) and bypass the redundant layers for these messages. For example, if the IP layer is to be bypassed, the filter will forward packets straight to the transport layer.

Layer bypass can be used very effectively for audio messages. UDP/IP is commonly used for such messages, so first consider the IP layer which performs the following major functions:

IP: Routing, fragmentation/reassembly, IP address checks, IP header checksum, checks for malformed headers and packets, IP option processing.

A receiving host does not perform any routing. Short live audio messages need not be fragmented/reassembled. IP options are used for network testing, so they do not apply here. The packet filter checks the destination IP address, so this function of the IP layer is already being handled by the packet filter. The IP header checksum and other error checks can be bypassed safely for the following reason: the packet filter examines the various fields in the header

to detect audio packets. If the header has been corrupted, the filter will not recognize the packet; it will be forwarded to the IP layer, normal processing will occur, the errors will be detected, and the packet will be discarded.

Now we look at the feasibility of bypassing the UDP layer which performs the following major functions:

UDP: UDP datagram checksum, port address checks, generate ICMP messages if destination port does not exist, pass incoming datagrams to correct socket.

The UDP checksum is usually turned off for live audio messages. The packet filter already checks the destination port address. If this check fails, the packet is routed through the full protocol stack where error handling (if needed) can occur. So, for live audio messages, the packet filter bypasses both IP and UDP layers and forwards packets straight to the socket layer.

Layer bypass can also be used for handling other types of short messages such as web server requests as discussed in [13].

3.3 Improving Data-Touching Overheads

We now show that the single-copy scheme — which, without hardware support, has limited value for non-real-time systems — can be used effectively for video communication with no special hardware support. The key to the effectiveness of the single-copy scheme in real-time systems is a real-time scheduler which guarantees that the application executes at its period and does not face unpredictable delays. Hence, incoming packets will stay in the NA buffers for no longer than the period of the application. This is in contrast to non-real-time applications where no such bound exists on how long an application may take to retrieve its packets from the NA.

Real-time audio and video applications run with some period T. T for audio is quite short, usually 10–30ms [3], but video applications can run as slow as 30 or even 10 frames/s, giving a T as large as 0.1s. This is the maximum time messages for a video application have to stay in NA buffers. If NA buffers are about to overflow and the video messages have not been processed, packets for these messages have to be copied out of the NA into kernel buffers to make room for incoming packets. This can occur if a burst of non-real-time packets arrive, filling NA buffers in a short period of time. Following is an analysis of how often this might happen when both real-time and non-real-time packets are being received through the NA.

Estimating Non-Real-Time Packet Arrivals: Poisson processes have previously been used to model packet arrivals. However, studies have shown that wide-area network traffic is too bursty to be correctly modeled by a Poisson process [14]. In fact, only empirical models exist for web browsing [15] and other mass data transfer applications. This precludes any closed-form derivation of non-real-time packet arrival distributions. Instead, we present an engineering approximation of packet arrival rates to see if the single-copy scheme can be used successfully in IAs.

We use web browsing as a representative non-real-time networking application. Measurements of web traffic have shown that retrieval of even small web pages take more than 2 seconds [15]. This is the time needed to look up the remote host's DNS entry and establish TCP connections. After this initial phase data transfer begins at the rate of 1 byte per 90–100μs [15]. Most web pages are relatively small-sized. Considering the small display screens that IAs have, we assume a 10kB page size and 20 packets to carry 10kB. With these assumptions and using 90μs as the per-byte transfer time (which is faster than the wireless link speeds available to most IAs today), we get a packet arrival rate of 22 packets/s or 2.2 packets/0.1s. Even if due to burstiness, five times as many packets arrive within $T = 0.1s$, we still get only 11 packets/T. Even if 5 bursts of 11 packets/T occur during download (highly unlikely since the majority of web pages fit in fewer than 30 packets) and the user spends just one second reading the document, the probability of getting a burst of 11 packets/T is only 5 times in $35T$ seconds or 0.143. This is negligible considering that NAs typically have 128–256 buffers.

Note that the above analysis is true even for the NAs with the simplest buffer-management policies, such as LANCE. As such, the single-copy scheme for real-time applications does not require any special/expensive hardware support which is an important consideration for IAs.

4 Evaluation Results

We want to evaluate the effectiveness of our architecture in handling both short audio and long video messages. We implemented our protocol architecture within EMERALDS on a 25MHz Motorola 68040 processor. The 68040 is typical of CPUs used in many IAs today. (Refer to [13] for evaluation results on a faster processor.) We use two 68040s in our experiments, connected by a 10Mb/s private Ethernet using the LANCE network adapter.

For evaluation, we implemented UDP/IP using our architecture. The protocol and LANCE device driver code was taken from FreeBSD 4.4 and minor modifications were made to make it work with EMERALDS. For simplicity, we used a UDP/IP-specific packet filter. Interested readers are referred to [11] for more generalized high-performance packet filters.

4.1 Performance Improvements

We sent datagram messages from one processor to another and measured the total overhead of receive-side protocol processing including interrupt handling and all relevant context switches using a 5MHz on-chip timer. For each data point (fixed message size) we repeated this experiment 100 times and averaged the results. Further increase in number of samples did not result in any significant change in averaged results.

Short Messages: Figure 3 shows the total receive overhead for short message sizes. This figure presents measurements for the cases when processing is done by a special network thread (labeled "standard architecture"), regular LRP, and LRP when layer bypass is

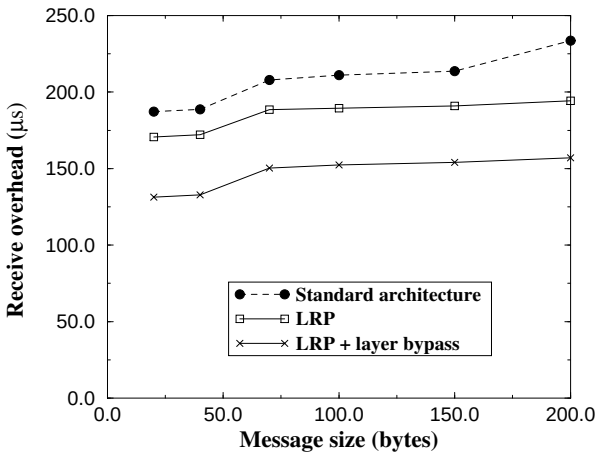


Figure 3: Receive overhead for short messages.

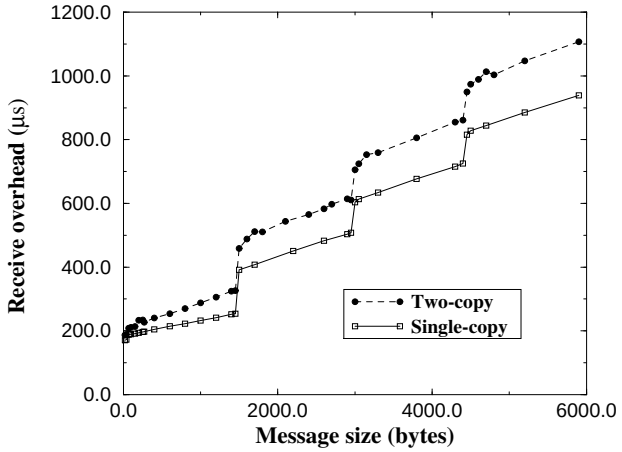


Figure 4: Receive overheads for long messages.

used as well. In all cases, the UDP checksum is turned off. From the figure, we see the benefit of bypassing IP and UDP layers. Performance is improved 20% (beyond that of LRP). Note that the sharper variations in the plots are a result of BSD's mbuf allocation scheme [4] and are not related to the protocol architecture.

Long Messages: Figure 4 plots the receive overhead for messages ranging from 20 to 6000 bytes. It shows that single-copy incurs 15–22% less overhead than the two-copy scheme. Note that the sharp increases in overheads approximately every 1500 bytes is due to messages being fragmented into Ethernet packets, each of which generates a separate interrupt.

5 Conclusion

Information appliances (IAs) are an emerging class of devices which are used for specialized communication tasks such as audio/video communication over the Internet. The Internet-centric nature of IAs combined with the need to keep costs low in these mass-produced devices (which results in the use of slow/cheap processors) dictates that the communication subsystem in the OS be highly efficient to enable audio/video communication despite slow hardware. In this paper we presented a protocol architecture which improves message reception overhead for real-time audio and video communication. Overhead for short live audio messages is reduced 20% by safely bypassing protocol layers. For long video messages, we showed that the single-copy scheme can be used effectively — without hardware support — to reduce overheads by 15–22%.

References

[1] T. Lewis, "Information appliances: Gadget netopia," *IEEE Computer*, vol. 31, no. 1, pp. 59–70, January 1998.

[2] *Appliance war could make web less open.* News Briefs, IEEE Computer, vol. 30, no. 10, pp. 20–25, October 1997.

[3] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," RFC 1890, January 1996.

[4] J. Kay and J. Pasquale, "Measurement, analysis, and improvement of UDP/IP throughput for the DECstation 5000," in *Proc. Winter USENIX*, pp. 249–258, January 1993.

[5] D. Kandlur, D. Saha, and M. Willebeek-LeMair, "Protocol architecture for multimedia applications over ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1349–1359, September 1996.

[6] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, "Afterburner," *IEEE Network*, vol. 7, no. 4, pp. 36–43, July 1993.

[7] K. M. Zuberi and K. G. Shin, "EMERALDS: A microkernel for embedded real-time systems," in *Proc. Real-Time Technology and Applications Symposium*, pp. 241–249, June 1996.

[8] D. Mosberger, L. L. Peterson, P. G. Bridges, and S. O'Malley, "Analysis of techniques to improve protocol processing latency," in *Proc. SIGCOMM*, pp. 73–84, August 1996.

[9] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang, "Optimistic incremental specialization: Streamlining a commercial operating system," in *Proc. Symposium on Operating Systems Principles*, pp. 314–324, December 1995.

[10] P. Druschel and G. Banga, "Lazy receiver processing (LRP): A network subsystem architecture for server systems," in *Proc. Operating Systems Design and Implementation*, October 1996.

[11] D. Engler and M. F. Kaashoek, "DPF: Fast, flexible message demultiplexing using dynamic code generation," in *Proc. SIGCOMM*, pp. 53–59, August 1996.

[12] C. Mercer and H. Tokuda, "An evaluation of priority consistency in protocol architectures," in *Proc. IEEE Conf. Local Computer Networks*, pp. 386–398, October 1991.

[13] K. M. Zuberi, *Real-Time Operating System Services for Networked Embedded Systems*, PhD thesis, University of Michigan, EECS Dept., 1998.

[14] V. Paxson and S. Floyd, "Wide area traffic: The failure of poisson modeling," *IEEE/ACM Trans. Networking*, vol. 3, no. 3, pp. 226–244, June 1995.

[15] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW client-based traces," Technical Report BU-CS-95-010, Boston University, Computer Science Department, 1995.

LIRA: An Approach for Service Differentiation in the Internet *

Ion Stoica, Hui Zhang
Carnegie Mellon University
Pittsburgh, PA 15213
e-mail: {istoica,hzhang}@cs.cmu.edu

Abstract

In this paper, we study the Assured Service model proposed by Clark and Wroclawski [3, 4]. While existing schemes use service profiles that are defined in terms of absolute bandwidth, it is difficult, if not impossible, to design provisioning algorithms that achieve simultaneously good service quality and high resource utilization for such services with large spatial granularities.

We propose an Assured Service model, called LIRA (Location Independent Resource Accounting), in which service profiles are defined in units of resource tokens, rather than absolute bandwidth. The number of resource tokens charged for each in-profile packet is a dynamic function of the path it traverses and the congestion level. Defining service profile in terms of resource tokens allows more dynamic and flexible network control algorithms that can simultaneously achieve high utilization and ensure high probability delivery of in-profile packets. We present an integrated set of algorithms that implement the model. Specifically, we leverage the existing routing infrastructure to distribute the path costs to all edge nodes. Since the path cost reflects the congestion level along the path, we use this cost to design dynamic routing and load balancing algorithms. To avoid packet re-ordering within a flow, we devise a lightweight mechanism that binds a flow to a route so that all packets from the flow will traverse the same route. To reduce route oscillation, we probabilistically bind a flow to one of the multiple routes. Simulation results are presented to demonstrate the effectiveness of the approach.

*This research was sponsored by DARPA under contract numbers N66001-96-C-8528 and N00174-96-K-0002, and by a NSF Career Award under grant number NCR-9624979. Additional support was provided by Intel Corp., MCI, and Sun Microsystems. Views and conclusions contained in this document are those of the authors and should no be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, MCI, Sun, or the U.S. government.

1 Introduction

As the Internet evolves into a global commercial infrastructure, there is a growing need to support more enhanced services than the traditional best-effort service. To address this issue, several new QoS models (guaranteed, controlled load, committed rate) have been proposed [26, 31]. Collectively, they are called Integrated Services or Intserv models. Recently, there are new efforts in the IETF to develop a new class of service models called Differential Services or Diffserv models [2, 3, 4, 17, 22, 29].

While these schemes differ in details, they are very similar at the architectural level. Usually a scheme consists of the following components: (a) a service profile between each customer (user) and the Internet Service Provider (ISP) that defines the commitment of the ISP to the user, (b) ingress nodes at the ISP edge which police the aggregate traffic from each user to make sure that no user exceeds its service profile, (c) network nodes inside the ISP core which implement a variety of packet forwarding, buffer management, and scheduling behaviors in order to control packet queueing delay, loss, and/or throughput, and (d) a set of bits in the header of each packet used to trigger mechanism for differential processing inside the network. Usually, there are two types of bits. The first type specifies the differential processing behavior requested by the user, such as drop or delay preference. These bits are *not* modified by the routers. The second type of bits can be changed by routers and encodes the dynamic information. An example is the bit that encodes whether the packet is in or out of the service profile.

The key difference between Intserv and Diffserv is that while Intserv provides end-to-end QoS service on a per flow basis, Diffserv is intended to provide service differentiation among the *traffic aggregates* to different users *over a long timescale*. Such difference at the service level has important implications on the complexity of the network-level mechanisms required to implement these services. In particular, to provide Intserv, each router needs to support a flow level signaling protocol such as RSVP [32], maintain per flow state, and perform scheduling and manage buffers on a per flow basis. Since there can be a large number of

flows in the Internet, it is an open question whether Intserv can be implemented in a scalable fashion. Diffserv, on the other hand, pushes the complexity to the network edge, and requires very simple priority scheduling/dropping mechanisms inside the core. An important property of the Diffserv schemes considered in this paper is that each router treats identically all packets which have the same bits set. That is, routers only distinguish a small number of aggregated classes of packets, where a class represents all packets with the same marking.

Existing Diffserv schemes are based on the concept of service profile. From the service's point of view, there are three aspects that a Diffserv model needs to specify [3, 4]:

- semantics of the service profile: what exactly is provided to the customer (user)?
- spatial granularity of the service: is the service profile applied to traffic destined to a single destination, a group of destinations, all nodes of an ISP, or everywhere in the Internet?
- level of assurance: how likely is an in-profile packet to be delivered to the destination?

Two examples of differential service models are the Assured Service proposed by Clark and Wroclawski [3, 4] and the Premium Service proposed by Jacobson et. al [22]. The Premium Service provides the equivalent of a dedicated link of fixed bandwidth between two edge nodes. The main advantage of the Premium Service over the current Intserv models such as guaranteed or controlled load is its implementation simplicity – it does not require per flow management at core routers.

The Assured Service supports coarse spatial granularity, i.e., service profiles are applied to traffic defined to more than one destination. It is important to notice that in addition to the implementation simplicity, the Assured Service also provides a service semantic that is richer than those provided by the existing Intserv models. For example, while Intserv and Premium services are better suited for steady and long-lived traffic, the Assured Service provides better support for traffic aggregates that consist of many short-lived bursty flows with different destinations (such as WEB traffic).

We believe that a service model with a coarse spatial granularity embodies some of the fundamental motivations behind the design of the Assured Service. The coarse spatial granularity leads to a smaller number of service profiles, which in turn improves the scalability by reducing the amount of state needed at the edge of the network. In addition, service profiles with coarser spatial granularities also mean higher traffic aggregation for each profile, which allows users to achieve a higher degree of statistical multiplexing gain.

However, as the spatial granularity becomes larger than one destination, it is more difficult to support a service with a fixed bandwidth profile. In this situation, there is a fundamental conflict between maximizing resource utilization and achieving a high service assurance. Since the network does not know in advance where the packets will go, in order to provide high service assurance, it needs to provision enough resources to *all possible* destinations. This will result in a severe resource under-utilization.

In this paper, we show that by defining service profiles in terms of the amount of resource tokens rather than the absolute bandwidth, we can design dynamic and flexible network control algorithms that can achieve high resource utilization, while at the same time delivering in-profile packets with high probability. The key aspect of the model is that the service differentiation is based on resource tokens rather than the exact amount of bits per second. The amount of resource tokens charged for each bit is a dynamic function of the path and the congestion level. This avoids the worst-case provisioning dilemma faced by existing solutions. In addition, as we will discuss in Section 4, our scheme can also be used to implement differential services with service profiles defined in terms of absolute bandwidth.

The rest of the paper is organized as follows. In Section 2 describes our model based on resource tokens and proposes a set of mechanisms to implement the model. Section 3 presents simulation experiments to demonstrate the effectiveness of our solution. Section 4 justifies the new service model and discusses possible ways for our scheme to implement other differential service models. Finally, in Section 5 we present the related work, and in Section 6 we summarize our contributions.

2 *LIRA*: Service Differentiation based on Resource Right Tokens

In this section, we present our differential service model, called *LIRA* (Location Independent Resource Accounting), with service profiles defined in terms of resource tokens rather than absolute amounts of bandwidth.

We consider the following simple two bits encoding scheme. The first bit, called the *preferred* bit, is set by the application or user and indicates the dropping preference of the packet. The second bit, called *marking* bit, is set by the ingress routers of an ISP and indicates whether the packet is in- or out-of profile. More precisely, when a preferred packet arrives at an ingress node, the node marks it if the user has not exceeded its profile; otherwise the packet is left unmarked[1]. The reason to use two bits instead of one is that in an Internet environment with multiple ISPs, even if a packet may be out-of profile in some ISPs on the earlier portion of its path, it may still be in-profile in a subsequent

[1] In the paper, we will use the terminology of *marked* or *unmarked* packets to refer to packets in or out-of the service profile, respectively.

ISP. Having a dropping bit that is unchanged by upstream ISPs on the path will allow downstream ISPs to make the correct decision. Core routers implement a simple behavior of priority dropping. Whenever there is a congestion, a core router always drops unmarked packets first.

In this paper, we focus on mechanisms for implementing LIRA in a single ISP. We assume the following model for the interaction of multiple ISPs: if ISP A is using the service of ISP B, then ISP B will treat ISP A just like a regular user. In particular, the traffic from all ISP A's users will be treated as a single traffic aggregate.

2.1 LIRA Service Model

With LIRA, each user i is assigned a service profile that is characterized by a resource token bucket (r_i, b_i), where r_i represents the resource token rate, and b_i represents the depth of the bucket. Unlike traditional token buckets where each preferred bit entering the network consumes exactly one token, with resource token buckets the number of tokens needed to admit a preferred bit is a dynamic function of the path it traverses.

Although there are many functions that can be used, we consider a simple case in which each link i is associated a cost, denoted $c_i(t)$, which represents the amount of resource tokens charged for sending a marked bit along the link at time t. The cost of sending a marked packet is computed as $\sum_{i \in P} L \times c_i(t)$, where L is the packet length and P is the set of links traversed by the packet. While we focus on unicast communications in this paper, we note that the cost function is also naturally applicable to the case of multicast. As we will show in Section 3, charging a user for every link it uses and using the cost in routing decisions help to increase the network throughput. In fact, it has been shown in [19] that using a similar cost function[2] for performing the shortest path routing gives the best overall results when compared with other dynamic routing algorithms.

It is important to note that the *costs* used in this paper are not monetary in nature. Instead they are reflecting the level of congestion and the resource usage along links/paths. This is different from pricing which represents the amount of payment made by an individual user. Though costs can provide valuable input to pricing policies, in general, there is *no* necessary direct connection between cost and price.

Figure 1 illustrates the algorithm performed by ingress nodes. When a preferred packet arrives at an ingress node, the node computes its cost based on the packet length and the path it traverses. If the user has enough resource tokens in its bucket to cover this cost, the packet is marked, admitted in the network, and the corresponding number of resource tokens is subtracted from the bucket account. Otherwise, depending on the policy, the packet can be either dropped, or treated as best effort. Informally, our goal at the user level is to ensure that users with "similar" communication patterns receive service (in terms of aggregate marked traffic) in proportion to their token rates.

The crux of the problem then is the computation and distribution of the per marked bit cost for each path. In this section, we first present the algorithm to compute the cost of each marked bit for a single *link*, and next present an algorithm that computes and distributes the per-path cost of one marked bit by leveraging existing routing protocols. We then argue that this dynamic cost information is also useful for multi-path routing and load balancing purposes. To avoid route oscillation and packet reordering within one application-level flow, we introduce two techniques. First, a lightweight scheme is devised to ensure that all packets from the same application-level flow always travel the same path. The scheme is lightweight in the sense that no per flow state is needed in any core routers. Second, rather than using a simple greedy algorithm that always selects the path with the current lowest cost, we use a probabilistic scheme to enhance system stability.

2.2 Link Cost Computation

A natural goal in designing the link cost function in *LIRA* is to avoid marked packets being dropped. Since in the worst case all users can compete for the same link at the same time, a sufficient condition is to have a cost function that exceeds the number of tokens in the system when the link utilization approaches unity. Without bounding the number of tokens in the system, this suggests a cost function that goes to infinity when the link utilization approaches unity. Among many possible cost functions that exhibit this property, we choose the following one:

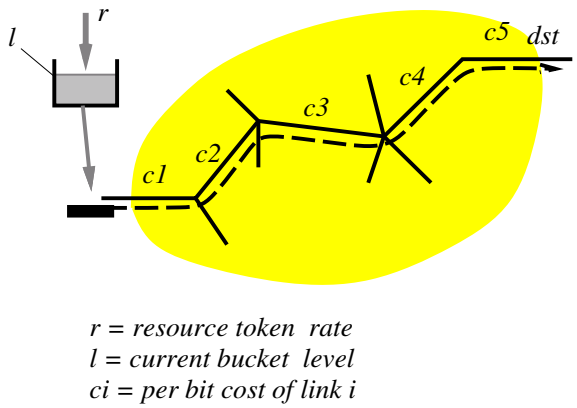
$$c(t) = \frac{a}{1 - u(t)}, \tag{1}$$

where a is the *fixed* cost of using the link[3] when it is idle, and $u(t)$ represents the link utilization at time t. In particular, $u(t) = R(t)/C$, were $R(t)$ is the traffic throughput at time t, and C represents the link capacity. Recall that $c(t)$ is measured in tokens/bit and represents how much a user is charged for sending a marked bit along that link at time t.

In an ideal system, where costs are instantaneously distributed and the rate of the incoming traffic varies slowly, a cost function as defined by Eq. (1) guarantees that no marked packets are dropped inside the core. However, in a real system, computing and distributing the cost information incur overhead, so they are usually done periodically. In addition, there is always the issue of propagation delay. Because of these, the cost information used in admitting packets at ingress nodes may be obsolete. This may cause

[2] It can be shown that when all links have the same capacity our cost is within a constant factor from the cost of shortest-dist(P, 1) algorithm proposed in [19].

[3] In practice, the network administrator can make use of a to encourage/discourage the use of the link. Simply by changing the fixed cost a, a link will cost proportionally more or less at the same utilization.



Ingress Node Alg.

```
upon packet arrival :
  bit_cost = c1 + c2 + c3 + c4 + c5;
  packet_cost = packet_length * bit_cost;
  if (preferred(packet) && l > packet_cost)
     mark(packet);
     l -= packet_cost;
```

Figure 1. When a preferred packet arrives, the node computes the packet's cost, and the packet is marked if there are sufficient resource tokens.

packet dropping, and lead to oscillations. Though oscillations are inherent to any system in which the propagation of the feed-back information is non-zero, the sensitivity of our cost function when the link utilization approaches unity makes things worse. In this regime, an incrementally small traffic change may result in an arbitrary large cost change. In fact one may note that Eq. (1) is similar to the equation describing the delay behavior in queueing systems [18], which is known to lead to system instability when used as a congestion indication in a heavily loaded system.

To address these issues, we use the following iterative formula to compute the link cost:

$$c(t_i) = a + c(t_{i-1})\frac{\widehat{R}(t_i, t_{i-1})}{C}. \qquad (2)$$

where $\widehat{R}(t', t'')$ denotes the average bit rate of the marked traffic during the time interval $[t', t'')$. It is easy to see that if the marked traffic rate is constant and equal to R, the above iteration converges to the cost given by Eq. (1). The main advantage of using Eq. (2) over Eq. (1) is that it is more robust against large variations in the link utilization. In particular, when the link utilization approaches unity the cost increases by at most a every iteration. In addition, unlike Eq. (1), Eq. (2) is well defined even when the link is congested, i.e., $\widehat{R}(t_{i-1}, t_i) = C$.

Unfortunately, computing the cost by using Eq. (2) is not as accurate as by using Eq. (1). The link may become and remain congested for a long time before the cost increases large enough to reduce the arrival rate of marked bits. This may result in the loss of marked packets, which we try to avoid. To address this problem we use only a fraction of the link capacity, $\bar{C} = \beta C$, for the marked traffic, the remaining being used to absorb the unexpected variations due to inaccuracies in the cost estimation[4]. In this paper we chose β between 0.85 and 0.9.

[4] β is similar to the pressure factor used in some ABR congestion control schemes for estimating the fair share [14, 25].

2.3 Path Cost Computation and Distribution

In *LIRA*, the cost of a marked bit over a path is the sum of the costs of a marked bit over each link on the path. Once the cost for each link is computed, it is easy to compute and distribute the path cost by leveraging existing routing protocols. For link state algorithms, the cost of each marked bit can be included as part of the link state. For distance vector algorithms, we can pass and compute the partial path cost in the same way the distance of a partial path is computed with respect to the routing metric.

2.4 Multipath Routing and Load Balancing

Since our algorithm defines a dynamic cost function that reflects the congestion level of each link, it is natural to use this cost function for the purpose of multi-path routing. In this paper, we compute the k shortest paths for each destination or egress node using unit link metric. While the obvious solution is to send packets along the path with the minimum cost (in the sense of *LIRA*, see Section 2.1) among the k paths, this may introduce two problems: (a) packet reordering within one application-level flow, which may negatively affect end-to-end congestion control algorithms, and (b) route oscillation, which may lead to system instability.

We introduce two techniques to address these problems. First, we present a lightweight mechanism that binds a flow to a route so that all packets from the flow will traverse the same route. Second, to reduce route oscillation, for each new flow, an ingress node probabilistically binds it to one of the multiple routes. By carefully selecting the probability, we can achieve both stability and load-balancing.

2.4.1 Forwarding Algorithm For Multiple Path Routing

As discussed earlier, we will maintain multiple routes for each destination. However, we would like to ensure that all

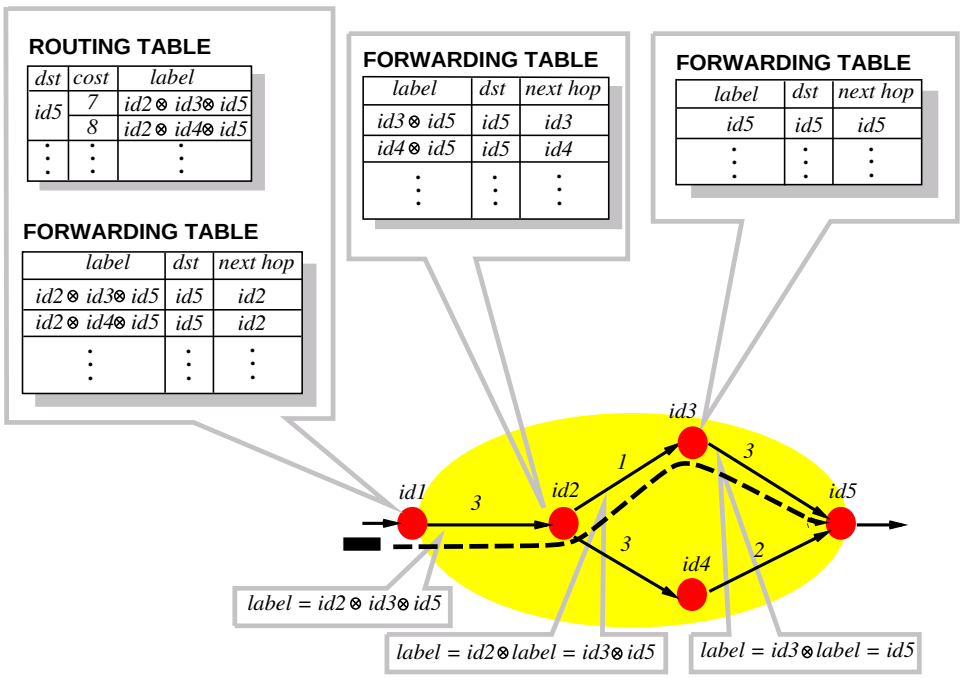


Figure 2. Example of route binding via packet labeling.

packets belonging to the same flow are forwarded along the same path.

The basic idea is to associate with each path a label computed as the XOR over the identifiers of all routers along the path, and then associate this label with each packet of a flow that goes along that path. Here we use the IP address as the identifier. More precisely, a path $P = (id_0, id_1, \ldots, id_n)$, where id_0 is the source and id_n is the destination, is encoded at the source (id_0) by $l_0 = id_1 \otimes id_2 \otimes \ldots \otimes id_n$. Similarly, the path from id_1 to id_n is encoded at id_1 by $l_1 = id_2 \otimes \ldots \otimes id_n$. A packet that travels along path P is labeled with l_0 as it is leaving id_0, and with l_1 as it is leaving d_1. By using XOR we can iteratively re-compute the label based on the packet's current label and the node identifier. As an example, consider a packet that is assigned label l_0 at node id_0. When the packet arrives at node id_1, the new label corresponding to the remaining of the path, $(id_1, \ldots, id_n)$, is computed as follows:

$$l_1 = id_1 \otimes l_0 = \quad (3)$$
$$id_1 \otimes (id_1 \otimes id_2 \otimes \ldots \otimes id_n) = id_2 \otimes \ldots \otimes id_n.$$

It is easy to see that this scheme guarantees that the packet will be forwarded exactly along the path P. Here, we implicitly assume that all alternate paths between two end-nodes have unique labels. Although theoretically there is a non-zero probability that two labels may collide, we believe that for practical purposes it can be neglected.

Next we give some details of how this mechanism can be implemented by simply extending the information maintained by each router in the routing and forwarding tables. Besides the destination and the route cost, each entry in the routing table also contains the label associated with that path.

$$< dst, < cost^{(1)}, l^{(1)} >, \ldots < cost^{(k)}, l^{(k)}) >> \quad (4)$$

Similarly, the forwarding table should contain an entry for each path:

$$< l^{(1)}, dst, next_hop^{(1)} > \ldots < l^{(k)}, dst, next_hop^{(k)} > \quad (5)$$

In Figure 2 we give a simple example to illustrate this mechanism. Assume that nodes id_1 and id_5 are edge nodes, and there are two possible paths from id_1 to id_5 of costs 7, and 8, respectively. Now, assume a packet destined to id_5 arrives at id_1. First the ingress node id_1 searches the classifier table (not shown in the Figure), that maintains a list of all flows, to see whether this is the first packet of a flow. If it is, the router uses the information in the routing table to probabilistically bind the flow to a path to id_5. At the same time it labels the packet with the encoding of the selected route. In our example, assume the path of cost 7, i.e., (id_1, id_2, id_3, id_5), is selected. If the arriving packet is not the first packet of the flow, the router automatically labels the packet with the encoding of the path to which the flow is bound. This can be simply achieved by keeping a copy of the label in the classifier table. Once the packet is labeled, the router checks the forwarding table for the next hop by matching the packet's label and its destination. In our case, this operation gives us id_2 as the next hop. When the packet arrives at node id_2 the router first computes a new label based on the current packet label and the router

identifier: $label = id_2 \otimes label$. The new label is then used to lookup the forwarding table.

It is important to note that the above algorithm assumes per flow state only at ingress nodes. Inside the core, there is no per flow state. Moreover, the labels can speed-up the table lookup if used as hash keys.

2.4.2 Path Selection

While the above forwarding algorithm ensures that all packets belonging to the same flow traverse the same path, there is still the issue of how to select a path for a new flow. The biggest concern with any dynamic routing protocol based on congestion information is its stability. Frequent route changes may lead to oscillations.

To address this problem, we associate a probability with each route and use it in binding a new flow to that route. The goal in computing this probability is to equalize the costs along the alternate routes, if possible. For this we use a greedy algorithm. Every time the route costs are updated we split the set of routes in two equal sets, where all the routes in one set have costs larger than the routes in the second set. If there is an odd number of routes, we leave the median out. Then, we decrease the probability of every route in the first set, the one which contains the higher cost routes, and increase the probability of each route in the second set by a small constant δ. It can be shown that in a steady-state system, this algorithm converges to the optimal solution within δ.

2.5 Algorithm Scalability

As described so far, our scheme requires to maintain k entries for each destination in both the forwarding table used by the forwarding engine and the routing table used by the routing protocol, where k is the maximum number of alternate paths. While this factor may not be significant if k is small, a more serious issue that potentially limits the scalability of the algorithm is that in the vanilla form it requires to maintain an entry for each destination, where in reality, to achieve scalability, routers really maintain the longest-prefix of a group of destinations that share the same route [11]. Since our algorithm works in the context of one ISP, we can maintain an entry for each *egress node* instead of each destination. We believe this is sufficient as the number of egress nodes in an ISP is usually not large.

However, assume that the number of egress nodes in an ISP is very large so that significant address aggregation is needed. Then we need to also perform cost aggregation. To illustrate the problem consider the example in Figure 3. Assume the addresses of d_0 and d_1 are aggregated at an intermediate router r_1. Now the question is how much to charge a packet that enters at the ingress node r_0 and has the destination d_0. Since we do

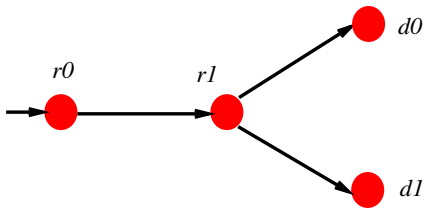


Figure 3. Topology to illustrate the label and cost aggregation.

not keep state for the individual routes to d_0, and d_1 respectively, we need to aggregate the cost to these two destinations. In doing this, a natural goal would be to maintain the total charges the same as in a reference system that keeps per route state. Let $R(r_1, d_i)$ denote the average traffic rate from r_1 to d_i, $i = 1, 2$. Then, in the reference system that maintains per route state, the total charge per time unit for the aggregate traffic from r_1 to d_0 and d_1 is: $cost(r_1, d_0)\,R(r_1, d_0) + cost(r_1, d_1)\,R(r_1, d_1)$. In a system that does not maintain per route state, the charge for the same traffic is $cost(r_1, d_0, d_1)(R(r_1, d_0) + R(r_1, d_1))$, where $cost(r_1, d_0, d_1)$ denotes the per bit aggregate cost. This yields

$$cost(r_1, d_0, d_1) = \frac{cost(r_1, d_0)\,R(r_1, d_0)}{R(r_1, d_0) + R(r_1, d_1)} + \frac{cost(r_1, d_1)\,R(r_1, d_1)}{R(r_1, d_0) + R(r_1, d_1)}. \quad (6)$$

Thus, any packet that arrives at r_0 and has either destination d_0 or d_1 is charged with $cost(r_0, r_1) + cost(r_1, d_0, d_1)$. Obviously, route aggregation increases the inaccuracies in cost estimation. However, this may be alleviated by the fact that the route aggregation usually exhibits high localities.

Another problem with address aggregation is that a label can no longer be used to encode the entire path to the destination. Instead, it is used to encode the *common* portion of the paths to the destinations in the aggregate set. This means that a packet should be relabeled at every router that performs aggregation involving the packet's destination. The most serious concern with this scheme is that it requires to maintain per flow state and perform packet classification at a core router (r_1 in our example). Fortunately, this scalability problem is alleviated by the fact that we need to keep per flow state *only* for the flows whose destination addresses are aggregated at the *current* router. Finally, we note that this problem is not specific to our scheme; any scheme that (i) allows multiple path routing, (ii) performs load balancing, and (iii) avoids packet reordering has to address it.

3 Simulation Experiments

In this section we evaluate our model by simulation. We conduct four experiments: three involving simple topolo-

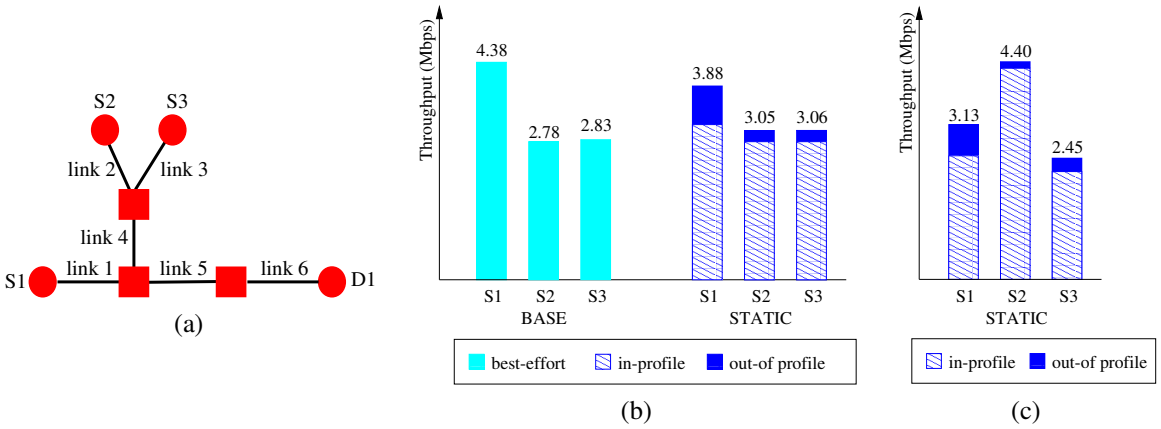


Figure 4. (a) Topology used in the first experiment. Each link has 10 Mbps capacity. $S1$, $S2$, and $S3$ send all their traffic to $D1$. (b) The throughputs of the three users under BASE and STATIC schemes. (c) The throughputs under STATIC when the token rate of $S2$ is twice the rate of $S1$/$S2$.

gies which help to gain a better understanding of the behavior of our algorithms, and one more realistic example with a larger topology and more complex traffic patterns. The first experiment shows that if all users share the same congested path, then each user receives service in proportion to its resource token rate. This is the same result one would expect from using a weighted fair queueing scheduler on every link, with the weights set to the users' token rate. In the second experiment, we show that by using dynamic routing and load balancing, we are able to achieve the same result – that is, each user to receive service in proportion to its token rate – in a more general configuration where simply using weighted fair queueing scheduler on every link is not sufficient. In the third experiment, we show how load balancing can significantly increase the overall resource utilization. Finally, the fourth experiment shows how the behaviors observed in the previous experiments scale to a larger topology.

3.1 Experiment Design

We have implemented a packet level simulator which supports both Distance Vector (DV) and Shortest Path First (SPF) routing algorithms. To support load balancing we extended these algorithms to compute the k-th shortest paths. The time interval between two route updates is uniformly distributed between 0.5 and 1.5 of the average value. As shown in [10] this choice avoids the route-update self-synchronization. In SPF, when a node receives a routing message, it first updates its routing table and then forwards the message to all its neighbors, excepting the sender. The routing messages are assumed to have high priority, so they are never lost. In the followings we compare the following schemes:

- BASE – this scheme models today's best-effort Internet, and it is used as a baseline in our comparison. The routing protocol uses the number of hops as the distance metric and it is implemented by either DV or SPF. This scheme does not implement service differentiation, i.e., both marked and unmarked packets are identically treated.

- STATIC – this scheme implements the same static routing as BASE. In addition, it implements *LIRA* by computing the link cost as described in Section 2.2, and marking packets at each ingress node according to the algorithm shown in Figure 1.

- DYNAMIC-k – this scheme adds dynamic routing and load balancing to STATIC. The routing protocol uses a modified versions of DV/SPF to find the first k shortest paths. Note that DYNAMIC-1 is equivalent to STATIC.

Each router implements a FIFO scheduling discipline with a shared buffer and a drop-tail management scheme. When the buffer occupancy exceeds a predefined threshold, newly arrived unmarked packets are dropped. Thus, the entire buffer space from the threshold up to its total size is reserved to the in-profile traffic[5]. Unless otherwise specified, throughout all our experiments we use a buffer size of 256 KB and a threshold of 64 KB.

The two main performance indices that we use in comparing the above schemes are the user *in-profile* and user *overall* throughputs. The user in-profile throughput repre-

[5] We note that this scheme is a simplified version of the RIO buffer management scheme proposed by Clark and Wroclawski [4]. In addition, RIO implements a Random Early Detection (RED) [9] dropping policy, instead of drop-tail, for both in- and out-of profile traffic. RED provide an efficient detection mechanism for the adaptive flows, such as TCP, allowing them to gracefully degrade their performances when congestion occurs. However, since in this study we are not concerned with the behavior of individual flows, for simplicity we chose to not implement RED.

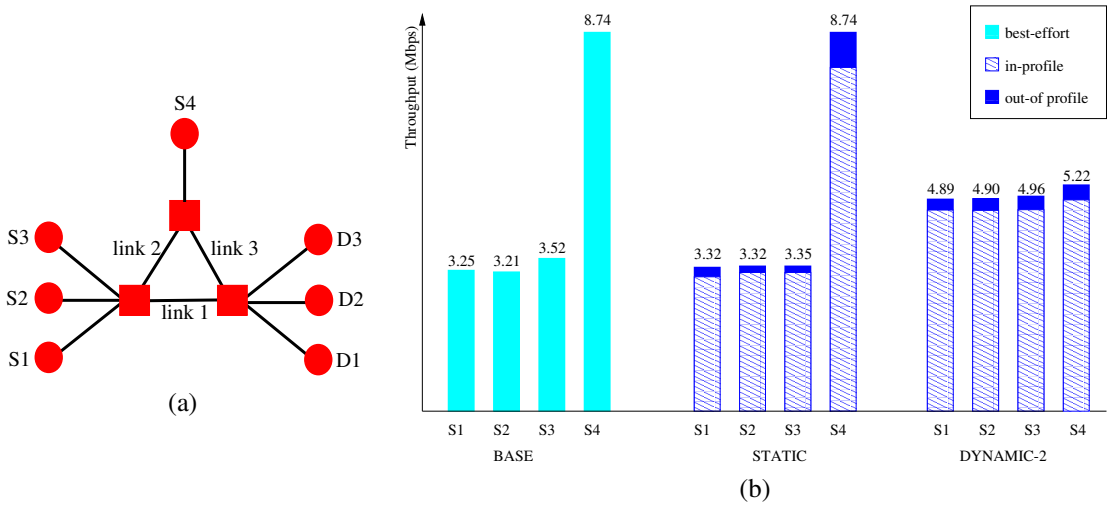


Figure 5. (a) Topology used in the second experiment. $S1$, $S2$, $S3$, and $S4$ send all their traffic to $D1$, $D2$, and $D3$, respectively. (b) The throughputs of all users under BASE, STATIC, and DYNAMIC-2.

sents the rate of the user aggregate *in-profile* traffic delivered to its destinations. The overall throughput represents the user's *entire* traffic — i.e., including both the in- and out-of profile traffic — delivered to its destinations. In addition, we use user *dropping rate* of the in-profile traffic to characterize the level of service assurance.

Recent studies have shown that the traffic in real networks exhibits the self-similar property [5, 23, 24, 30] — that is, the traffic is bursty over widely different time scales. To generate self-similar traffic we use the technique originally proposed in [30], where it was shown that the superposition of many ON-OFF flows with ON and OFF periods drawn from a heavy tail distribution, and which have fixed rates during the ON period results in self-similar traffic. In particular, in [30] it is shown that the aggregation of several hundred of ON-OFF flows is a reasonable approximation of the real end-to-end traffic observed in a LAN.

In all our experiments, we generate the traffic by drawing the length of the ON and OFF periods from a Pareto distribution with the power factor of 1.2. During the ON period a source sends packets with sizes between 100 bytes and 1000 bytes. The time to send a packet of minimum size during the ON period is assumed to be the time unit in computing the length of the ON and OFF intervals.

Due to the high overhead incurred by a packet-level simulator, such as ours, we limit the link capacities to 10 Mbps and the simulation time to 200 sec. We set the average interval between routing updates to 5 sec for the small topologies used in the first three experiments, and to 3 sec for the large topology used in the last experiment. In all experiments, the traffic starts at time $t = 20$ sec. The choice of this time is such that to guarantee that the routing algorithm finds at least one path between any two nodes by time t. In order to eliminate the transient behavior, we start our measurements at time $t = 50$ sec.

3.2 Local Fairness and Service Differentiation

This experiment shows that if all users send their traffic along the same congested path, they get service in proportion to their token rate, as long as there is enough demand. Consider the topology in Figure 4(a), where users $S1$, $S2$, and $S3$ send traffic to $D1$. Figure 4(b) shows the user overall throughputs over the entire simulation under BASE. As it can be seen, $S1$ gets significantly more than the other two. In fact, if the traffic from all sources were continuously backlogged, we expect that $S1$ to get half of the congested links 5 and 6, while $S2$ and $S3$ to split the other half. This is because even though each user sends at an average rate higher than 10 Mbs, the queues are not continuously backlogged. This is due to the bursty nature of the traffic and due to the limited buffer space at each router.

Next, we run the same simulation for the STATIC scheme. To each user we assign the same token rate, and to each link we associate the same fixed cost. Figure 4(b) shows the user overall and in-profile throughputs. Compared to BASE, the overall throughputs are more evenly distributed. However, the user $S1$ still gets slightly better service, i.e., its in-profile throughput is 3.12 Mbps, while the in-profile throughput of $S2$/$S3$ is 2.75 Mbps. To see why, recall from Eq. (1) that link cost accurately reflects the level of congestion on that link. Consequently, in this case links 5 and 6 will have the highest cost, followed by link 4, and then the other three links. Thus, $S2$ and $S3$ have to "pay" more than $S1$ per marked bit. Since all users have the same token rates, this translates into lower overall throughputs for $S2$ and $S3$, respectively.

To illustrate the relationship between the user's token rate and its performance, we double the token rate of $S2$.

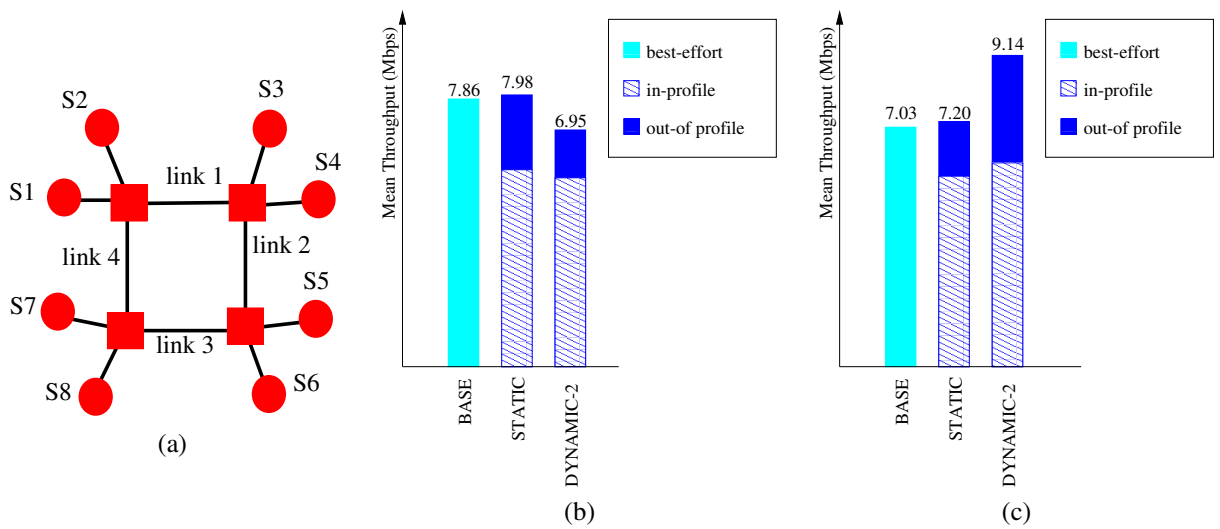


Figure 6. (a) Topology used in the third experiment. Mean throughputs when (b) load is balanced, and (c) when it is unbalanced, i.e, S3 and S4 are inactive.

Figure 4(c) shows the overall and in-profile throughputs of each user. In terms of in-profile traffic user $S2$ gets roughly twice the throughout of $S3$ (i.e., 4.27 Mbps vs. 2.18 Mbps).

Finally, we note that there was no marked packets dropped in any of the above simulations. For comparison more than 60 % of the out-of profile traffic was dropped.

3.3 User Fairness and Load Balancing

In this section we show how dynamic routing and load balancing help to improve user level fairness and achieve better resource utilization. Consider the topology in Figure 5 where users $S1$, $S2$, $S3$ and $S4$ send traffic to users $D1$, $D2$ and $D3$. Again the fixed costs of all links are equal, and all users are assigned the same token rate.

Figure 5(b) shows the overall and in-profile throughputs of $S1$, $S2$, $S3$ and $S4$ under BASE, STATIC and DYNAMIC-2, respectively. When BASE and STATIC are used, each user sends always along the shortest paths. This results in $S1$, $S2$ and $S3$ sharing link 1, while $S4$ using alone link 3. As a consequence $S4$ receives significantly better service than the other three users. Since it implements the same routing algorithm, STATIC does not improve the overall throughputs. However, compared with BASE, STATIC guarantees that in-profile packets are delivered with very high probability (again, in this experiment, no marked packets were dropped). On the other hand, when DYNAMIC-2 is used each user receives almost the same service. This is because, now users $S1$, $S2$ and $S3$ can use both routes to send their traffic, which allow them to compete with user $S4$ for link 3. User $S4$ still maintains a slightly advantage, but now the difference between its overall throughput and the overall throughputs of the other users is less than 7%. In the case of the in-profile traffic this difference is about 5%. As in the previous experiment the reason for this difference is because when competing with $S4$, the other users have to pay, besides link 3, for link 2 as well.

Thus, by taking advantage of the alternate routes, our scheme is able to achieve fairness in a more general setting. At the same time it is worth noting that the overall throughput also increases by almost 7 %. However, in this case, this is mainly due to the bursty nature of $S4$'s traffic which cannot use the entire capacity of link 3 when it is the only one to use it, rather than load balancing.

3.4 Load Distribution and Load Balancing

This experiment shows how the load distribution affects the effectiveness of our load balancing scheme. For this purpose, consider the topology in Figure 6(a). In the first simulation we generate flows that have the source and the destination uniformly distributed among users. Figure 6(b) shows the means of the overall throughputs under BASE, STATIC, and DYNAMIC-2, respectively [6]. Due to the uniformity of the traffic pattern, in this case BASE performs very well. Under STATIC we get slightly larger overall throughput, mainly due to our congestion control scheme, which admits a marked packet only if there is a high probability to be delivered. However, under DYNAMIC-2 the performances degrades. This is because there are times when our probabilistic routing algorithm selects longer routes, which leads to inefficient resource utilization.

Next, we consider an unbalanced load by making users $S3$ and $S4$ inactive. Figure 6(c) shows throughput means under BASE, STATIC, and DYNAMIC-2, respectively. As it can be noticed, using DYNAMIC-2 increases the mean by 30 %. This is because under BASE and STATIC schemes

[6] We have also computed standard deviations for each case: the largest standard deviation was 0.342 for the overall throughput under STATIC scheme, and 0.4 for the in-profile throughput under DYNAMIC-2.

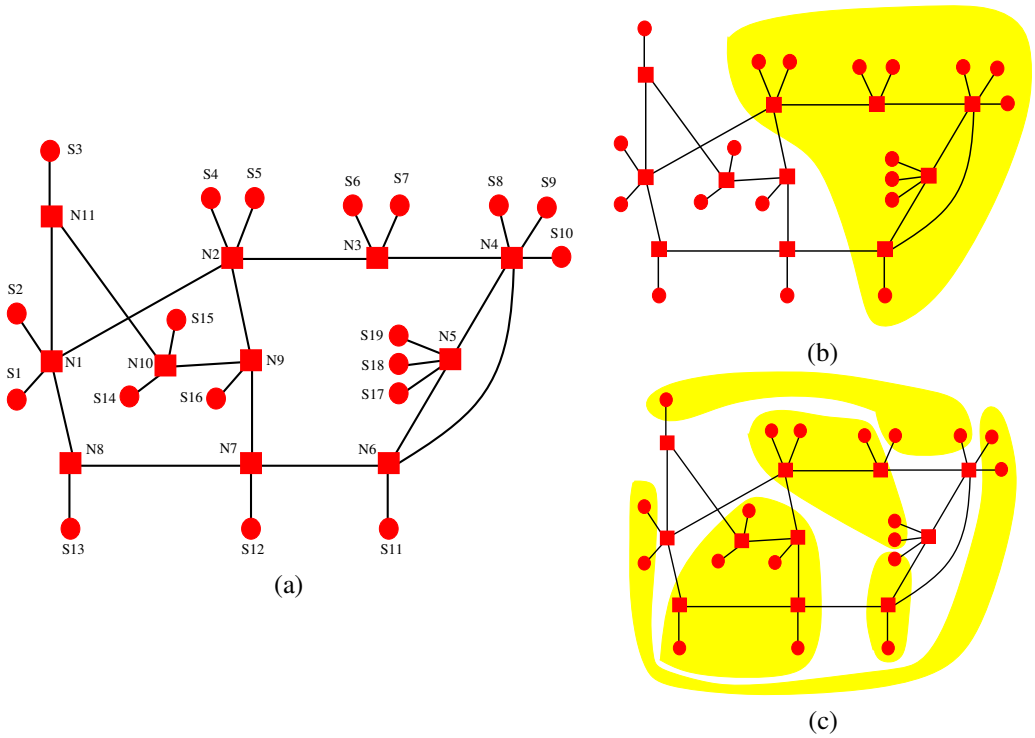


Figure 7. Topology similar to the T3 topology of the NSFNET backbone network containing the IBM NSS nodes.

the entire traffic between $S1$, $S2$ and $S5$, $S6$ is routed through links 3 and 4 only. On the other hand, DYNAMIC-2 takes advantage on the alternate route through links 1 and 2.

Finally, in another simulation not shown here we considered the scenario in which $S5$, $S6$, $S7$, and $S8$ send their entire traffic to $S3$ and $S4$, respectively. In this case DYNAMIC-2 outperforms almost two times STATIC and BASE both in terms of in-profile and overall throughputs. This is again because BASE and STATIC use exclusively links 3 and 2, while DYNAMIC-2 uses the other two links as well.

3.5 Large Scale Example

In this section we consider a larger topology that closely resembles the T3 topology of the NSFNET backbone containing the IBM NSS nodes (see Figure 7). The major difference is that in order to limit the simulation time we assume 10 Mbps links, instead of 45 Mbps. We consider the following three scenarios.

In the first scenario we assume that load is uniformly distributed, i.e., any two users communicate with the same probability. Figure 8(a) shows the results for each scheme, results which are consistent with the ones obtained in the previous experiment. Due to the congestion control which reduces the number of dropped packets in the network, STATIC achieves higher throughput than BASE. On the other hand, the dynamic routing and load balancing are not effective in this case, since they tend to generate longer routes which leads to inefficient resource utilization. This is illustrated by the decrease of the overall and the in-profile throughputs under DYNAMIC-2 and DYNAMIC-3, respectively.

In the second scenario we assume unbalanced load. More precisely, we consider 11 users (covered by the shaded area in Figure 7(b)) which are nine times more active than the other, i.e., they send/receive nine times more traffic than the others.[7] Unlike the previous scenario, in terms of overall throughputs DYNAMIC-2 outperforms STATIC by almost 8 %, and BASE by almost 20 % (see Figure 8(b)). This is because DYNAMIC-2 is able to use some of the idle links from the un-shaded partition. However, as shown by the results for DYNAMIC-3, as the number of alternate paths increases both the overall and in-profile throughputs start to decrease.

In the last scenario we consider the partition of the network shown in Figure 7(c). For simplicity, we assume that only users in the same partition communicate between them. This scenario models a virtual private network (VPN) setting, where each partition corresponds to a VPN. Again, DYNAMIC-2 performs the best[8], since it is able to make

[7] This might model the real situation where the east coast is more active between 9 and 12 a.m. EST, than the west coast.

[8] The mean of the user overall throughput under DYNAMIC-2 is 15 %

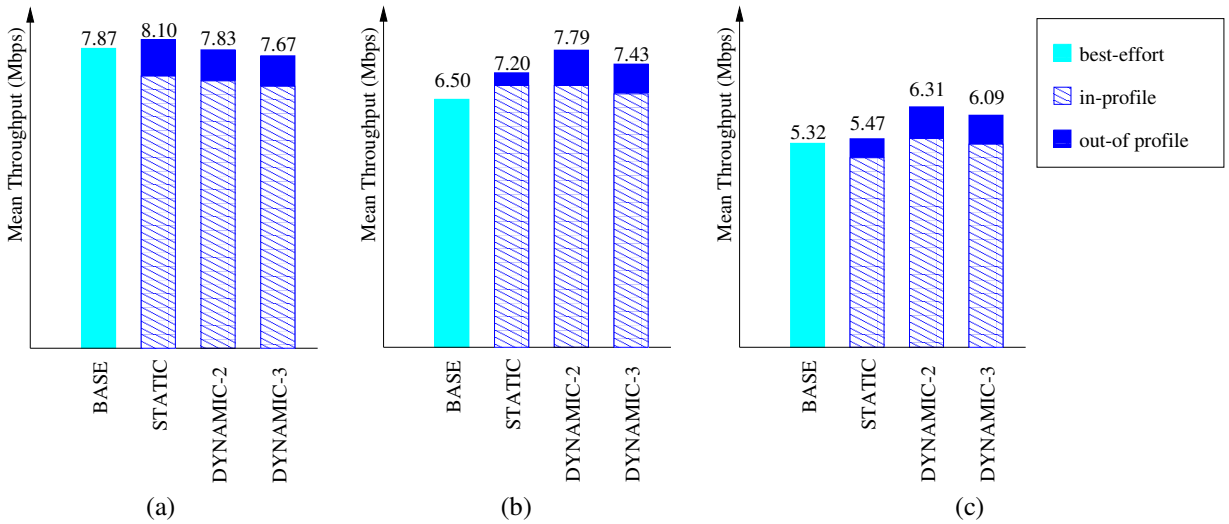


Figure 8. The throughputs when the load is balanced (Figure 7(a)), (b) unbalanced ((Figure 7(b)), and (c) when the network is virtually partitions (Figure 7(c)).

use of some links between partitions that otherwise would remain idle.

Finally, we note that across all simulations presented in this section the dropping rate for the marked packets was never larger than 0.3 %. At the same time the dropping rate for the unmarked packets was over 40 %.

3.6 Summary of Results

Although the experiments in this section are far from being exhaustive, we believe that they give a reasonable image of how our scheme performs. First, our scheme is effective in providing service differentiations at the user level. Specifically, the first two experiments show that users with similar communication patterns get service in proportion to their token rates. Second, at least for the topologies and the traffic model considered in these experiments, our scheme ensures that marked packets are delivered to the destination with high probability.

Consistent with other studies [19], these experiments show that performing dynamic routing and load balancing make little sense when the load is already balanced. In fact, doing dynamic routing and load balancing can actually hurt, since, as noted above, this will generate longer routes which may result in inefficient resource utilization. However, when the load is unbalanced, using DYNAMIC-k can significantly increase the utilization and achieve a higher degree of fairness.

Finally, we note that the in-profile dropping rate decreases as the the number of alternate paths increases. For example in the last experiment in the first two scenarios the dropping rate is no larger than 0.3 % under STATIC and 0 under DYNAMIC-2 and DYNAMIC-3, respectively, while in the last scenario the percentage decreases from 0.129 % for STATIC, to 0.101 % for DYNAMIC-2, and to 0.054 % for DYNAMIC-3.

larger than under STATIC, and 18 % larger than under BASE.

4 Discussion

In this paper, we have studied a differential service model, called *LIRA*, in which the service profile is specified in terms of resource tokens instead of absolute bandwidth. Since the exact bandwidth of marked bits that a customer can receive from such a service is not *known* a priori, a natural question to ask is why such a service model is interesting.

There are several reasons. First, we believe that the apriori specification of an absolute amount of bandwidth in the service profile, though desirable, is not essential. In particular, we believe that the essential aspects that distinguish Diffserv from Intserv are the followings: (a) the service profile is used for traffic aggregate much coarser than per flow traffic, and (b) the service profile is defined over a timescale larger than the duration of individual flows, i.e. service profile is rather *static*. Notice that the degree of traffic aggregation directly relates to the spatial granularity of the service profile. On the one hand, if each service profile is defined for only one destination, we have the smallest degree of traffic aggregation. If there are N possible egress nodes for a user, N independent service profiles need to be defined. Network provisioning is relatively easy as the entire traffic matrix between all egress and ingress nodes is known. However, if a user has a rather dynamic distribution of egress nodes for its traffic, i.e., the amount of traffic destined to each egress node varies significantly, and the number of possible egress nodes is large, such a scheme will significantly reduce the chance of statistical sharing. On the other hand, if each service profile is defined for all

egress nodes, we have the largest degree of traffic aggregation. Only one service profile is needed for each user regardless the number of possible egress nodes. In addition to a smaller number of service profiles, such a service model also allows all the traffic from the same user, regardless of its destinations, to *statistically* share the same service profile. The flip side is that it makes it difficult to provision network resources. Since the traffic matrix is not known apriori, the best-case scenario is when the network traffic is evenly distributed, and the worst-case scenario is when all traffic goes to the same egress router.

Therefore, it is very difficult, if not impossible, to design service profiles that (1) are static, (2) support coarse spatial granularity, (3) are defined in terms of absolute bandwidth, and at the same time achieve (4) high service assurance and (5) high resource utilization. Since we feel that (1), (2), (4) and (5) are the most important for differential services, we decide to give up (3).

Fundamentally, we want a service profile that is static and egress node/path independent. However, to achieve high utilization, we need to explicitly address the fact that congestion is a local and dynamic phenomenon. Our solution is to have two levels of differentiation: (a) the user or service-profile level differentiation, which is based on resource token arrival rate. This is static and path independent; (b) the packet level differentiation, which is a simple priority between marked and unmarked packets and weighted fair share among marked packets. By dynamically setting the cost of each marked bit as a function of the congestion level of the path it traverses, we set up the linkage between the static/path-independent and the dynamic/path-dependent components of the service model.

A second reason for which our service model may be acceptable is that users may care more about the *differential* aspect of the service than the guaranteed bandwidth. For example, if user A pays twice as much as user B, user A would expect to have roughly twice as much traffic delivered as user B during congestion if they share same congested links, which is exactly what we accomplish in *LIRA*.

A third reason for which a fixed-resource-token-rate-variable-bandwidth service profile may be acceptable is that the user traffic is usually bursty over multiple timescales[5, 23, 30]. Thus, there is a fundamental mismatch between an absolute bandwidth profile and the bursty nature of the traffic[9].

We do recognize the fact that it is desirable for both the user and the ISP to understand the relationship between the user's resource token rate and its expected capacity. This can be achieved by measuring the rate of marked bits given a fixed token rate. Both the user and the ISP can perform this measurement. In fact, this suggests two possible scenarios in which *LIRA* can be used to provide a differential service with an expected capacity defined in terms of absolute bandwidth. In the first scenario, the service is not transparent. Initially, the ISP will provide the user with the following relationship

$$expected_capacity = f(token_rate, traffic_mix) \quad (7)$$

based on its own prior measurement. The user will measure the expected capacity and then make adjustments by asking for an increase or a decrease in its resource token rate. In the second scenario, the service is transparent. Both the initial setting and the subsequent adjustments of the service profile in terms of number of token rate will be made by the ISP only.

Therefore, one way of thinking about our scheme is that it provides a flexible and efficient framework for implementing a variety of Assured Services. In addition, the dynamic link cost information and the statistics of the resource token bucket history provide good feedback both for individual applications to perform runtime adaptation, and for the user or the ISP to do proper accounting and provisioning.

5 Related Work

Our work is highly influenced by Clark and Wroclawski's Assured Service proposal [3, 4]. The key difference is that we define service profiles in units of resource tokens rather than absolute bandwidth. In addition, we propose a resource accounting scheme and an integrated set of algorithms to implement our service model.

Another related proposal is the User-Share Differentiation (USD) [29] scheme, which does *not* assume absolute bandwidth profiles either. In fact, with USD, a user is assigned a share rather than a token-bucket-based service profile. For *each* congested link in the network traversed by the user's traffic, the user shares the bandwidth with other users in proportion to its share. The service provided is equivalent to one in which *each* link in a network implements a weighted fair queueing scheduler where the weight is the user's share. With USD, there is little correlation between the share of a user and the aggregate throughput it will receive. For example, two users that are assigned the same share can see drastically different aggregate throughputs. A user that has traffic for many destinations (thus traverse many different paths) can potentially receive much higher aggregate throughput than a user that has traffic for only a few destinations.

There is a huge body of related work that addresses the resource allocation problem both for single and multiple resources. However, to the best of our knowledge, none of

[9] Some recent measurements show that the aggregate traffic over Internet backbone links are not very bursty. We note that this is not inconsistent with the observations that the aggregate traffic from a campus to the Internet exhibits long range dependency.

the existing proposals address the problem of allocating resources for traffic aggregate that has a large spatial granularity. In general, they are limited in scope to allocating resources along individual paths only. In addition, these schemes usually require each user to maintain per resource state, or/and each resource to maintain per user state. In the following, we discuss several of the more relevant schemes.

Waldspurger and Weihl have proposed a framework for resource management based on lottery tickets [27, 28]. Each client is associated a certain number of tickets which encapsulate its resource rights. The number of tickets a user receives is similar to the user's income rate in *LIRA*. This framework was shown to provide flexible management for various single resources, such as disk, memory and CPU. However, they do not give any algorithm(s) to coordinate tickets allocation among multiple resources.

Ferguson et al. proposed a flow control economy to allocate network resources such as links and buffers among competing virtual circuits (VCs) [7, 8]. In this model, each VC is endowed certain funds for buying resources. The VC's goal is to buy a minimum capacity on all links along its path, and use the extra money to minimize the average end-to-end delay. The resource prices are set so that the supply and the demand are balanced. It has been shown that such economy converges and the resulted allocations are pareto-optimal.

MacKie-Mason and Varian have proposed a model, called "smart markets", in which each packet carries a bid that represents how much the user is willing to pay for it [20]. At each congested link along a path a cutoff price is computed and only packets that have a higher bid are forwarded; the other packets are buffered. At the service level it is unclear how the priorities of individual packets translate into expected network throughput. In addition, in order to achieve high level of service assurance, a user needs to know the smallest bid along the path. No mechanisms are given to propagate this information to the users.

Awerbuch et al. [1] have proposed an on-line reservation algorithm to maximize the throughput in a network where the *duration* of each reservation is known in advance. The algorithm guarantees that the throughput is within $O(\log nT)$ factor of the throughput achieved by an optimal off-line algorithm, where n is the number of nodes and T is the maximum duration of a reservation. In the scheme, each link is associated with a cost that is a exponential function of its current utilization. Also, each connection is associated with a profit which is received only if the request is granted. The goal of the algorithm is then to maximize the overall profit. While this algorithm differs significantly form ours both in assumptions and goals, we note that the mechanisms used to implement *LIRA* can also be used to implement this scheme.

Kelly et. al [15, 16] have considered the problem of bandwidth allocation between competing streams with elastic traffic. In particular, they propose a mathematical model to analyze the stability and fairness of a class of rate controlled algorithms. In this model each user chose the charge per unit of time that it is willing to pay for a route. In turn the network computes the user rates according to a proportionate criterion. However, they only consider the model where resources are allocated on the basis of per virtual circuit.

To increase resource utilization, in this paper we propose performing dynamic routing and load balancing among the best k shortest paths between source and destination. In this context, one of the first dynamic routing algorithms, which uses the link delay as metric, was the ARPANET shortest path first [21]. Unfortunately, the sensitivity of this metric when the link utilization approaches unity resulted to relative poor performances. Various routing algorithms based on congestion control information were proposed elsewhere [12, 13]. The unique aspect of our algorithm is that it combines dynamic routing, congestion control and load balancing together. Also we alleviate the problem of system stability which plagued many of the previous dynamic routing algorithms by defining a more robust cost function and probabilistically binding a flow to a route. We also note that our link cost is similar to the one used in [19]. In particular, it can be shown that when all links have the same capacity, our link cost is within a constant factor of the cost of shortest-dist(P, 1) algorithm presented in [19]. It is worth noting that shortest-dist(P, 1) performed the best among all the algorithms studied in [19].

6 Summary

We study models and algorithms that support Assured Service with service profiles defined over large spatial granularities. We propose a service model in which the service-profile is defined in units of resource tokens rather than the absolute bandwidth, and an accounting scheme that dynamically determines the number of resource tokens charged for each in-profile packet. We present a set of algorithms that efficiently implement the service model. In particular, we introduce three techniques: (a) distributing path costs to all edge nodes by leveraging existing routing infrastructure; (b) binding a flow to a route (route-pinning) without maintaining per flow state; (c) multi-path routing and probabilistic binding of flows to paths to achieve load balancing. Simulation results are presented to demonstrate the effectiveness of the approach. To the best of our knowledge, this is the first complete scheme that explicitly addresses the issue of large spatial granularities.

While these techniques are developed in the context of supporting Assured Service, they may be useful in other contexts. For example, by combining the route-pinning technique with the SCED+ algorithm proposed in [6], guar-

anteed or premium service can be provided without the need for per flow management at core routers.

As future work, we plan to extend this work to support multiple ISP environments, multicast communication, and both sender and receiver-based charging schemes.

References

[1] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proceedings of FOCS'93*, October 1993.

[2] S. Blake. Some issues and applications of packet marking for differentiated services, July 1997. Internet Draft.

[3] D. Clark. Internet cost allocation and pricing. *Internet Economics, L. W. McKnight and J. P. Bailey (eds.)*, pages 215–252, 1997.

[4] D. Clark and J. Wroclawski. An approach to service allocation in the internet, July 1997. Internet Draft.

[5] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic evidence and possible causes. In *Proceedings of the ACM SIGMETRICS 96*, pages 160–169, Philadelphia, PA, May 1996.

[6] R. L. Cruz. Sced+: Efficient Management of Quality of Service Guarantees. In *Proceedings of INFOCOM'98*, pages 625–642, San Francisco, CA, 1998.

[7] D. F. Ferguson. *The Application of Microeconomics to the design of Resource Allocation and Control Algorithms in Distributed Systems*. PhD thesis, 1989.

[8] D. F. Ferguson, C. K. Nikolau, and Y. Yemini. An economy flow control in computer networks. In *Proceedings of INFOCOM'90*, 1990.

[9] S. Floyd and V. Jacobson. Random early detection for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, July 1993.

[10] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. In *Proceedings of ACM SIGCOMM'93*, pages 33–44, San Francisco, CA, September 1993.

[11] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (cidr): An address assignment and aggregation strategy (rfc 1519), September 1993.

[12] D. W. Glazer and C. Tropper. A new metric for dynamic routing algorithms. *IEEE Transaction on Communication*, 38(3):360–367, March 1990.

[13] S. J. Golestani. *A Unified Theory of Flow Control and Routing in Communication Networks*. PhD thesis, Department of Computer Science, University of Virginia, 1980.

[14] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and R. Viswanathan. Erica switch algorithm: A complete description, August 1996. ATM Forum/96-1172.

[15] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecomunications*, 8:33–37, 1997.

[16] F. P. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 40, June 1998.

[17] K. Kilkki. Simple integrated media access (sima), June 1997. Internet Draft.

[18] L. Kleinrock. *Queueing Systems*. John Wiley and Sons, 1975.

[19] Q. Ma, P. Steenkiste, and H. Zhang. Routing high-bandwidth traffic in max-min fair share networks. In *Proceedings of the ACM SIGCOMM'96*, pages 206–217, Palo Alto, CA, October 1996.

[20] J. MacKie-Mason and H. Varian. Pricing the interenet. *Public Access to the Internet, B. Kahin and J. Keller (ed.)*, 1994.

[21] J. M. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the arpanet. *IEEE Transaction on Communication*, 28(5):711–719, May 1980.

[22] K. Nichols, V. Jacobson, and L. Zhang. An approach to service allocation in the internet, November 1997. Internet Draft.

[23] V. Paxon and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[24] V. Paxon and S. Floyd. Why we don't know how to simulate the internet. In *Proceedings of the Winder Communication Conference*, December 1997.

[25] L. Roberts. Enhanced prca (proportionat rate control algorithm), August 1994. ATM Forum/94-0735R1.

[26] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service (rfc 2212), September 1997.

[27] C. A. Waldspurge. *Lottery and Stride Scheduling: Flexible Proportional -Share Resource Management*. PhD thesis, 1995.

[28] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of OSDI 94*, pages 1–12, November 1994.

[29] Z. Wang. User-share differentiation (usd) scalable bandwidth allocation for differentiated services, May 1998. Internet Draft.

[30] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. In *Proceedings of the ACM SIGCOMM'95*, pages 100–113, Boston, MA, August 1995.

[31] J. Wroclawski. Specification of the controlled-load network element service (rfc 2211), September 1997.

[32] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Communications Magazine*, 31(9):8–18, September 1993.

Processor Scheduling in Routers for RSVP Message Processing

Martin May† T. V. Lakshman† Anwar Elwalid*

† Bell Labs
Lucent Technologies
101 Crawfords Corner Road
Holmdel NJ 07733 USA

* Bell Labs
Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ USA

Abstract

We propose CPU-scheduling schemes for processing RSVP messages in routers. We consider scenarios where there can be both bandwidth and CPU contention, i.e., the router processors are sufficiently loaded that RSVP messages may be dropped at the processors. When this happens, it is possible that link capacity is available but yet reservations do not succeed, i.e., reservation blocking is possible even if link-bandwidth is available. We show that the common first-in-first-out (FIFO) processing of RSVP messages leads to poor performance when the processor utilization is high. Link bandwidth can be wasted due to suboptimal control message processing and link utilization can oscillate considerably causing poor performance. We propose simple alternate adaptive scheduling mechansims that keep the link utilization high and hence reduce the request blocking rate. The adaptation changes the relative rates of processing so that PATH and RESERVE messages are processed at a higher rate when the current reserved link utilization is low, and the relative rate of processing of TEAR-DOWNs is increased when the link utilization is high. Apart from link-state, the adaptation also takes into account recent resource request sizes, and the relative link speeds when the same processor controls multiple links of differing speeds. We show by simulations that the proposed simple adaptive scheme has a lower request blocking rate than both FIFO and weighted round robin with non-adaptive weights.

I. Introduction

Providing differentiated services in an IP network has become an area of strong research and commercial interest. One possible approach is the one where type-of-service (TOS) bits defined in the IP packet header are interpreted in an agreed upon manner [8] so as to provide differentiated services. Another possibility is the reservation based approach which uses signaling to reserve resources for the duration of a flow or flow aggregrates. For this case, the current protocol of choice for signaling resource reservations is the Reservation Setup Protocol (RSVP) [1]. RSVP, as an example, can be used in conjunction with service models such as guaranteed rate and controlled load [9] to request quality of service for certain flows.

RSVP facilitates exchange of resource reservation information among routers in the Internet and it is a soft state protocol which relies upon periodic refreshes to maintain router state information. Refreshes that are not sent or processed within a timer expiration interval cause the established flow state to be torn down. The periodic refreshes and consequent soft state information in routers permit the protocol to operate robustly in the presence of route changes, lost signaling messages, and without always requiring explicit tear-down of state. By using soft state protocols, IP networks can offer services comparable to those in virtual circuit networks with explicit connection set-up, tear-down and connection specific routing.

However, the periodic refreshes cause the processing load in routers to increase with the number of RSVP flows that have been set-up even if these flows are not actively sending packets. There are two types of scalability issues that can potentially arise when using RSVP. One is due to the need to perform packet classification to determine which RSVP flow (if any) that an arriving packet belongs to. This has to be done for every arriving packet and can become a potential bottleneck when the number the number of established RSVP flows is large (in the thousands) and link rates are very high. We do not consider this issue in this paper. Instead, we assume that packet classification and forwarding is done (by specialized hardware or by processors) in a subsystem different from the processor processing RSVP signaling messages [6]. The second scalability issue is that caused by the control processor, which processes RSVP (and possibly other) messages, becoming bottlenecked due to high message processing load.

The RSVP load offered to the control processor consists of i) traffic due to reservation setups and teardowns and ii) traffic due to refreshes generated by RSVP-flows that have already been established. Even

though refreshes require very little processing, the offered traffic due to refreshes increases with the number of RSVP flows in-progress. Also, delayed processing of refreshes can cause timer expiration in downstream routers and cause costly spurious tear-downs of flows. Hence, refreshes need to be processed at a higher priority to minimize tear-downs of flows in progress. We consider both the case where there is a control processor per interface and the case where a control processor is used for a group of interfaces (links). Even in the case where there is a control processor per interface, the processing load due to refreshes can be very high as for instance when the link speed is very high and the average request size is small (resulting in a very large number of established flows). Recent studies have shown that core Internet routers, which have to handle and maintain about 45000 flows per minute [7], have saturated CPU capacities. In this environment, simplistic RSVP message handling strategies are inadequate and can lead to poor performance.

It may be argued that the choice of processor should be dictated by appropriate traffic engineering considerations and so the processor should never be highly utilized. Even in this case, there can be temporary overloads of the control processor. This can be due to a mass call-in like scenario which generates a large number of new reservation requests or tear-downs. Link failures causing a large number of route changes can cause a temporary overload which if dealt with using FIFO scheduling may cause unnecessary tear-downs.

Also, the control processor, most likely, is not going to be processing RSVP messages alone. Instead, it may also handle route updates, multicast protocols, and perhaps slow-path forwarding of IP packets which need options processing. Recent publications have shown that routing table sizes are increasing rapidly and will probably increase processing loads for updates. Degermark, in [4], measured the time that a router CPU needs to build up the routing table as being between 50ms and 100ms. In conjunction with the observations of Vern Paxson about current routing instability, one should expect large transient loads on router CPUs. With a burst of routing instabilities, and re-calculations of OSPF routes, there can be high CPU usage strictly due to routing tasks. These are factors that may cause RSVP message processing to experience processing bottlenecks.

When the control processor becomes a bottleneck, it is possible that link capacity is available but yet reservation requests do not succeed, i.e., reservation blocking is possible even when link capacity is available. Clearly, appropriate scheduling of the control processor can be used to maximally utilize the link capacity within the constraints of the available processing capacity. In this paper, we first show that FIFO processing of messages results in several problems such as oscillating link utilization, wastage of available link bandwidth, and a higher request blocking rate. We then propose adaptive processor scheduling schemes that adjust the relative priorities with which different RSVP messages are processed based on link utilization and set-up, tear-down request sizes seen in the recent past. We evaluate the performance and efficacy of these schemes using simulations. These schemes are for the case where there is a control processor per interface. Next we consider the case where a control processor controls multiple interfaces. Here, there is the additional constraint of fairness. The processor scheduling should not result in some interfaces being highly utilized while requests on other interfaces are persistently blocked even when there is available capacity. We generalize the schemes proposed for the processor per interface case to this case of processor per group of interfaces.

The paper is organized as follows: The next section describes the basic RSVP message processing model that we use. Section 3 describes our adaptive scheduling scheme. Section 4 points out problems with existing RSVP implementations and their impact on the overall protocol performance (such as oscillating link utilization) and also has a performance study of the schemes that we propose. Concluding remarks are in Section 5.

II. RSVP Message Processing

In this section, we briefly summarize aspects of the RSVP protocol that are relevant to the study of processor scheduling issues. RSVP is a receiver-oriented resource reservation protocol. Reservations are initiated when a sender requests a resource reservation (we only consider bandwidth reservations in this paper) such as a request for a certain amount of bandwidth. This request is signaled to the network using a PATH message. The PATH message is routed along the network to its destination (or set of destinations) through a series of routers just as any other IP packet. The router establishes a flow-state for the flow (or aggregated flows) indicated by this PATH request and then the PATH message is propagated[1]. Also, it starts a timer which tears-down the flow state if no RESV or UPDATE messages are received for the flow before timer expiration.

When a PATH message reaches its destination, the receiver sends back a RESV message with a bandwidth request which is possibly different from that requested in the PATH message. When a router receives a RESV message for which there is an established flow-state, it

[1] To multiple next-hops in the case of multicasts. However, we restrict ourselves to the unicast case in this paper

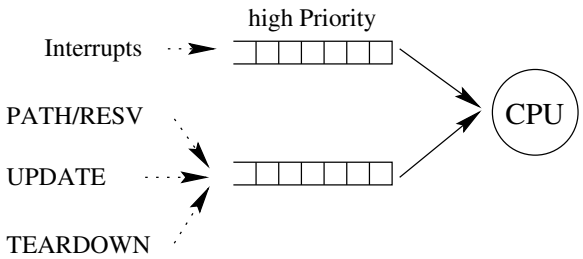


Fig. 1. FIFO scheduling for RSVP Messages with high-priority timer expiration processing

commits the requested bandwidth to the flow if sufficient bandwidth is available. Otherwise, the request is blocked. From the point-of-view of this router, the flow is now in progress. Each message receipt, resets the flow-state tear-down (refresh) timer. The flow in progress is kept alive by periodic UPDATE messages which are generated by the sender. Each router, upon processing of UPDATE messages, resets its tear-down timer and propagates the UPDATE message. A flow is torn-down and resources released, when either an explicit TEAR-DOWN message is received or when there is a refresh timer expiration.

In the absence of any agreed upon timer adaptation mechanisms, we assume that the refresh timers are set to fixed values that are at least an order of magnitude larger than typical round-trip times.

Our initial processing model is as follows: There is a control processor per high-speed link. RSVP signaling messages that arrive over the link are sent to the control processor where they are put in a queue for further processing. The simplest service discipline for the queue is FIFO with no distinctions made between the different message types. To ensure that interrupts due to timer expirations are not lost, these are processed with high priority. Therefore, as shown in Figure 1, the processor scheduling is done using priority queueing with two queues. The high priority queue processes timer expirations in a FIFO manner, and the low priority queue processes all RSVP messages in a FIFO manner. If the control CPU is not a bottleneck, this discpline is adequate. FIFO processing is not adequate when the control processor is overloaded or has high utilization and so has large processing queues.

As an example, consider the case when the link-utilization is very high. In this case, processing of PATH or RESV ahead of TEAR-DOWN messages is not fruitful since bandwidth is unlikely to be available to satisfy new requests and so requests will be blocked. Under high-link utilization, it would be beneficial to process TEAR-DOWN messages in the queue before processing PATH messages. However, with FIFO processing we cannot adopt such link-state dependent preferential behavior. Similarly, when the link utilization is low, we can defer processing of TEAR-DOWNs and instead use the bottlenecked CPU resources to process new PATH and RESV messages. Again, this is not permitted by FIFO scheduling. Furthermore, we would always like to give priority to UPDATE messages since defering UPDATE message processing can cause spurious loss of state in the same router or in downstream routers.

In Section 4, we show by simulation that FIFO scheduling results in undesirable behavior and though simple is not a good candidate discipline for scheduling the processing of RSVP messages in scenarios where the processor utilization is high. Therefore, we propose adaptive processor scheduling schemes that try to reduce blocking and increase link utilization.

III. Adaptive Processor Scheduling for RSVP Message Processing

Instead of FIFO processing of RSVP messages, an immediate scheduling change is to use weighted round-robin scheduling. The general hierarchical weighted fair queueing scheme we use is as shown in Figure 2. The rest of this section describes the scheme that we use in detail.

Suppose we classify signaling messages into just three categories: PATH and RESV, UPDATE, and TEAR-DOWN. We can then set the weights for processing each of these message types based on some a priori traffic statistics. However, there is an immediate disadvantage to using fixed weights because it is not clear what weight should be given to UPDATEs. The UPDATE traffic increases in proportion to the number of RSVP flows that have already been established. Moreover, if UPDATE messages are lost due to insufficient weight being assigned to it, then flows already set-up are unnecessarily torn down. A possibility is to give priority to UPDATES and use weighted round-robin amongst the other classes. This has the problem that if the UPDATE traffic is very high due to a very large number of flows having been set-up, then TEAR-DOWNs are not processed adequately. Hence, flows which could have been torn down may last longer and prevent new flows

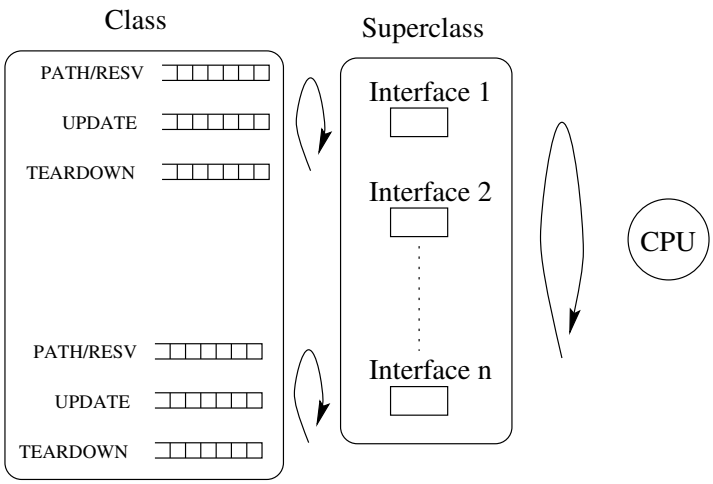


Fig. 2. Hierarchical Adaptive Weighted Round Robin Scheduling with a processor controlling multiple interfaces

from having their reservations satisified. Furthermore, a fixed weight round-robin mechanism does not take into account link utilization and knowledge of link utilization can be used to increase the probability of requests being satisfied.

The scheduling scheme we propose also uses weighted round-robin but with adaptive weights. The scheme is based on the following: (1) Updates always need a fixed fraction of the bandwidth. This fixed fraction can be made an inceasing function of the number of flows in progress but with an upper bound so that there is always a minimum bandwidth available for processing the other message types. (2) When link utilization is low, PATH and RESV messages must be given a higher weight since delaying the processing of TEAR-DOWNs does not adversely impact request blocking. Similarly, when link utilization is high, TEAR-DOWNs must be given a higher weight since processing PATH or RESV messages before TEAR-DOWNs will probably result in the bandwidth requests in those messages being denied. (3) The weights used have to be adjusted based on the average sizes of requests and tear-downs in the recent past. As an example, if reservations are small, and link utilization is low then the weight given to RESV messages have to be increased since processing of each message has a much smaller impact on the link utilization. Similarly if request sizes are large, then the weights for reservations can be scaled down. (4) Processing times for each message type, if known, should be taken into account in assigning weights. (5) Instantaneous queue-lengths by message class can be used to make weights dependent on short-term arrival rates.

Adaptation for processor per link:

For simplicity of exposition, let us consider only three classes of service. PATH and RESV messages are assigned one class, UPDATE messages are assigned to a second class, and TEAR-DOWN messages are assigned to a third class. Let the corresponding weights be denoted by w_{PR}, w_u and w_t. When a timer expires, it generates an interrupt to tear-down state. We accord high priority to these interrupts and try to process these interrupts immediately. This is because these interrupts are most likely to be generated when RSVP messages have been dropped somewhere in the network. If the processing corresponding to these interrupts is not done (such as due to them being accorded priority lower than that accorded to other tasks which can potentially saturate the processor), then the flows corresponding to these interrupts are orphaned (with no timers set to eliminate state) and they will hold on to their reserved bandwidths indefinitely causing bandwidth wastage.

The weight for updates is set to a fixed fraction of the link capacity. The only adaptation is in making this fraction proportional to the number of flows in progress.

Let PR_{ave} be the average request size expressed as a fraction of the link capacity. It is updated at every PATH or RESERVE message arrival instant in the following manner. Let $current_request_size$ be the size (expressed as a fraction of the link capacity) of the bandwidth request in the message being currently processed. Then, $PR_{ave}(new)$, the updated value of the average request size is computed from the current average, $PR_{ave}(old)$, using exponential smoothing, i.e.,

$$\begin{aligned} PR_{ave}(new) &= \alpha_{PR} * PR_{ave}(old) \\ &+ (1 - \alpha_{PR}) * current_request_size \end{aligned}$$

We choose α_{PR} so that it tracks request sizes over

time-scales of the order of a few hundred inter-arrival times. Let TD_{ave} be the average tear-down size computed, with smoothing factor α_{TD}, in a manner similar to that for PR_{ave}. Let U be the fraction of link capacity that has currently been reserved. Define $n_{PR} = (1-U)/PR_{ave}$ and $n_{TD} = U/TD_{ave}$. Then, n_{PR} denotes the number of average sized reservation requests that need to be processed to fill up the available link capacity, and n_{TD} denotes the number of average sized tear-down requests that would have to be processed to free up the utilized link capacity. (Note that the average request size and average tear-down size need not be the same since these are measured as averages over intervals typically much shorter than call holding times.)

The weight for processing PATH and RESERVEs, w_{PR}, is now chosen as $n_{PR}/(n_{PR}+n_{TD})$. Similarly, the weight for processing TEAR DOWNS, w_{TD}, is chosen as $n_{TD}/(n_{PR}+n_{TD})$. Here, there is an implicit assumption that the processing times for all the three message types are the same. Since that is not the case in practice, we scale the weights in proportion to the processing times (also these weights must be rescaled to account for the capacity allocated to UPDATEs). This basic scheme that we use has the property that the weight accorded to PATH and RESERVE message processing is increased when the request sizes are small and when the link utilization is low, i.e., $1-U$ is large. Therefore, at low utilization the system tries to increase the rate at which PATH and RESERVEs are processed since delaying TEAR-DOWNs does not affect link utilization. Similarly, when the link utilization is very high, the system processes TEAR-DOWNs at a faster rate and thereby increase the chance that the next PATH or RESERVE which is processed has a higher likelyhood of not being blocked.

One drawback of this scheme is that we do not take arrival rates into account. If we want to take arrival rates into account, an implicit way of doing this is to use average queue lengths as well in computing weights. An simple scheme is to add to n_{PR} the average queue length for this class (computed over the same time scale) and to add to n_{TD} the average queue length for TEAR DOWNs. The weights are now computed in the same manner as above. Simulation results for these cases are presented in the next section.

Processor scheduling for multiple links per processor:

The extention of the above schemes for multiple links per processor can be done by using a two-level hierarchical weighted round-robin scheme instead of a single level weighted round-robin scheme. At the higher level, there a super-class is associated with each link that the processor controls. At the lower level, for each link super-class we have the same service classes as in the processor per link scheme (see Figure 2). We first compute the weights for the lower class associated with each super-class in the same manner as described above. Let $1, 2, \ldots n$ be the links associated with a processor and let the weights for the level 1 class of link i be denoted by w^i_{PR} and w^i_{TD}. For each super-class i, we then compute the super-class weight W_i as

$$W_i = \frac{w^i_{PR} + w^i_{TD}}{\sum_{j=1}^{n} w^j_{PR} + \sum_{j=1}^{n} w^j_{TD}} \quad (1)$$

With this scheme, if any link is highly utilized (hence experiencing a high blocking) or lightly utilized (hence wasting bandwidth which could be used if the processor were not a bottleneck) then one of its lower class weights is high. If other links are not in these extreme situations, then their weights are not as high. Hence, the super-class weights tends to give a higher share of the processor to links which are at the extremes of utilization and have a backlog of messages to be processed. This enforces a certain amount of fairness amongst the links. Note that if one link has a large number of small requests while another link has a small number of large requests, then the former link gets a higher super-class weight. Also, note that link speeds can be widely differing since during weight computation, the link speeds and request sizes are normalized.

The update weight for each super-class can be obtained by calculating the update weight for each link, w^i_u, as before and then computing the UPDATE super-class weight for link i as

$$W^i_u = \frac{w^i_u}{\sum_{j=1}^{n} w^j_u} \quad (2)$$

IV. Performance Studies

The network configuration simulated is either a single router or two routers in tandem attached to a large number of sources and receivers that generate RSVP messages. The characteristics of reservations request vary in the different simulations. We assumed that the greatest number of reservation requests are small requests e.g. for audio conferences (about 64 kb/sec). But some sources generate requests for much larger bandwidth. These requests may be thought of as coming from video servers or video conferencing systems. To simulate other tasks that the processor may do, we simulate extra tasks such as routing table recalculations. The relative service times for RSVP message processing that we use in the simualtions were obtained using

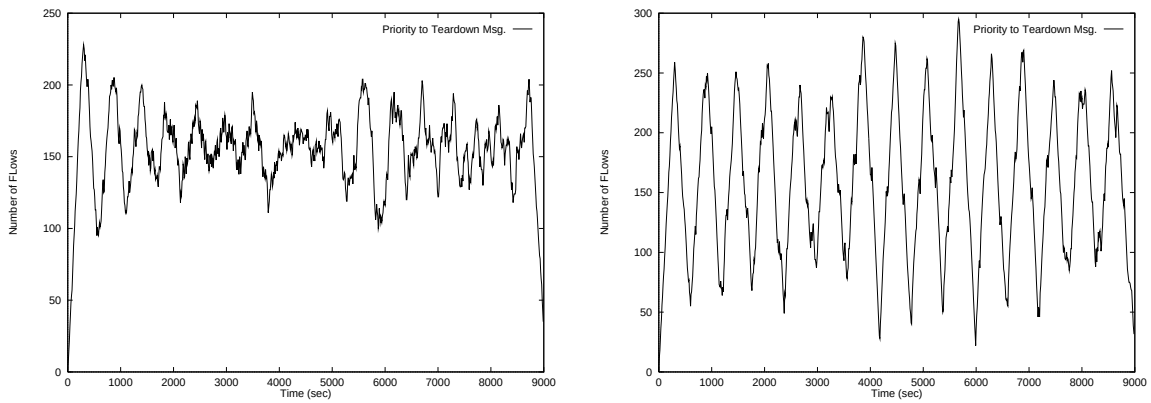


Fig. 3. Number of flows low/high CPU load without router timeouts

measurements on an existing distribution of RSVP software.

A. FIFO and Priority Scheduling

We first simulated what a priori looks like a simple and useful scheduling scheme. We use FIFO processing for PATH, RESV, TEAR-DOWN and UPDATE messages. We process state tear-downs due to timer expiration with high priority (see Figure 1). This is because if timer expirations share the same FIFO processing with other messages, some timer expirations may never get processed when the system is overloaded. This leads to orphaned flows and reserved bandwidth which is never released leading to loss of link capacity. For simplicity, we made the request size to be the same for all reservation messages (i.e., models a scenario with many flows of same type).

Figure 3 shows the link utilization (expressed in number of flows since requests are of constant size) as a function of time for two levels of processor utilization denoted by low and high. For "low" utilization, the load offered is below capacity without tear-downs. Adding the tear-down load causes processor saturation. For high utilization, the processor is always saturated. This models periods of CPU overload such as those caused by routing transients. During these overloaded periods, good scheduling should make best use of CPU resources so as to make the system behave robustly and prevent undesirable oscillations in link usage.

For the plots in Figure 3, the refresh timer is set to very large values so that it does not expire. Nevertheless, we see that the pattern of link utilization is very unsatisfcatory. Initially, when the number of flows is not very large, the processor load is low enough that many flows are succesfully established. Note that to establish a flow, both its PATH and RESV message must be succesfully processed. As the number of flows increases, the UPDATE traffic increases proportionally and hence increases processor load. Still the number of established flows keeps increasing. Next, some of the flows which have been established start generating TEAR-DOWN messages since their holding times have elapsed. Since we give priority to TEAR-DOWNs, the number of flows in progress decreases. When the number of flows has decreased sufficiently, the arrival rate of TEAR-DOWNs decreases sufficiently that new flows get established faster than the rate at which flows are being torn down. Hence, the number of flows increases again. After a delay, equal to the holding time TEAR-DOWNs are generated again and the number of flows (and hence link utilization) goes down. This oscillation is more pronounced at high offered loads. However, with the refresh timer virtually turned off, only certain choice of traffic and service time parameters produces these strong oscillations. If the refresh timer is set to a reasonable value, then oscillations happen very often as explained below.

Figure 4 shows results for a scenario similar to that in Figure 3 except that the refresh timer is not disabled. We see extreme oscillation in both cases with the number of flows in progress periodically becoming nearly zero. This can be explained as follows. As more and more flows are set up, the processor load keeps increasing since the UPDATE traffic increases in proportion to the number of flows in progress. The number of flows in progress increases at the rate of RESERVE message processing since the processing of each RESERVE message sets up a new flow unless the link saturates first and causes blocking (which does not happen in the example shown). This increasing load eventually causes the processor to saturate and the FIFO queue builds up with PATH, RESERVE and UPDATE messages waiting to be processed. Since the processor is saturated and the load increases with each processed RESERVE mes-

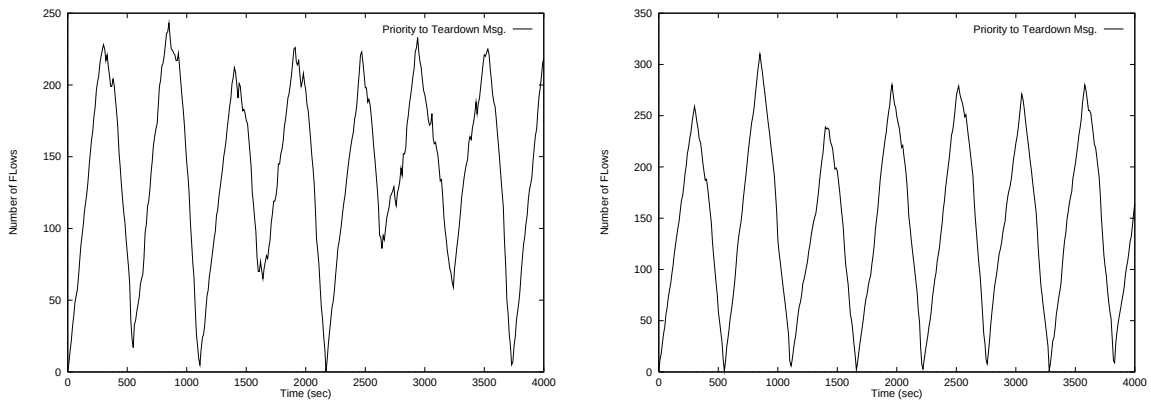


Fig. 4. Number of flows low/high CPU load and router timeouts

sage, the queue eventually overflows. Newly arriving UPDATE messages are dropped and eventually timer expirations happen for flows whose UPDATE messages have been dropped. These cause the termination of all flows for which UPDATEs have been lost causing the link utilization to continually decrease (since timer expirations which have priority are processed faster than new flow set-up). During the time that these timer expirations are being processed, the low priority FIFO queues does not receive any service. Hence, more UPDATEs (as well as other arriving messages) are lost or delayed. This causes even more timer expirations which further starve the low priority queue of access to the CPU. Flows are constantly torn down with very little chance of new flows being succesfully setup. When sufficient flows have been torn-down, the processing load decreases and new flows can be setup causing link utilization to increase. This process repeats causing oscillations in link usage. The oscillatory behavior is caused due to FIFO processing of UPDATEs, priority processing of timer expirations (to prevent bandwidth wastage due to orphaned flows), and due to processor saturation happening before link saturation (as can happen if request sizes are small in comparision with link capacity and the processor is shared amongst multiple links or has to perform multiple functions). The three combine to cause a postive feedback effect leading to oscillations.

B. Performance of Adaptive Scheduling

Figure 5 shows simulation results when the processor is not completely overloaded. The results shown are for FIFO processing of all messages, fixed weighted round-robin (with weights chosen to be proportional to service times and labeled FWRR in the plots), and adaptive weighted round-robin schemes (labeled simply as WRR in the plots). All requests have sizes between 1 and 5 kbits/s. The interarrival times of the requests are exponentially distributed.

In Figure 5, the number of flows in progress when each of these schemes is used is plotted as a function of time. Since the processor load is moderate, the effect of scheduling is not very pronounced and both FIFO and adaptive weighted round robin have similar behavior with the number of flows being slightly larger for the latter. However, for FIFO there are more spurious tear-downs due to UPDATE messages not being processed in time. These spurious tear-downs are smaller for fixed round-robin and lowest for adaptive round-robin showing the usefulness of adaptive processor scheduling. The fixed weighted round robin has the worst behavior in terms of the number of flows because the static weights are not optimized for the offered load.

Figure 6 shows the same setup as that for Figure 5 but with a high fraction of the CPU usage being due to route processing. Hence the CPU is saturated even though the offered load due to RSVP messages is the same. When the CPU load increases the advantage of our mechanisms are clearly evident, in terms of number of flows in progress and in terms of reserved bandwidth. The advantage of the adaptive weight scheme is obvious from the figure. Adaptive WRR is very effective in the two extreme cases where reserved bandwidth is very low (we accept more reservations) and when the reserved bandwidth reaches the maximum link bandwidth. In this case, we maximize the number of accepted calls since adaptive WRR frees the bandwidth of closed connections faster than FWRR (Figure 6).

In the extreme cases, adaptive WRR may lead to a higher number of router timeouts than FWRR. Suppose the link is utilized very little. Then TEAR-DOWN weight decreases to a very small value and most of the processor capacity will be used to setup new flow reservations. The low rate of TEAR-DOWN processing can cause the refresh timer to expire and tear-down the

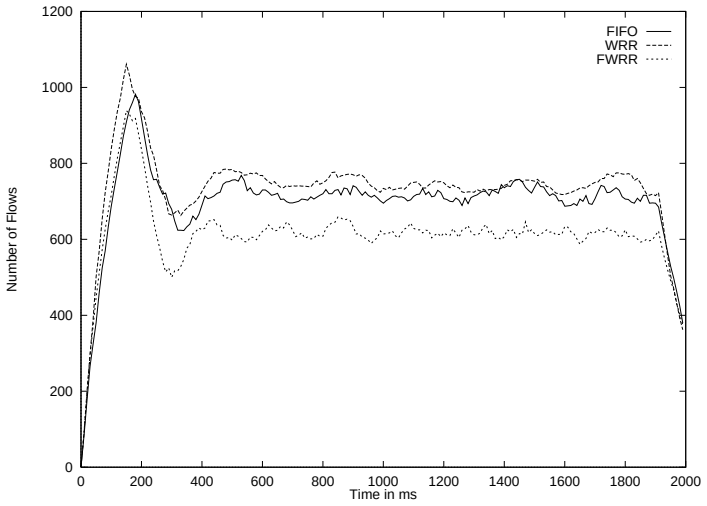


Fig. 5. Number of flows with a medium CPU Load

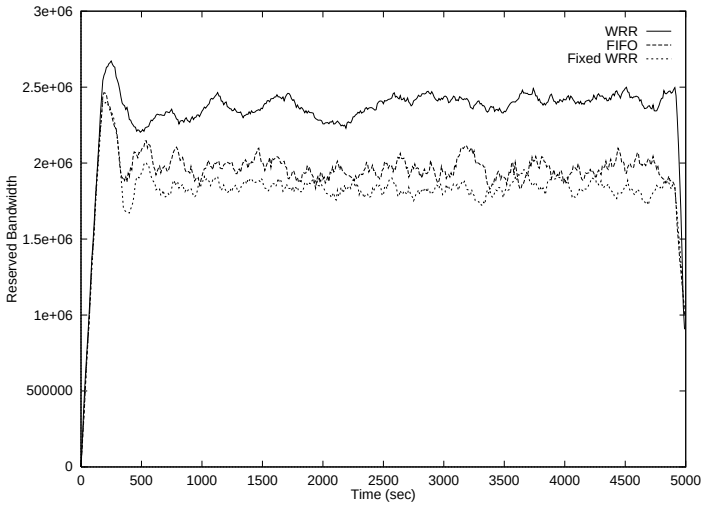


Fig. 6. Reserved bandwidth with a high CPU Load

flow. This is by choice. Since the processor is bottlenecked, we must decide whether to allocate processor resources to handle explicit TEAR-DOWNs or to allocate it to handle PATH or RESERVEs. Processing a TEAR-DOWN only makes more bandwidth available. Since the utilization is low, the system is not bandwidth constrained and so it is better to use scarce processor resources to process PATH or RESERVEs than for processing TEAR-DOWNs. Hence, TEAR-DOWN processing can be delayed as much as is allowed. By doing so, we just increase the holding time by the timeout period for the flow for which the TEAR-DOWN was not processed in time. This does not adversely affect performance since this happens only when link utilization is low.

Figure 7 shows the number of flows in the first router for a configuration where RSVP sources are connected to the first router which is connected by a single link to a second router to which are connected RSVP receivers. The processors in both routers are saturated. Both routers have a larger number of flows in progress (though only the number of flows in one of the routers is shown in the figure) under the adaptive weighted round robin scheme than with the FIFO scheme. Clearly, proper processor scheduling can improve utilization of available link bandwidth. In the example in Figure 7, there is approximately a 40% increase in the number of flows in progress.

Figure 8 shows that adaptive weighted round robin can prevent oscillations in link utilization (shown in the figure in terms of the number of flows in progress). This is because UPDATEs are always processed at a rate pro-

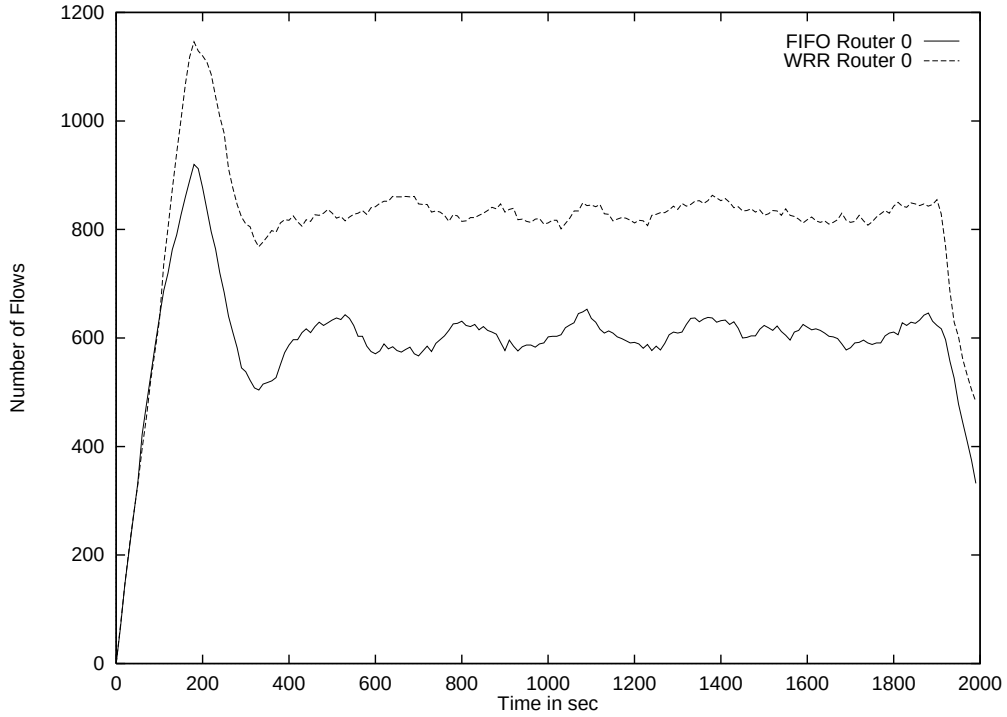


Fig. 7. Number of flows in router 1 with a high CPU Load and 2 routers in tandem configuration

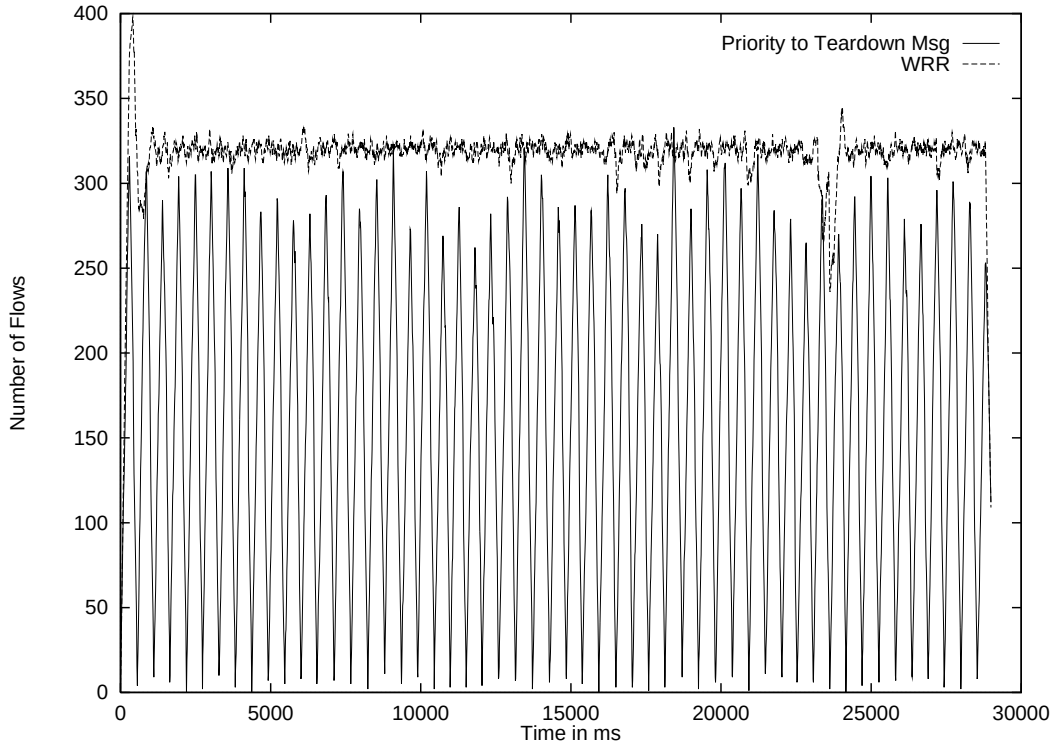


Fig. 8. Comparison of link utilization for FIFO combined with priority to TEAR-DOWNS with adaptive weighted round robin

portional to the number of flows in progress and are not queued behind PATH and RESERVE messages wait-

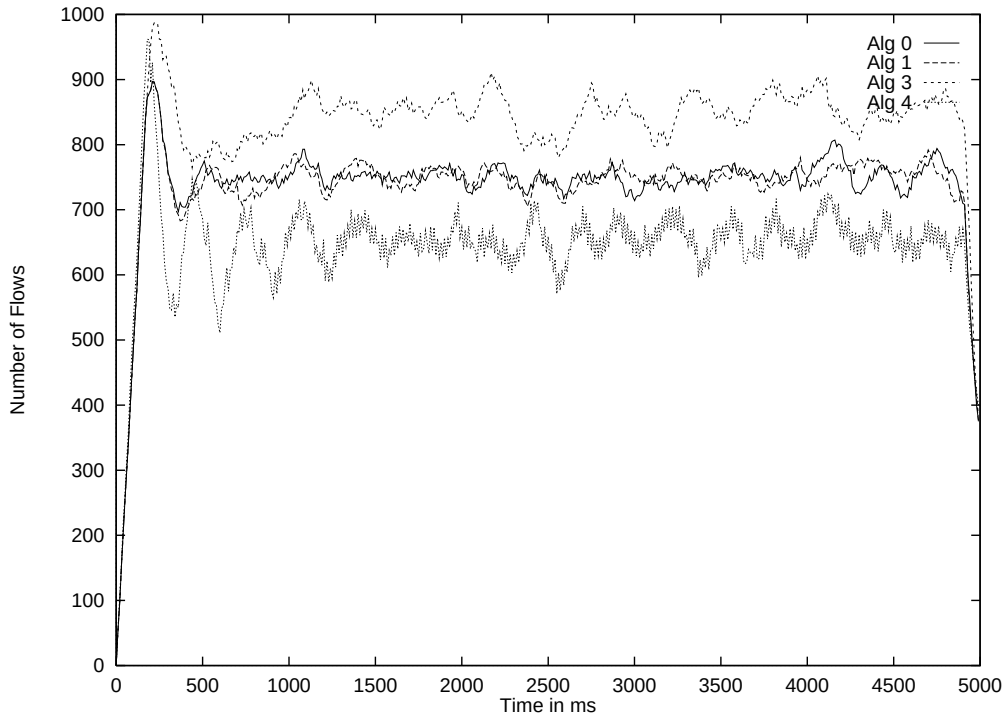


Fig. 9. Number of Flows for the algorithms with a high CPU Load

ing to be processed. Unless the UPDATE messages by themselves can saturate the processor (or if the UPDATE weights are set too low), UPDATE messages are not dropped or delayed long enough to cause timer expiration. The positive feedback effect, evident in FIFO processing, due to delayed UPDATEs causing timer expirations which further delay other updates and cause more timer expirations is not likely to happen since UPDATEs are seldom delayed long enough to cause timer expirations. Hence, oscillations in link utilization are reduced drastically in comparison to FIFO scheduling where the oscillations are so large that the number of flows in progress periodically becomes almost zero.

C. Variations to the basic adaptation scheme

We studied the impact of 4 different adaptive weight calculations.

- In the basic scheme, we calculated the weights as described in the simplest version in the previous sections.
- The second scheme, includes the queue content to the above calculation and so biases the basic scheme to message classes with longer queues.
- The third scheme improves on the second by multiplying the weights by the normalized work accumulated in the corresponding message queues.
- The fourth scheme determines the weights only according to the queue-length of each message class.

Figure 9 shows the resulting behaviour of the four algorithms (they are marked alg0, alg1, alg3, and alg4 respectively in the figure). The plots shows the number of flows in progress when the different algorithms are used and when the processor is saturated due to background task and RSVP message processing. We do not plot results for FIFO processing since we have already seen that FIFO performs poorly in comparison with basic adaptive scheme.

Since the basic adaptive scheme does not take queue lengths into account, it is not very responsive, when computing weights, to bursts of PATH, RESERVE or TEAR-DOWN requests that may arrive. Instead, the weights are set purely based on link-state, and average request sizes. If the link is not in the extreme ends of utilization, the weights change slowly (as they adjust to track the average request sizes which may change). Making the weights to be a function of the queue length permits the weight to change in response to increased or decreased arrivals of a particular message class. For the second scheme, we calculated the weights w_{PR} and w_t as follows: To determine w_{PR}, we first calculate n_{PR} as before. Then, calculate a new n^q_{PR} as a weighted sum of n_{PR} and the queue length of PATH and RESERVES. We varied the weight given to the queue length in the

range 0.5 to 1. Then w_{PR} is calculated as in the basic scheme except that we use n^q_{PR} instead of n_{PR}. The TEAR-DOWN weight w_t is calculated in a similar manner.

Since using queue lengths does not account for service times, the third scheme instead of using queue lengths uses accumulated work in each queue. We calculate the weights w_{PR} and w_t as in the basic scheme. We then multiply these weights by the normalized work in each queue. This should account for differing arrival patterns much better than the second scheme and this can be seen from Figure 9. The plots show that a combination of link utilization, request sizes (as in the basic scheme) combined with work in queue informaton gives the bests results. The fourth scheme which only uses queue-length information is unable to adapt to link-state and hence has the poorest performance. The basic scheme and the basic scheme with queue lengths added have approximately the same performance.

V. Concluding Remarks

Control processor scheduling is an important issue that can impact both the performance and robustness of networks. Yet processor scheduling in routers has not been studied as intensely as link scheduling has been. Considering the case of processor scheduling for RSVP message processing in routers, we showed that simplistic processor scheduling can cause poor link usage and cause blocking even when link capacity is available. We considered both the case of a processor per link and processor per group of links. We showed that FIFO, or FIFO combined with simple priorities are not adequate for satisfactory link utilization in situations where the control processor is somewhat highly loaded. We then proposed the use of simple adaptive hierarchical weighted round-robin scheduling to make best use of the links when the processor is loaded. We proposed adaptation schemes using link state, request size, and queue informaton. We demonstrated the advantages of adapation by simulation of different variants of the adaptive scheduling scheme. An area of future work is to include into the scheduling schemes the processing of other protocols and tasks that the control processor may perform (such as processing multicast joins/leaves) so as to improve performance and robustness under transient processor overload. Also, analytical explanation of the oscillations in link utilization, and study of network wide effects remain areas for future study.

References

[1] R. Braden et. al., "RSVP: A New Resource ReSerVation Protocol", *IEEE Network* September 1993. See also http://www.ietf.org/html.charters/rsvp-charter.html.

[2] A. Campbell, G. Coulson, "QoS Adaptive Transports: Delivering Scalable Media to the Deskto", *IEEE Multimedia*, March 1997.

[3] D. Clark, D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proceedings of ACM SIGCOMM '90.*

[4] M. Degermark et. al., "Small Forwarding Tables for Fast Routing Lookups", *Proceedings of ACM SIGCOMM '97.*

[5] Integrated Services Working Group, http://www.ietf.org/html.charters/intserv-charter.html

[6] V. P. Kumar, T. V. Lakshman, D. Stiliadis, "Beyond Best Effort: Router Architectures for Tomorrow's Internet", *IEEE Communications Magazine*, pp. 152-164, May 1998.

[7] S. Lin, "A Simulation Study of IP Switching", *Proceedings of ACM SIGCOMM '97.*

[8] K. Nichols et. al., "Differentiated Services Operational Model and Definitions", *Internet Draft*, February 1998.

[9] J. Wroclaswski, S. Shenker, "Specification of the Controlled Load Network Element Service", *Internet Draft*, November 1996.

YESSIR: A Simple Reservation Mechanism for the Internet

Ping Pan and Henning Schulzrinne

Abstract—

RSVP has been designed to support resource reservation in the Internet. However, it has two major problems: complexity and scalability. The former results in large message processing overhead at end systems and routers, and inefficient firewall processing at the edge of the network. The latter implies that in a backbone environment, the amount of bandwidth consumed by refresh messages and the storage space that is needed to support a large number of flows at a router are too large. We have developed a new reservation mechanism that simplifies the process of establishing reserved flows while preserving many unique features introduced by RSVP. Simplicity is measured in terms of control message processing, data packet processing, and user-level flexibility. Features such as robustness, advertising network service availability and resource sharing among multiple senders are also supported in the proposal.

The proposed mechanism, YESSIR (YEt another Sender Session Internet Reservations) generates reservation requests by senders to reduce the processing overhead, builds on top of RTCP, uses *soft state* to maintain reservation states, supports shared reservation and associated flow merging and is compatible with the IETF Integrated Services models.

YESSIR extends the all-or-nothing reservation model to support partial reservations that improve over the duration of the session.

To address the scalability issue, we investigate the possibility of using YESSIR for per-stream reservation and RSVP for aggregate reservation.

I. Introduction

A. Background

Continuous media applications such as Internet telephony, teleconferencing, interactive multimedia games and media-on-demand have become increasingly popular in the Internet. There are several driving forces behind the growth of continuous media applications: The deployment of IP multicast in the Internet via the MBONE overlay network has provided an important platform for testing and deploying various audio and video applications. Also, the Real-Time Transport Protocol (RTP) [1] facilitated the development of interoperable applications, which have become available across a wide range of platforms. Finally, end systems have become capable of generating and rendering highly-compressed multimedia content.

However, as the usage of the Internet has grown, packet loss, delay variations and lack of bandwidth [2] have made the current Internet unsuitable for widespread delivery of *predictably* high-quality continuous media services. While only sufficient network capacity can correct these problems, it is often desirable to give improved service to certain classes of applications[1].

We can distinguish per-packet and per-flow approaches to providing differentiated QOS:

P. Pan is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. pan@watson.ibm.com.

H. Schulzrinne is with the Department of Computer Science and the Department of Electrical Engineering, Columbia University, New York, NY 10027. schulzrinne@cs.columbia.edu.

[1] Particularly, if the users of these applications may be willing to pay for improved predictability in quality-of-service. Note that even with network bandwidth that is statistically sufficient, many continuous-media applications demand *consistent* service throughout the lifetime, for example, of a phone call or video-on-demand session, unaffected by traffic bursts of other applications.

Per-packet: Information necessary to obtain differentiated service is carried inside each data packet. In IPv4, for example, the type-of-service (TOS) byte may be used or priority may be accorded to certain port numbers. Routers do not need to maintain state beyond a rough classifier list that can be considered static; there is no control protocol overhead.
This approach cannot guarantee a certain QOS, since the amount of traffic injected with a particular TOS value, for example, cannot be bounded. Thus, the per-packet approach is subject to intentional and unintentional denial-of-service attacks.
Per-flow: Resources for a set of packets distinguished by source or destination addresses and port numbers ("flows") are reserved ahead of time, with reservations established and torn down dynamically.

Both approaches can be combined; for example, a resource reservation protocol can limit the amount of traffic with a particular TOS value.

We will focus primarily on per-flow resource reservation. Currently, RSVP [3], [4] is the reservation protocol of record in the Internet. Unfortunately, its generality implies a cost in complexity, as discussed in Section I-C. We set out to address some of these issues.

B. Overview

This paper describes and evaluates YESSIR (YEt another Sender Session Internet Reservation[2]), an in-band, sender-based resource-reservation protocol based on RTP that offers significantly lower code and run-time complexity than RSVP.

YESSIR is motivated by the observation that a large fraction of the applications that require guaranteed quality-of-service are continuous media applications and that a substantial fraction of these either use or will use the Real-Time Transport protocol (RTP) to deliver their data. YESSIR and RSVP can operate side-by-side in the same network, without affecting the certainty of guarantees offered to applications.

In this section, we will briefly explain some of the design decisions that introduce complexity into RSVP and describe features of RTP relevant to YESSIR. The rest of paper is organized as follows: In Section II, we will outline the design goals of YESSIR. The YESSIR reservation mechanism will be detailed in Section III. Section IV reports on our experimental implementation and its performance. Several open issues (including a possible solution to the scaling problem) are discussed in Section V. We summarize in Section VII.

C. RSVP Complexity Issues

Initially, RSVP was perceived as a light-weight reservation protocol, in comparison, for example, with ATM signaling pro-

[2] The name reflects the proper attitude of a resource reservation protocol in a well-designed network.

tocols such as Q.2931. However, as implementations are weighing in at between 10,000 and 30,000 lines of code, it seems appropriate to review some of the design features that contribute to the complexity:

Receiver orientation: In RSVP, receivers make reservations, based on information provided by senders. This allows individual receivers within a single multicast group to request different levels of service guarantees, including none. It seems likely, however, that receivers will simply request whatever traffic bandwidth the sender has indicated, either through RSVP PATH messages or some other session initiation protocol [5].

The separation of reservation and path-finding messages for receiver-oriented reservation mechanisms imposes additional processing and protocol complexity.

Receiver diversity: At least for bandwidth diversity, reservations are an inappropriate means to distinguish classes of receivers. Bandwidth diversity could only be accomplished by "thinning" flows, i.e., dropping packets, as flows reach parts of the network endowed with less bandwidth. However, random packet dropping will quickly degrade most audio and video encodings due to their use of prediction across packet boundaries. Other mechanisms, such as layered multicast [6], were found to be superior to support diverse receiver populations.

RSVP receivers can request different values of queuing delay as part of their resource reservation. However, in high-speed wide-area networks, queuing delay is dominated by propagation delays; also, in popular scheduling disciplines such as weighted fair queuing, queuing delay for an individual flow can only be improved by allocating more than its "true" bandwidth, thus incurring allocation inefficiency, particularly if reserved flows constitute a large fraction of the total link capacity.

Receiver diversity and receiver orientation require that nodes merge incoming reservations into a single reservation setting aside the least upper bound of the requests of all downstream receivers. Flow merging also introduces the need for "blockade" state to prevent so-called killer reservation (see Section III-E).

Flow management: Because reservation requests are generated from downstream, keeping track of next-hops can become difficult and CPU intensive, particularly in multicast-capable non-broadcast multiple access (NBMA) networks such as ATM subnets.

Application modification: Since RSVP is an out-of-band protocol, applications need to be modified, either to take advantage of kernel-level support for RSVP or to convey their resource requirements to some external agent that makes reservations on their behalf. Both solutions incur complexity.

Firewall complexity: Firewalls are also complicated by RSVP's out-of-band nature. First, they need to parse and pass RSVP requests; they also need to correlate between the flows (session-sender pairs) described in RSVP messages and the actual data streams (Fig. 1).

The reservation mechanism introduced here avoids these problems.

D. RTP Features Useful for Resource Reservation

RTP [1] has been designed to provide end-to-end delivery services for data with real-time characteristics. Although protocol-independent, applications normally run RTP on top of UDP to make the use of its multiplexing and checksum services. It has been widely implemented on multimedia systems across all operating systems and is part of the ITU H.323 recommendation for conferencing and Internet telephony. Examples include *vic* [7], *vat* [8], *rat* [9] and *NeVoT* [10] for teleconferencing over the MBONE, *NetMeeting* from Microsoft and conferencing tools from Netscape.

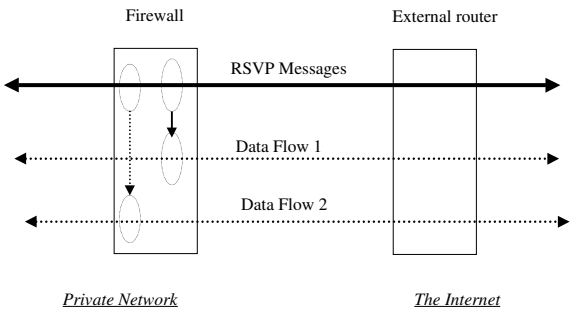


Fig. 1. RSVP overhead at a firewall

Although RTP was not intended as a resource reservation protocol, resource reservation can benefit from the following RTP features:

In-band signaling: RTP defines two types of packets: RTP for data transport and RTCP for control. Each RTP session consists of one RTP data stream and one corresponding RTCP stream, originated by one or more participants. When carried over UDP, data and control packets use adjacent port numbers, so that a router or firewall can easily map from a control stream to the corresponding data stream.

Periodic sender/receiver notification: Senders and receivers periodically send RTCP packets containing reports to the multicast group. Data senders distribute sender reports (SRs) that indicate, inter alia, the number of bytes and packets transmitted since the last report and information allowing the estimation of round-trip times. Data receivers include receiver reports (RRs) that indicate packet loss and delay statistics, among others.

By evaluating these reports, all participating members have knowledge of traffic characteristics, network congestion and session membership. Routers can deduce the resource requirements of a session from these reports, as will be discussed below.

The period between reports has a minimum of 5 seconds and scales with the number of participants, keeping the RTCP session control overhead limited to no more than 5% of the data bandwidth. Senders are allocated at least 25% of the session control bandwidth.

Embedded in applications: RTP is typically implemented as part of the application. As will be shown in Section III-B, even an RTP application that runs the current version of RTCP can be used to initiate resource requests. No kernel modifications,

beyond the support of IP router alert options (Section I-E), are needed.

E. IP Router Alert Option

The IP router alert option [11], [12] alerts transit routers to more closely examine the contents of an IP packet. In other words, routers can intercept packets not addressed to them directly, with little performance impact. For example, RSVP PATH messages are carried in IP packets that include the router alert option. Thus, even though RSVP PATH messages are addressed to end systems, PATH messages are intercepted and processed by all transit routers. We make use of router alert options to mark RTCP sender report for YESSIR processing.

II. Design Objectives

YESSIR offers an alternative, light-weight approach to resource reservation in the Internet, using RTCP sender reports to reserve resources in the network.

It has the following properties:

Sender-initiated reservation: As motivated earlier, we anticipate that many applications cannot make full use of the benefits of receiver-initiated reservations. Sender-initiated reservation may also fit better with policy and billing, as the number of senders making reservations is likely to be much smaller than the number of receivers. In many existing systems, such as cable television, the cost of "resource reservation" is bundled with the cost of content, simplifying billing. (Also, a provider of pay-per-view services would likely want to avoid the case where subscribers pay, fail to reserve resources and then ask for their money back since the quality was unacceptable.) In the absence of an Internet-wide authentication and cross-provider billing service, it is far easier for the relatively small number of large-scale content providers, residing at known network addresses, to arrange for payment with major backbone providers than individual subscribers.

Note that RSVP could also be modified to have PATH messages initiate reservations, so that the benefits of sender orientation for some applications do not depend on the use of YESSIR.

Robustness and soft-state: Similar to RSVP and PIM [13], routers maintain reservation states as *soft state*, i.e., reservations disappear by themselves if not refreshed periodically. In YESSIR, this avoids orphan reservations and adapts quickly to routing changes. As in RSVP, an explicit teardown mechanism using RTCP BYE packet avoids holding reservations for a number of soft-state refresh intervals after the requesting application has terminated.

Allow partial reservations: The function of resource reservations is to protect existing streams against disruption by other streams that arrive later.

In "classical" reservation systems, reservations are either made or denied end-to-end. Depending on the system, the requestor can always either ask again, at some cost to the network if done too often ("redialing"). Some systems, including RSVP, also allow to specify a range of resource requests to increase the likelihood of success, however, this can cause low-bandwidth regions to experience high packet losses despite reservations.

We propose an additional reservation model, that of a partial reservation, where some fraction of the links have resource protection for a particular flow, others may not. On links without reservation, traffic is carried on a best-effort basis and the resource reservation request continues downstream towards the receivers. Since YESSIR is a soft-state protocol that resends reservation requests periodically, a flow can acquire a reservation on a link when another flow terminates, without having to retry at the application layer. The user can decide whether to put up with a partially successful reservation and hope that more links will be added as the session continues or cancel the session. For a live presentation, where inserting an end-to-end reservation means missing the event, a user may well decide that the prospect of improving reservation fortunes may be better than not listening at all or foregoing all resource reservations[3]. YESSIR supports both end-to-end and partial reservations.

Provide different reservation styles: YESSIR supports *individual* and *shared* reservation styles. Individual reservations are made separately for each sender, whereas shared reservations allocate resources that can be used by all senders in an RTP session.

Individual reservations are called for when all senders are active simultaneously, e.g., for distribution of participant video in a conference, while shared reservations are appropriate where several senders alternate, e.g., for audio in a conference. (Shared reservations also avoid the problem that a new speaker may not be able to acquire a reservation; they can re-use the existing reservation of the previous speaker.)

These styles are simplified versions of the fixed filter and wildcard filter reservations in RSVP. Note that the shared reservation styles, one of the distinguishing features of RSVP, does not depend on receiver orientation. YESSIR handles the shared reservation style from the sender's direction, while RSVP supports shared reservation (shared-explicit and wildcard-filter styles) from the receiver's direction.

Low protocol and processing overhead: Rather than defining another signaling protocol, YESSIR messages are transported by RTCP. Given that RTP is in-band signaling and its data and control packets are tightly coupled, updating packet classifiers and firewalls can be simplified. YESSIR uses one message to set up a reservation. Its processing algorithm is very simple, as we will illustrate in Section III-B.

Interoperable with RTP and the IntServ models: YESSIR messages are piggybacked in RTCP. The operation of existing RTP functions at end systems is not affected at all. YESSIR can describe the traffic flows in terms of the service models [14], [15] that have been specified in the IETF IntServ working group.

Provide link resource advertising function: The purpose of making link-level resource reservation is to meet end-to-end application requirements. To that end, YESSIR is able to query and carry collected network resource information to the end systems.

[3] Partial reservations can lead to fragmentation, where a large number of flows all have partial reservations, with unacceptable quality. This aspect is the subject of current work. The soft-state mechanisms also gives a slight advantage to high-bandwidth flows or flows with few senders, as they may get to send RTCP requests more frequently.

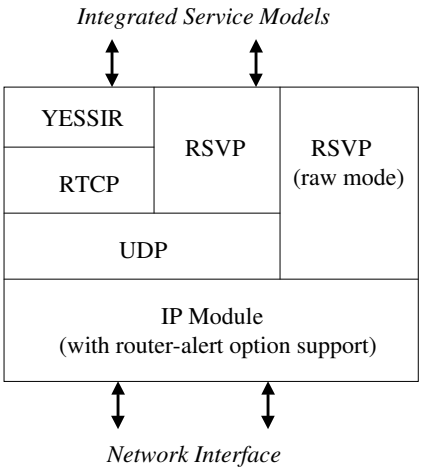


Fig. 2. Protocol relationships

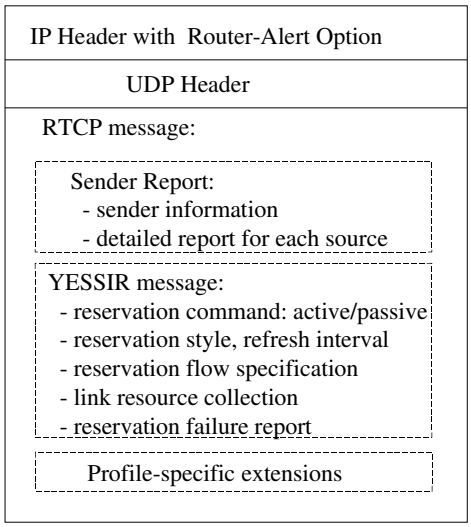


Fig. 3. YESSIR message format

III. YESSIR Operation

A. Protocol Overview

YESSIR reservation messages consist of RTCP sender-report messages, possibly enhanced by additional YESSIR-specific data, carried in IP packets with router-alert options. The placement and relationships to other protocols are shown in Figure 2.

Reservation requests generated by senders are intercepted and processed by those routers that support the router-alert IP option. Routers that do not support the option or YESSIR forward the RTCP message unaltered to the next hop. End systems ignore the router alert option. Thus, YESSIR can be deployed incrementally and without affecting the behavior of end systems.

An optional reservation extension for RTCP is defined. It is piggybacked at the end of an RTCP report (SR or RR), as shown in Figure 3. The YESSIR extension consists of a generic fragment, a flowspec fragment, an optional network monitoring fragment and an optional reservation error fragment. The generic fragment instructs the router as to the desired reservation style (individual or shared), the soft-state refresh interval and whether to make partial reservations. The flowspec fragment provides the router with the necessary information as to whether to admit the flow and what resources to set aside. The flow specification can be in several formats. The optional network monitoring fragment stores link resource information. If it is present in a request, every router along the path updates the link information in the fragment. Currently, we use the ADSPEC format [16] defined by the IETF IntServ working group. Routers where reservation requests fail indicate the reason for failure in the optional reservation error fragment. The fragment is used to collect error information that will allow end systems and network administrators to diagnose reservation failure inside the network.

B. Outline of Operation

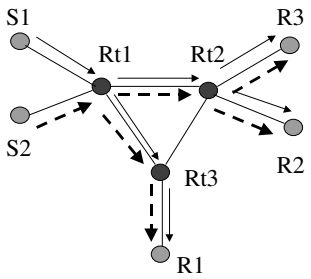
Senders periodically multicast RTCP sender reports (SRs) to all members of the multicast group (or the other party, if unicast). Sender reports contain transmission and reception statistics. Routers may either use the transmission statistics or additional YESSIR flowspecs and other elements.

As shown in Figure 3, YESSIR may insert reservation information into SR, however, YESSIR can also operate without any additional information beyond what is already contained in RTCP sender reports. When an RTCP SR is received by a router, the router will attempt to make a resource reservation according to the information specified in the message.

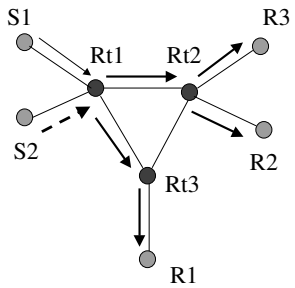
If a reservation request cannot be granted at a router, the SR packet will continue to be forwarded to the next hop(s). The router has the option of inserting reservation failure information into the SR. As a part of RTCP receiver reports (RR), the receivers will provide failure information to the senders. Based on RRs received, senders can either drop the session, or lower the reservation request and transmitted bandwidth.

If a reservation request is accepted by a router, the corresponding RTP data stream information is added into the packet classifier, and the router's scheduler is updated to support the new stream.

Instead of basing reservations on flowspecs, YESSIR can also operate in a measurement mode. Measurement mode makes use of the fact that RTCP SRs contain a byte count and a timestamp. If the first RTCP packet for a session does not contain a flowspec, the router simply records the timestamp and byte count, but does not make a reservation. If a second packet for the same session comes along, the router computes the difference in time stamps and byte counts and thus computes an estimated rate. It then establishes reservations for this measured bandwidth, updated as new RTCP packets arrive. Compared to other measurement-based admission controls [17], this frees the router from the burden to count packets and estimate rates. Another measurement method, which we have not explored in detail, simply has the end system mark an RTP data packet ev-



(a) *Distinct Reservation style*: Reservations for S1are shown as in solid line; S2, in dotted line.



(b) *Shared Reservation style*: At Rt1, after flow merging between reservation for S1 (solid line) and S2 (dotted line), a single reservation (thicker line) is made to Rt2 and Rt3.

Fig. 4. Different reservation styles (S1 and S2 are senders, R1, R2 and R3 are receivers in a single multicast RTP session; Rt1, Rt2 and Rt3 are routers)

ery so often with an IP router alert option. Each RTP packet contains a payload type indication, which indicates the media encoding (e.g., G.711-encoded voice). For many low bit rate codecs, the payload type is associated with a fixed rate (e.g., 64 kb/s for G.711), so that the router can make reservations based on that information alone. This mode, while less general and flexible than the current YESSIR mode, has the advantage of trivial header parsing and fixed refresh intervals. (It also incurs the danger of increased packet delay variation and packet reordering since some RTP packets would traverse a routers "slow path", while most would not.)

Reservation states in each router are maintained as soft-state. The reservation is automatically removed if no RTCP SR is received within several consecutive refresh intervals. To reduce the processing burden at routers, instead of having routers to initiate refresh messages, RTP senders periodically generate RTCP SR's with IP router-alert option to refresh reservations. Compare with hop-by-hop refresh mechanism that has been used in RSVP, the end-to-end refresh mechanism in YESSIR reduces both routers' processing cost (as shown in Section IV-B) and protocol overhead. For example, for a single-sender session, the sender uses merely 1.25% of the session bandwidth for sending SR's. Assuming that a typical SR is about 100 bytes long, any sender with a session bandwidth above 2100 b/s will send a report at least every 30 seconds.

In addition, an RTCP BYE message, sent when a group member leaves, releases the YESSIR state record and any resource reservations.

C. Reservation Styles

YESSIR defines two reservation styles, individual and shared. In individual reservations, every sender in a RTP session has a resource reservation of its own. As shown in Figure 4 (a),router Rt1 receives reservation requests from both senders S1 and S2. After making a reservation, there are two separate reservations on links between Rt1 to Rt2 and Rt3. Depending on the amount of requested resource, RTP data streams from S1 and S2 may have different levels of reservation.

In a shared reservation, all senders of an RTP session share a single resource reservation in the network. As illustrated in Figure 4 (b), the links Rt1-Rt2 and Rt1-Rt3 have a single shared reservation. The amount of resources reserved on the link is the least upper bound (LUB) of the individual flow requests from S1 and S2. For example, if S1 and S2 request 10 kb/s and 15 kb/s of bandwidth, respectively, the shared bandwidth for link Rt1-Rt2 will be 15 kb/s. If there is a reservation failure, the reservation rejection information and the merged flow specification will be piggybacked in the RTCP sender report. Receivers will feed back the failure information and rejected reservation request to all participating members, including the senders. The senders can use these reports from receivers to adjust their requests. Flow merging issues will be addressed further in section III-E.

D. Flow Specification

A sender can specify the resource it is requesting (the flowspec) in different formats. We have considered three types for YESSIR: IntServ, RTP PT (payload-type), and TOS (type-of-service).

For applications that support the IntServ traffic models, namely the controlled-load [14] and guaranteed service [15], the flowspec format will be the one that has already been designed by the IETF IntServ working group [16]. In the flowspec, the requested bandwidth, the burst size and a service class need to be specified explicitly.

For some well-known and well-understood traffic types such as voice, the flowspec contained in the RTCP SR can simply list the current RTP payload type [18]. Separate ranges of the payload type values have been set aside for audio and video, so that a router can assign RTP flows at different granularity: by session, by payload type value or by media class. To reduce the number of queues, a router may simply assign all voice traffic to

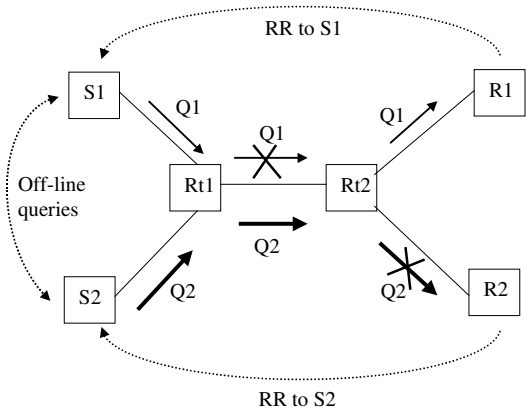


Fig. 5. Problems due to resource contention

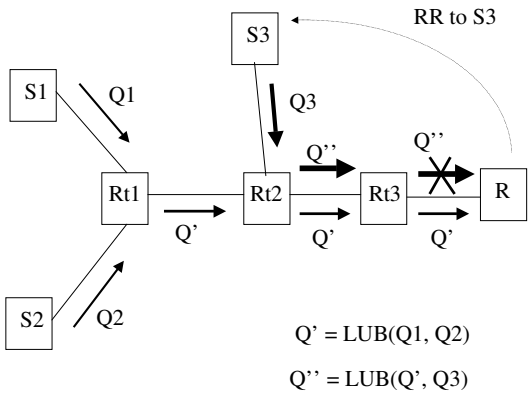


Fig. 6. Flow merging for shared reservation

a single high-priority queue, for example and just track the multicast destination and accumulated bandwidth for each session.

Similar to the RTP PT format, the TOS format allows routers to use the IP type of service information in RTP data packets to map them the appropriate scheduler queue. The YESSIR flowspec contains the TOS value and the allocated bandwidth. This allows the router to keep track of the bandwidth allocated for each TOS value, preventing over-commitment, yet avoids having to look up per-flow state for each packet. To prevent abuse by end applications, gateways rather than end systems would be expected to set the TOS value.

E. Killer Reservations

In a heterogeneous network, a reservation request may fail for any number of reasons at a router. Unfortunately, such failures may also affect requests from other senders. Figure 5 demonstrates the *killer reservation* effect. Two requests $Q1$ and $Q2$ (where $Q1 < Q2$) arrive at router Rt1. If $Q2$ arrives first and is accepted at Rt1, but rejected at Rt2, it could cause a smaller reservation $Q1$ to be rejected at Rt1 since the resource has been taken by $Q2$. As a result, neither request will enjoy and end-to-end QoS guarantee.

RSVP and ATM each solve this problem differently. RSVP generates RESVERR messages and creates a *blockade state* to allow smaller reservation requests going through while blocking large requests. Unfortunately, blockade states are difficult to manage and incur high implementation complexity. If an ATM reservation cannot be accepted by switch, that switch sends back a resource release message towards the sender to tear down the reserved resource at upstream nodes.

In YESSIR, partial reservations for both $Q1$ and $Q2$ will be made. However, senders receive an indication that the reservation was only partially successful and can then change or drop the reservation, clearing the way for other reservations to succeed.

F. Flow Merging

In YESSIR, flow merging only takes place for shared reservations. As discussed earlier, the merged flowspec is the least upper bound (LUB) value of the flowspecs from all participating senders. Here, we propose a best-effort approach to flow merging: when there is already a reservation in place, this reservation remains if a larger reservation request from another sender cannot be granted. As a result, all senders will have some fraction of their bandwidth reserved, though they may have different reservation requirements.

Figure 6 shows an example. S1 and S2 are the initial senders of a shared-reservation RTP session. The merged flowspec Q' is reserved inside the network, where $Q' = \text{LUB}(Q1, Q2)$. Later, a new sender S3 joins the RTP session and requests $Q3$ worth of resources. Router Rt2 tries to reserve the merged flowspec $Q'' = \text{LUB}(Q', Q3)$. Assume the reservation is successful and the new request Q'' is relayed to router Rt3. If Rt3 cannot reserve Q'', it should continue to use the previous reservation Q'. Sender S3 will be informed about the last workable reservation Q' from receiver R via RTCP and will ultimately decide if it wishes to continue to participate in the session or whether it can lower its sending rate.

G. Error Handling at Routers

In YESSIR, a router does *not* generate error messages to the senders, nor does it try to automatically correct problems such as *killer reservation* that are introduced due to reservation failures at neighboring routers. Instead, it inserts error information into the SR message. It is up to the receivers to inform the senders about reservation failures via RTCP receiver reports. Also, RTCP sender reports containing YESSIR reservation requests are always forwarded, even if unsuccessful.

We chose this approach for several reasons:

1. This behavior is simple to implement. As shown in several RSVP implementations [19], the support for error message handling and associated blockade states are costly in terms of protocol processing, timer management and extra state storage.

2. For links where resources are relatively plenty, such as a gigabit Ethernet, there is no reason to reserve resources for small data streams. In this case, a router should ignore YESSIR messages, and forward the requests downstream.
3. Managing resource over shared-media network such as Ethernet and token-ring networks is difficult. In this case, a router can insert a "reservation-undoable" flag in the error fragment of the RTCP SR message and forward it downstream.
4. More importantly, as described earlier, reservations are soft state. If a resource is not available at the first reservation time, there is always a possibility that reservation can be made during refresh times.

H. Dynamic Reservation Feature

An RTP session may not require a reservation for its whole duration. If reservations cost money, an application may well decide to only reserve network resources if best effort service proves unsatisfactory[4]. RTP-based applications supporting YESSIR can easily operate in this "reserve-when-needed" mode, as YESSIR reservation requests are coupled with RTCP messages. RTCP receiver reports have been designed to monitor traffic statistics. Senders can monitor receiver reports and only include a reservation request if a sufficiently large fraction of receivers indicate reception problems.

I. Network Resource Advertising

In order to satisfy end-to-end service requirements, we adapted the OPWA (*One-Pass With Advertising*) scheme proposed by Shenker and Breslau [20] and described by Wroclawski [16] for YESSIR. Here is how it works in the context of YESSIR: each reservation request message carries a network monitoring fragment that consists of fields for hop counts, propagation delay, aggregated bandwidth and delay bounds. As SR messages traverse routers, this fragment will be updated at every hop. The receivers, upon reception of the SRs, will send the collected path resource information back to the senders in RTCP receiver reports. The senders can refer the path resource information to adjust their reservation levels by sending new requests.

J. Updating the Packet Classifier

As shown in Figure 7, when a YESSIR message is received, the reservation setup agent will query the local traffic control database for resource availability. If the resource is sufficient at the egress interface(s), the agent updates the database and the scheduler.

According to the RTP profile, RTP data uses an even port number and the corresponding RTCP stream uses the next higher (odd) port number. Thus, during the parsing of RTCP messages, RTP data packet information including the IP source and destination addresses, port numbers and protocol type can be learned automatically.

After the router successfully sets up the scheduler, it inserts RTCP's IP source and destination address, protocol type (presumably, UDP), and the corresponding RTP data port numbers

[4]It obviously runs the risk that reservations will fail when the network is sufficiently busy to drop best-effort traffic.

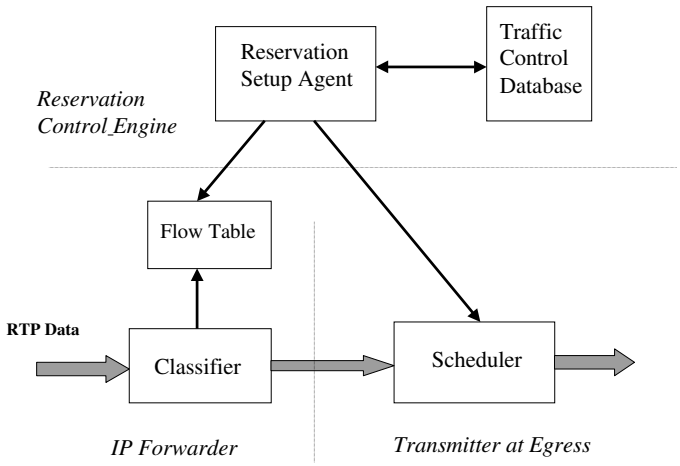


Fig. 7. A router model for reservation support

into the flow table. When RTP data packets are received, the packet classifier filters on the IP and UDP headers and forward the packets to the scheduler.

K. Security Considerations

RTCP and RTP data are tightly coupled. Thus, at a firewall, when a rule for a particular RTP data stream is defined, it will be automatically applied to the corresponding RTCP messages. Similarly, if a rule has been define to accept certain RTCP messages, the associated RTP data will be accepted as well. Supporting reservations across firewalls is therefore greatly simplified.

Currently, YESSIR relies on security mechanisms at the IP layer to provide authentication. If necessary, it would be easy to add an authentication facility to either RTCP or the YESSIR elements.

IV. Implementation and Measurements

A. YESSIR Processing Algorithm

The processing of YESSIR messages in a router is very similar to that of RSVP PATH messages, except that the router needs to call the local resource managers to make appropriate reservation in the case of YESSIR. Briefly, the following algorithm can be used to make reservation on arrival of a YESSIR message. The router will only "see" RTCP messages where the IP packet header has the router alert option set.
1. Perform a quick sanity checks on the UDP and RTCP headers;
2. locate the flow's reservation state in the router based on the IP source and destination addresses and UDP port numbers; [5]
3. if we cannot locate one, create a new reservation state;
4. query the routing tables to find egress interface(s);
5. make a reservation based the flowspec fragment in YESSIR message or the bandwidth measurement in the router state

[5]Alternatively, we could also hash on the 4-byte RTP synchronization source identifier (SSRC) instead of 12 bytes of source/destination information. Even though the SSRC is only unique within each RTP session, the probability of collision is low.

Code Section	Time (μsec)	% of Total
PATH Processing:		
PATH entry creation	410.51 ± 7.51	37.12%
Route query	40.61 ± 1.84	3.67%
Send PATH downstream	283.16 ± 3.85	25.61%
RESV Processing:		
RESV entry look-up	11.03 ± 0.43	1.00%
Update reservation info	126.35 ± 1.10	11.43%
Flow merging and forward RESV	234.14 ± 3.27	21.17%
Single RSVP flow setup overhead	$1{,}105.80 \pm 9.47$	100%

TABLE I

PROCESSING OVERHEAD FOR RSVP TRIGGER MESSAGE

Code Section	Time (μsec)	% of Total
YESSIR entry creation	41.40 ± 1.24	11.61%
Route query	38.43 ± 2.00	10.77%
Update reservation info	23.33 ± 0.38	6.54%
Send YESSIR msg downstream	253.53 ± 0.74	71.08%
Single YESSIR flow setup overhead	356.68 ± 2.84	100%

TABLE II

PROCESSING OVERHEAD FOR YESSIR TRIGGER MESSAGE

record;
6. store reserved resource information in the reservation state;
7. relay the message to each egress interface.

B. A Direct Measure of Protocol Overhead

We have implemented both RSVP and YESSIR on the IBM Common Router Architecture software platform. Both implementations have the similar data structures and coding style, and share the same set of data processing routines. We measured the various costs associated with RSVP and YESSIR on a router. The measured router was the IBM 2210 Nways Multiprotocol Router, which is based on a Motorola 68040 processor with bus speed of 32MHz. Processing times were measured by reading clock ticks from the timer register of the processor that has a timing resolution of 31.25 ns per tick. We divided the times into categories so that we can have somewhat loose subjective comparison between RSVP and YESSIR.

The underlying routing protocol used in our experiment is OSPF, and it updates the routing table in case of route changes. A route query operation is a straight forward route-table look-up. All RSVP flows are set up as controlled-load, fixed-filter style, and encapsulated in IP with the router-alert option. YESSIR messages use the RTP PT (Payload-Type) format, the individual reservation style, and are encapsulated in RTCP, UDP and IP with the router-alert option. Since the packet classifier and scheduler are implemented differently depending on the physical network interface, but are the same for RSVP and YESSIR, we chose to bypass them in our tests. Data collected here only reflects the RSVP and YESSIR control path behavior.

Tables I and II present the protocol processing overheads of setting a new RSVP or YESSIR flow. Table III and IV show the processing overhead of refreshing a flow.

C. The Processing Overhead Analysis

From the measurement shown in Tables I and II, we observe that a router can set up a new reservation flow with YESSIR three faster than if it uses RSVP. On soft-state refresh, the YESSIR processing overhead is nearly 50% less than that of RSVP.

Comparing all these times, we see that the overhead of constructing and sending a message is about 250 μsec. This includes getting a new buffer [6], copying data and scheduling for transmission. RSVP requires to send two messages to setup a flow, while YESSIR takes only one message.

[6] Alternatively, we can simply modify the received packet instead of getting a new buffer. However, RSVP and YESSIR are designed to support multicast flows, where multiple buffers may be required to forward a single packet at a router.

Code Section	Time (μsec)
On receive:	
PATH entry look-up	30.39 ± 0.76
Route query	37.94 ± 1.99
RESV entry look-up	11.01 ± 0.35
Update reservation info	44.05 ± 1.39
Timer routine:	
Send PATH Refresh	262.02 ± 10.20
Send RESV Refresh	239.06 ± 3.44
Single RSVP flow refresh overhead	624.46 ± 12.26

TABLE III
PROCESSING OVERHEAD FOR RSVP REFRESH MESSAGE

Code Section	Time (μsec)
On Receive:	
YESSIR entry look-up	19.31 ± 0.69
Route query	39.93 ± 0.38
Update reservation info	24.49 ± 0.31
Forward YESSIR downstream	252.56 ± 2.00
Timer routine:	
YESSIR flow checking	8.03 ± 0.73
Single YESSIR flow refresh overhead	344.32 ± 1.88

TABLE IV
PROCESSING OVERHEAD FOR YESSIR REFRESH MESSAGE

The packet transmitting overheads become more critical during soft-state refresh process. YESSIR relies on end-to-end soft-state refresh, that is, end users periodically transmit RTCP SR's with IP-alert option to maintain the flows inside the network. As a result, a YESSIR refresh message takes about 344 μs to process. RSVP uses hop-by-hop soft-state refresh mechanism. A RSVP router in the network is required to send refresh messages both upstream and downstream. As shown in Table III, refreshing a RSVP flow takes about 624 μs. Even worse, transmitting RSVP refresh messages takes place at timer interrupt level, which locks up the memory bus during the processing, thus stalling the packet forwarding loop. If a router maintains a large number of RSVP flows, its packet forward performance can be serious degraded due to long timer interrupts. YESSIR, on the other hand, uses the timer to check the flow lifetime only, and therefore takes far less time in each timer interrupt (approximately 8 μs).

The times reported above suggest accessing packet memory at router can be expensive. Creating new flow entry and updating reservation information require extensive message parsing and copying. Given YESSIR's PT flowspec is far more smaller (one word in the message) than RSVP's IntServ-format Sender-Tspec and Flowspec objects, we observed that the cost for creating a new YESSIR entry is nearly ten times less than that for RSVP, and the cost for updating reservation data in YESSIR is five times less than that for RSVP.

D. State Maintenance Overehad Comparison

YESSIR reduces the message overhead in the network. Figure 8 shows the message overhead for RSVP and YESSIR for various numbers of receivers over a link. The protocol overhead for one RSVP flow is the summation of a PATH and a RESV message. We illustrate the overhead of YESSIR messages with IntServ flowspec format, and RTP payload-type (PT) flowspec.

RSVP has the ability to pack multiple flows inside a single RESV message. The figure shows that, for a MTU of 1500 bytes, the total protocol overhead with the "compressed" RESV format is still higher than what is for YESSIR.

V. OPEN ISSUES AND FUTURE WORK

A. A Possible Solution for Scalability

RSVP has scaling problem due to its inability to aggregate small flows. If every voice flow over an IP backbone has an RSVP connection, a router may have to manage thousands of flows at each link. For example, an OC-3 (155 Mb/s) link can support 2400 64 kb/s voice flows, taking approximately 1.2 MB of storage[7]. At a refresh interval of 30 seconds, this requires about 230 kb/s of bandwidth, based on a size of a PATH message including ADSPEC of 208 bytes and a RESV message for guaranteed service and fixed filter of 148 bytes. The router has to process about 80 refresh messages a second.

In comparison, in a typical router deployed in the backbone, it takes about 0.4 MB to store 50,000 routes[8]. The bandwidth for route updates in a stable network is negligible.

One solution to support a large number of real-time flows is to make reservations in a hierarchical fashion, by using RSVP inside the backbone and establishing a small number of large-bandwidth "virtual paths", while reserving individual flows using YESSIR in local and regional networks.

As illustrated in Figure 9, inside the backbone, routers B1 and B3 operate as RSVP proxy servers, and have established a reserved flow, B1–B2–B3. Senders S1 and S2 use YESSIR to set up reservations to receiver R over the Internet backbone. In a YESSIR message, there is a bit to indicate whether the request is active (that is, every router needs to try to make the reservation) or passive (routers must ignore the request).

When reservation requests from S1 and S2 are received, B1 will first turn the reservation bit to passive mode in the requesting messages, preventing reservations from taking place inside the backbone. At B1, the packet classifier is updated to re-direct RTP data traffic from S1 and S2 to the pre-established RSVP connection. At B3, when requests from S1 and S2 are received,

[7] ISI and IBM have reported that a RSVP flow requires up to 500 bytes for storage in their implementations.

[8] In IBM's NSFnet routers, it takes a total of 8 bytes to store a route, including the support for CIDR.

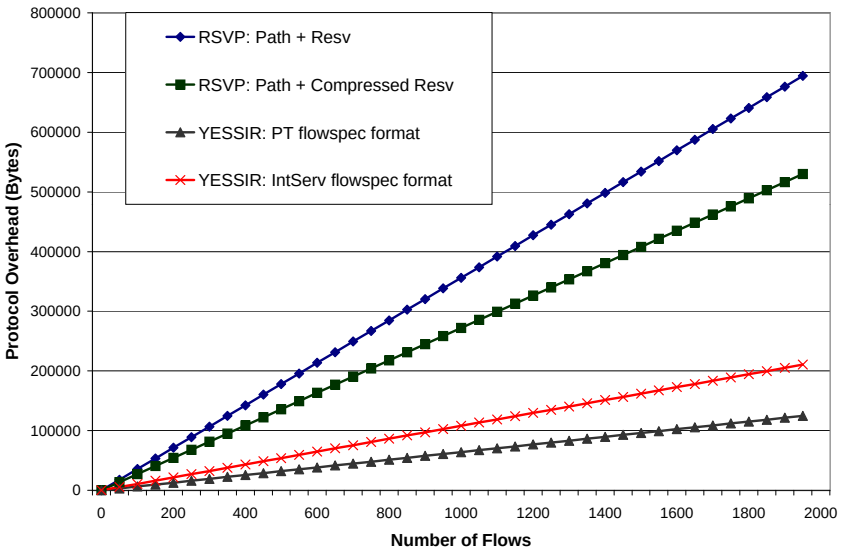


Fig. 8. Protocol overhead comparison between RSVP and YESSIR

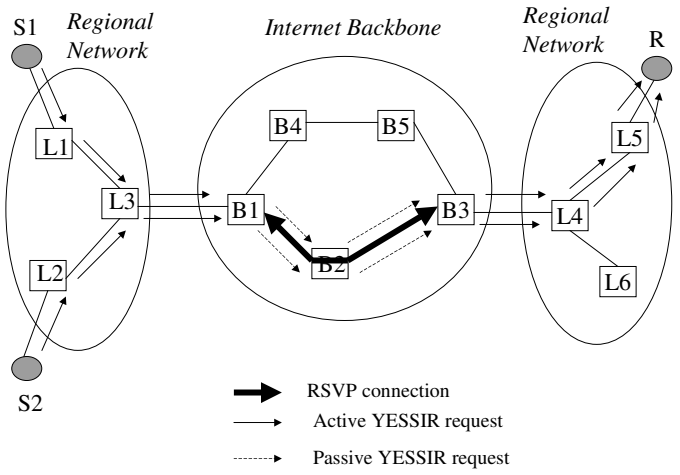


Fig. 9. An example of solving the scaling problem with two levels of reservation

the router will turn the reservation mode back to active. Requests will be routed toward the receiver R and make appropriated reservation along the way. An end-to-end reserved connection is therefore established.

Some of the unsolved issues are:

Selecting RSVP proxy servers: In the example, the RSVP flow is originated from B1 and terminated at B3. The mechanism and criteria to select a proxy server can be tricky: a BGP external speaker [21], a PIM rendezvous point [13], and a router managed by some policy agents are some of the candidates for RSVP proxy servers.

RSVP tunnel identification: The combination of source and destination address and port number should not be used to classify packets inside the backbone due to large storage overhead. How to classify packets in backbone RSVP routers needs to be studied. Possible solutions could be encapsulating data packets at the edge of the backbone, or making the use of CIDR [22], or managing IPv6 flow-ids properly.

Join YESSIR/RSVP gateway: In the example above, senders S1 and S2 have to somehow join themselves to the nearest

YESSIR/RSVP gateway, B1, prior to the reservation requesting time. The joining mechanism needs to be designed.

Reduce soft state overhead: Frequent refresh among routers can be costly if the number of flows to be managed is fairly large. On the other hand, infrequent refresh may reduce the system's ability to correct failure in timely fashion. A more efficient soft state management mechanism needs to be in place for YESSIR and RSVP. We will base our design on [23] and [24].

VI. Related Work

A number of protocols have explored sender-based reservations, including ST-II+ [25] and its predecessors, RTIP and RCAP [26] and CBSRP [27]. ST-II+ replaced IP with a new, connection-oriented Internet protocol and integrated resource reservation with establishing connectivity, thus making the smooth transition between reserved and best effort flows more difficult. RTIP and RCAP took a similar approach. All these protocols were out-of-band to the data protocol and used a "hard state" approach to state management, i.e., requiring explicit set-up and tear down of connections.

VII. Summary

Resource reservation is useful for supporting continuous-media services over the Internet. The question at this stage is: at what price? YESSIR provides a way to simplify the reservation processing and therefore reduce associated overhead at routers.

The YESSIR approach (1) is sender-initiated to support of "push" applications and simplify processing; (2) allows partial reservations; (3) supports multiple reservation styles; (4) uses soft state mechanisms to reliably and responsively maintain reservation states; and (5) takes advantage of the close relationship between RTP and RTCP packets for easy packet classification and firewall support.

We are in the process of implementing YESSIR on hosts and routers. The host implementation is based on *NeVoT*, *vic* and *vat*. The router implementation will be further developed and evaluated on experimental router prototypes from IBM Research.

We plan to interface YESSIR and RSVP to develop a hierarchical reservation system to solve the scaling problem.

References

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," RFC 1889, Internet Engineering Task Force, Jan. 1996.

[2] V. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California at Berkeley, Berkeley, California, May 1997.

[3] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a new resource ReSerVation protocol," *IEEE Network*, vol. 7, pp. 8–18, Sept. 1993.

[4] B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol (RSVP) – version 1 functional specification," RFC 2205, Internet Engineering Task Force, Oct. 1997.

[5] M. Handley, H. Schulzrinne, and E. Schooler, "SIP: Session initiation protocol," Internet Draft, Internet Engineering Task Force, Mar. 1998. Work in progress.

[6] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Palo Alto, California), pp. 117–130, Aug. 1996.

[7] S. McCanne and V. Jacobson, "vic: A flexible framework for packet video." in *Proc. of ACM Multimedia '95*. Nov. 1995.

[8] V. Jacobson, "Multimedia conferencing on the Internet," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, England), Aug. 1994. Tutorial slides.

[9] I. Kouvelas, V. Hardman, and A. Watson, "Lip synchronisation for use over the internet: Analysis and implementation," in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (London, England), Nov. 1996.

[10] H. Schulzrinne, "Voice communication across the Internet: A network voice terminal," Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.

[11] D. Katz, "IP router alert option," RFC 2113, Internet Engineering Task Force, Feb. 1997.

[12] D. Katz, R. Atkinson, C. Partridge, and A. Jackson, "IPv6 router alert option," Internet Draft, Internet Engineering Task Force, June 1997. Work in progress.

[13] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An architecture for wide-area multicast routing," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, UK), pp. 126–135, Sept. 1994.

[14] J. Wroclawski, "Specification of the controlled-load network element service," RFC 2211, Internet Engineering Task Force, Oct. 1997.

[15] R. Guerin, C. Partridge, and S. Shenker, "Specification of guaranteed quality of service," RFC 2212, Internet Engineering Task Force, Oct. 1997.

[16] J. Wroclawski, "The use of RSVP with IETF integrated services," RFC 2210, Internet Engineering Task Force, Oct. 1997.

[17] S. Jamin, *A measurement-based admission control algorithm for integrated services packet networks*. PhD thesis, Dept. of Computer Science, University of Southern California, Los Angeles, California, Aug. 1996.

[18] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," RFC 1890, Internet Engineering Task Force, Jan. 1996.

[19] G. Gaines and L. Salgarelli, "RSVP implementation survey," tech. rep., Institute for Information Technology of the National Research Council of Canada, July 1997.

[20] S. Shenker and L. Breslau, "Two issues in reservation establishment," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), Sept. 1995.

[21] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," RFC 1771, Internet Engineering Task Force, Mar. 1995.

[22] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless inter-domain routing (CIDR): an address assignment and aggregation strategy," RFC 1519, Internet Engineering Task Force, Sept. 1993.

[23] Jacobson, "Scalable timers for soft state protocols," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Kobe, Japan), Apr. 1997.

[24] P. Pan and H. Schulzrinne, "Staged refresh timers for RSVP," in *Proceedings of Global Internet*, (Phoenix, Arizona), Nov. 1997. also IBM Research Technical Report TC20966.

[25] L. Delgrossi and L. Berger, "Internet stream protocol version 2 (ST2) protocol specification - version ST2+," RFC 1819, Internet Engineering Task Force, Aug. 1995.

[26] A. Banerjea and B. A. Mah, "The real-time channel administration protocol.," technical report, UC Berkeley, 1991.

[27] S. T.-C. Chou and H. Tokuda, "System support for dynamic QOS control of continuous media communication," in *Third International Workshop on network and operating system support for digital audio and video*, (San Diego, California), pp. 322–327, IEEE Computer and Communications Societies, Nov. 1992.

Resource Reservation Agents in the Internet

Olov Schelén and Stephen Pink
Computer Science and Electrical Engineering
Luleå University of Technology, SE – 971 87 Luleå, Sweden
{olov, steve}@cdt.luth.se

Abstract

We present an architecture where clients can make resource reservations through agents. For each domain in the network, there is an agent responsible for admission control. The architecture provides scalable resource reservations for uni-directional virtual leased lines. Reservations from different sources to the same destination are aggregated as their paths merge toward the destination. When there are sufficient resources in a destination domain, prefix aggregation is performed. Thus, all reservations for a given destination domain are aggregated as their paths merge. Agents are responsible for setting up police points at edges for checking commitments.

1 Introduction

We are designing a resource reservation architecture where clients make admission requests through *agents* [3, 4, 5]. For each routing domain in the network there is an agent responsible for admission control. Each agent knows the topology and static link resources in its domain (figure1). The agent is an end-system that is configured for passively participating in a link-state routing protocol (e.g., OSPF) where each participating router has an identical topological database over the domain. Thus, an agent obtains a copy of the topological database with little signaling overhead, i.e., link-state advertisements are sent from the nearest router to the agent. Agents retrieve link properties, such as static bandwidths, by querying routers seen in the topological database. For this, we use a network management protocol (e.g., SNMP). Queries are done at startup and when topology changes are detected by the routing protocol. Routers need no extensions to allow agents to build a resource map through the management protocol. Agents use the resource map to perform admission control in their domain.

Admission requests contain the bandwidth to be reserved, a source and a destination address. For each request, agents can find where traffic will enter a domain and therefore agents can setup police points in edge routers checking that incoming traffic conforms. Packets using reserved resources are *marked* by applications or edge routers and checked by police points when entering a domain. Similar ideas, known as *differentiated services*, have been proposed in [1, 2].

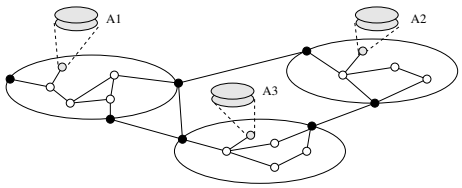


Figure 1: Reservation agents and their domains

In the differentiated services model, packets are marked to obtain either low drop probability or high scheduling priority. In this paper, we abstract from these details and use the term *priority packets* to denote marked packets, and the term *priority traffic* for all priority packets collectively. A key point of differentiated service and our agent architecture is to avoid non-scalable signaling state and expensive per packet processing in routers. Therefore resource reservations are handled by agents (end systems), and packet classification in routers involves only a small number of fixed service classes.

Independently of which packet scheduling model is used, there must be an admission control architecture to make sure that the amount of priority traffic is low enough to receive good quality of service. One issue is whether the admission control architecture should reserve resources over the links that are actually going to be used, or just limit the overall rights to send priority traffic. A scheme that doesn't consider exactly where traffic will go cannot give reliable commitments without very low utilization. Thus, it is likely that such a scheme would need to gamble and accept occasional service loss.

In this work we focus on agent-based admission control where the objective is to administer resources exactly, i.e., admission control in an agent maintains information about reserved resources on each link in its domain. Our objective is to find out if such a scheme can scale. The architecture offers a service compara-

ble to unidirectional *virtual leased lines*. In this model, requests may be immediate and open-ended or made in advance by including starting time and duration for the reservations. In [4] we show the implications of supporting a mix of immediate and advance reservations.

2 Reservation Model

An admission request can be directed to any agent, e.g., an agent that manages a user's account. Each agent sets up reservations between any two points in the network by invoking other agents. Thus there is *no difference* between reservations for *sending* or *receiving* data, reservations for *nomadic computing*, or reservations paid by a *third-party*.

Each reservation request contains a bandwidth to be reserved, a source and a destination address. The source address determines the point where the reservation starts (i.e, the point where traffic using the reservation will enter) and the destination address determines the point where it ends. An agent receiving a request first considers whether the starting point is in its domain. If the starting point is *not* in its domain, it finds an agent closer to the starting point and repeats the request with that agent. Technically speaking, it is possible to directly identify the agent that is responsible for the source domain, but in practice clients may prefer using a particular agent. Also, agents may prefer to deal with only a few adjacent agents. This means that a request is repeated in several steps. In figure 2, it is shown how a request is issued by node Dx for reserving resources from Ax to Dx.

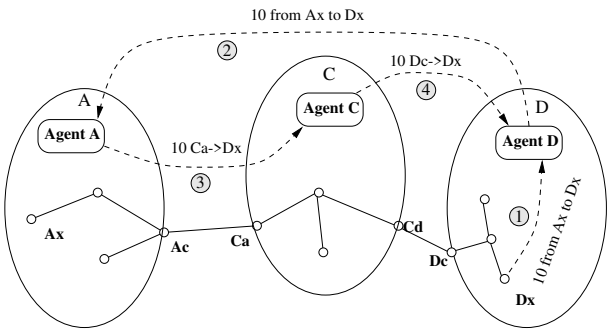


Figure 2: A receiver Dx requesting a reservation to receive data from Ax

If an agent finds that the starting point given in the request is in its domain, admission control is performed on the links from the starting point towards the destination, defined by the routing protocol. This is possible since all routers and the agent in a domain have the same topological database. Thus, an agent can perform admission control on the links from the starting point to an edge router in its domain.

An agent cannot perform admission control beyond its domain. Therefore, if the destination is outside of the agent's domain it must request a reservation with the neighboring agent, giving the edge point where traffic will cross the borders as the source address of the reservation (figure 2). Without distinguishing between reservations to different destinations it would be impossible for agents to grant reservations to other domains. This is because there is no way to make sure that there are sufficient reservations along any path in other domains further downstream (unless we accept very low utilization for priority traffic).

3 Aggregation

If reservations were always done "on demand" there would be signaling between all neighboring agents along the path for each end-to-end admission request. Therefore, our architecture supports several ways of aggregating reservation state in agents.

To grant requests spanning many routing domains, there must be a commitment from all agents along the path involved in the request. When an agent is making a reservation with a neighboring agent, the source address included in the request determines the point (router interface) where the traffic will cross the borders, i.e., enter the domain of the neighboring agent. The neighboring agent does not care where traffic came from originally. Thus, priority packets can use reserved resources as long as the destination address matches the reservation. This allows agents to aggregate traffic with different sources into one single reservation made with a neighboring agent.

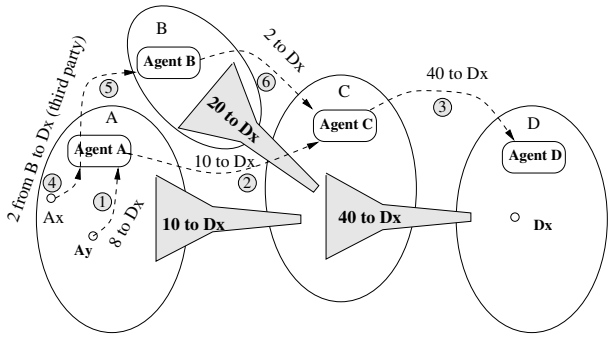


Figure 3: Funnels and aggregate reservations for one destination

An end-to-end resource reservation can be seen as a number of consecutive *funnels* (figure 3). Priority packets poured into any of these consecutive funnels use re-

served resources all the way to the destination. Agents may aggregate several requests with different source addresses into the same funnel, as long as they all specify the same destination. Agent C (in fig. 3) may set up a funnel starting at its edge by pre-reserving resources with neighboring agent D to a distant destination and then aggregate requests for resources between domains C and D into the established funnel. This bulk reservation implies that downstream agents only keep state for the aggregate. The agent that decides to aggregate into a bulk reservation keeps information about its individual commitments constituting the aggregate. Aggregation by merging reservations towards each destination can be done with full control over the resources. The funnels form a *sink-tree* with the root towards the destination. At each merging point, the reserved bandwidth on the outgoing link will be equal to, or larger than, the sum of the reserved bandwidth on the incoming links.

Requests that specify different destinations can not easily be aggregated into a funnel in the general case. If we allow funnels to split out in different directions, upstream agents must keep information about which destinations are involved and how resources are divided between them. This means that the aggregate cannot be seen as one unit. However, a reservation can cover different destinations if resources for the whole aggregate are reserved to any of the involved destinations. In addition, there should be a scalable way of representing the set of destination addresses. Both of these requirements are often met for well provisioned destination domains in the Internet.

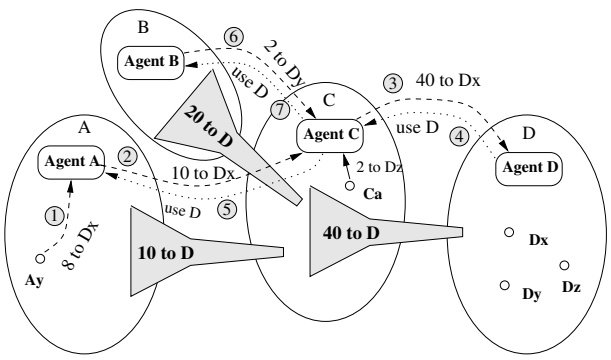


Figure 4: Funnels and prefix aggregation

Therefore, we allow aggregation to be increased by *generalizing* a reservation to be valid for all destinations within the same destination domain. It is only the agent responsible for the destination domain that can judge if resources are sufficient for allowing generalization. Consider the example in figure 4. Agent D can judge if the resources in D are sufficient to let incoming funnels be valid for the whole domain. Thus, agent D decides when a reservation request for a particular destination node within D can be replaced with a reservation for any node in domain D (or a subnet within domain D).

In IP networks an *address prefix* is often used to represent a domain (individual nodes within the domain are distinguished by the remaining suffix). An address prefix is represented by an IP address and an associated mask telling how many bits that are part of the prefix. An agent generalizes a request by replying with an *address prefix* to be used as destination address for the reservation (figure 4). The aggregation that results from generalizing a reservation is therefore called *prefix aggregation*.

Clients or agents that have been granted a prefix aggregated reservation can use it for any destination matching the prefix of the reservation. Agents providing prefix aggregated reservations must provide the committed bandwidth of priority traffic to any destination in the domain denoted by the prefix. A conservative agent would therefore allow prefix aggregation up to a value equal to the bandwidth of the narrowest link. In practice we believe that this can be quite useful for well provisioned local area networks.

To sum up, aggregation is happening in the agents as routers are not dealing with admission control at all. The motivation for aggregation may be less when reservation state is stored in agents with plenty of memory and disc. However, a per-flow reservation model has inherent scaling problems, both in terms of state and processing cost in agents. Therefore, it is important to show that agents can provide exact resource reservations with good scaling properties. Since agents have immediate access to the routing database, they can also handle changing routes and topologies with an outcome as favorable as possible.

4 Policing and classification

The source address in an admission request is used by agents to set up an ingress police point, i.e., in the router closest to the source. Egress policing can be setup by using the destination address, i.e., the routing protocol can find the point where traffic is leaving the domain. By policing at the edges, a provider can check that the rate of priority traffic stays within commitments. At each edge point, there may be several commitments to different destinations. Policing of individual commitments would therefore involve classifying priority packets by their destination addresses before measuring used bandwidth. Although the number of commitments are kept low through aggregation, policing is demanding as it involves per-packet process-

ing compared to reservation setup which involves only per-commitment signaling. At the edge routers, close to the original sources of data, per-commitment policing may be affordable as there are generally few commitments and a low aggregate data rate. If all agents police correctly at the original sources, and we have an architecture that reserves resources along exact routes end-to-end, there is little need for policing in the backbone.

However, a model relying on policing at original sources only, would be volatile as it is based on global trust. Therefore, we suggest adding a model for scalable policing at edges of backbone domains. The method is to check a dynamic subset of the commitments. The agent can go through its commitments associated with the actual edge point and set up close policing by matching different destinations one-by-one, or randomly, in the router. The key idea is that the agent has knowledge about any policing that could be done, but saves per-packet policing overhead in the routers by not checking all commitments at once. When violating traffic is found, it is reported to the agent (the agent sets up policing and a trap with the router by using a management protocol). When an agent gets a report from a police point about a violated commitment, the agent should report it back to the upstream agent. The upstream agent has knowledge about the constituent parts that were aggregated into the violated commitment. By performing the same method at each agent, the police point for catching the violating traffic will gradually move back towards the sender (as far as there are cooperating agents), where it should have been regulated in the first place. For managing policing from agents, one architectural requirement is to standardize an interface to allow trusted agents to set up police points in their routers.

5 Interoperation with RSVP

In RSVP [6], reservations are made for individual data flows (i.e., application data streams). In that model, per-flow reservations are maintained in the routers. This results in scaling problems due to reservation state and packet classification overhead in routers. This model inherently relies on per-flow semantics and therefore it is hard to perform aggregation across different administrative domains. This has led some of the RSVP designers and others to advise keeping RSVP out of the backbone.

Having per-flow reservation state in the routers, as in RSVP, is not advisable for unicast reservations. Even if aggregation is supported over some parts of the path, there would generally be an explosion in state and packet processing overhead compared to ordinary best-effort routing. For multicast, the overhead for RSVP can probably be motivated. Already with best-effort multicast, the network trades off bandwidth savings for per-flow state in the routing table. The extra state for RSVP would probably not add too much overhead. One should, however, make sure that multicast and RSVP are used only when resulting multicast trees are sufficiently dense (branched) to justify the control state. Therefore, the agent architecture can be used for unicast reservations, while RSVP is used for multicast reservations.

6 Conclusion

The agent architecture can provide resource reservations in a global network for offering virtual leased lines from any point to any point (i.e., independently of where reservation requests originate). The key ideas of our architecture are *aggregation* as paths merge towards destinations, *prefix aggregation* over destination domains, *advance reservations*, and minimal *flexible policing*.

The architecture separates functionality between packet forwarding in routers and QoS negotiation in agents. For enforcement, agents must set up police points in their edge routers.

References

[1] D. D. Clark and W. Fang. Explicit allocation of best effort packet delivery service. Technical report, MIT Lab for Computer Science, Boston, Massachusetts, November 1997.

[2] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the internet. Internet draft, Bay Networks, LBNL and UCLA, November 1997.

[3] O. Schelén and S. Pink. An agent-based architecture for advance reservations. In *IEEE 22nd Annual Conference on Computer Networks (LCN'97)*, Minneapolis, Minnesota, November 1997.

[4] O. Schelén and S. Pink. Sharing resources through advance reservation agents. In *Proceedings of IFIP Fifth International Workshop on Quality of Service (IWQoS'97)*, New York, May 1997.

[5] O. Schelén and S. Pink. Aggregating resource reservations over multiple routin domains. In *Proceedings of IFIP Sixth International Workshop on Quality of Service (IWQoS'98)*, Napa Valley, California, May 1998.

[6] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: a new resource ReSerVation protocol. *IEEE Network Magazine*, 7(5):8–18, September 1993.

Achieving Quality of Service through Network Performance Management

S. Keshav and R. Sharma
Cornell Network Research Group
Department of Computer Science
Cornell University, Ithaca, NY 14853
{skeshav, sharma}@cs.cornell.edu
http://www.cs.cornell.edu/cnrg

Abstract

We believe that current mechanisms for providing quality of service guarantees in integrated services networks have two major flaws. First, these mechanisms, such as signaling, scheduling, and reservations, require significant changes to the existing Internet. Second, they do not provide mechanisms for management of QoS state. In this paper, we present an alternative approach which attempts to provide 'macroscopic' QoS guarantees through effective network management.

1. Introduction

The Internet is exponentially increasing in *breadth* and *depth*. By an increase in *breadth*, we mean that the Internet is reaching ever more users, doubling its user population every year. This increase is likely to continue as developing countries get connected and small-footprint TCP/IP implementations bring embedded computers online. There is a parallel increase in *depth*, that is, in the set of services available to an endpoint. For many years, this set was small, and was provided by applications such as FTP, telnet, Usenet, and email. In recent years, the set has rapidly increased and now includes the World Wide Web, financial services like stock trading, distance education through audio and video multicast, and access to numerous databases.

Because of its rapid increase in breadth and depth, the Internet is widely accepted as an essential part of the global communication infrastructure. How well does it fit this role? A study of Internet performance reveals that the user experience is highly variable and often poor. While there are many millions of satisfied users, it is not hard to find angry customers unable to complete a stock trade, confused students whose video feed has been mysteriously cut off, or users who just cannot connect to popular websites. In view of the social and economic cost of these problems, there has been a great effort, over the last decade, to build networks that can guarantee some form of quality of service. (The solution of this general problem automatically solves the more restricted problem of supporting continuous-media applications in an integrated services network.)

Providing users with a consistent and reliable quality of service has usually been studied from the perspectives of scheduling, traffic profiling, signaling, and reservations. In practice, this may prove to be too narrow a view of the network. *We believe that the service quality experienced by a typical Internet user today is determined more by the quality of network management than by scheduling and reservation mechanisms*. This is particularly true of users on legacy networks that rarely support per-user or per-application quality of service guarantees. In such networks, network performance management tools that help in judiciously selecting link capacities and routing metrics may do more to improve a user's service quality than the introduction of complex mechanisms such as Weighted Fair Queueing, RSVP, and differential pricing. Moreover, even if more sophisticated mechanisms are introduced and become popular, we would still need tools to administer policies and manage network performance by tuning parameters such as the degree of aggregation and the weights associated with each service class.

Unfortunately, existing network management systems do a poor job of managing current networks, let alone the complex mechanisms that may be introduced in future networks. They are usually 'GUI-centric', painting pretty pictures of network status, rather than helping to find and fix network performance problems (for example, see HP's NetMetrix [1]). In the absence of good tools, scientific process and good engineering practice gives way to trial-and-error and a 'gut-feeling' approach to performance management. For instance, network operators plan the layout and configuration of their network in an *ad hoc* manner. However, naively adding capacity to a network

can actually lead to a decrease in the performance of the network! Similarly, at present, one cannot test a network component without actually plugging it into the network--the only way to completely test a router's BGP configuration is to plug it into the Internet. However, a wrongly configured backbone router can accidentally partition the Internet, a fact that may be easy to discover but hard to correct. Clearly, we need better tools to plan network topology, configure components, and to test a component without compromising the integrity of the network.

Our research goal, therefore, is to create a suite of network performance management tools that allow network administrators to provide quality of service to users. We do so by developing algorithms and tools for network discovery and monitoring, visualization, fast simulation, capacity planning, router configuration, and fault diagnosis. We believe these form the basis for the next generation of network performance management systems, which not only observe network functioning, but also actively participate in diagnosing and managing network performance. In the rest of this paper, we outline the current state of our tools.

2. The network lifecycle

Our approach to network management is based on the observation there are five stages in the evolution and management of networks (Figure 1).

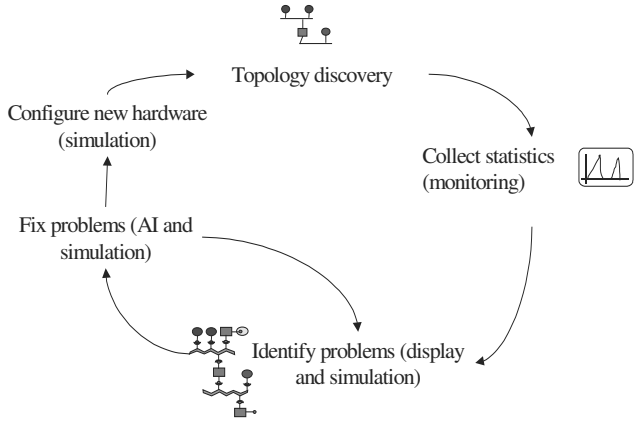


Figure 1. Network performance management lifecycle

These are (a) discovering existing topology, (b) collecting statistics, (c) identifying performance problems, (d) dealing with performance problems, and (e) configuring new hardware. We are developing algorithms and tools for each of these stages, as outlined next. We note in passing that current network management solutions deal only with stages (a) and (b). A significant exception is the Netsys suite of tools from Cisco [2], which offers tools for all five phases of the lifecycle. This product, however, is proprietary, and we are not aware of public descriptions either of its algorithms or its effectiveness. Moreover, it is targeted primarily at dealing with L-2 problems in networks of Cisco routers: we hope to model and manage heterogeneous L-3 networks.

2.1. Discovering topology

Network topology refers not only to the physical layout of components and cables within a given domain, but also the logical partitioning of the domain into subnets. The basic idea in our algorithm is to mark discovered nodes (corresponding to IP addresses) as 'temporary' or 'permanent' (by analogy with Dijkstra's algorithm). Each temporary node, when accurately identified as a router or an end-system, is added to the permanent set. Neighbors of the newly added permanent node are then added to the temporary set for further processing. Thus, network discovery problem reduces to coming up with algorithms for discovering temporary nodes and for identifying permanent nodes. Some of the heuristics we use for generating temporary nodes are to (a) do a DNS domain transfer, which lists every name and IP address in the domain, (b) `ping` a random IP address in the domain, adding it to the temporary set if it responds (c) `traceroute` a random IP address in the domain and add every node along the discovered path, and (d) use SNMP to get a router's routing table, then extract all the nodes in the table. Next we cast out spurious addresses, then discover routers, links, and link speeds. We discover a router either using SNMP (an SNMP GET returns the identity) or DNS (a lookup returns multiple IP addresses). We discover links by SNMP queries on routers or by carefully chosen `traceroutes`. Finally, we discover link speeds using Jacobson's `pathchar` program [3]. The discovered topology information is stored in a hierarchical file system. The attributes of each node are stored as files within a directory devoted to the node. This simplifies access to the topology. For instance, to ping all the hosts in subnet 128.84.154 we simply type `cd 128.84.154; foreach i(*); ping $i; end;`

Although our discovery tool still needs tuning, it can discover all the 10,000 hosts and routers in the `cornell.edu` domain, and over a million nodes in the Internet at large. We are currently experimenting with a new discovery algorithm that correlates `traceroutes` to the same randomly chosen IP address from different sites in the Internet to discover Internet backbone topology. For instance, a `traceroute` to `ncst.ernet.in` from hosts in Stanford and Berkeley uniquely identifies the ISP connecting this host to the Internet.

2.2. Collecting statistics

Traditional network management tools use SNMP to collect statistics from managed objects. SNMP requires the manager to poll the managed object to obtain information about dynamically changing MIB variables. However, if there is a large delay between the manager and the managed object, we cannot do fine-grained polling or implement efficient network control algorithms. Moreover, during periods of congestion, precisely when control is important, SNMP requests may be lost. Therefore, in addition to using SNMP to collect network statistics, we have developed an extension to SNMP called 'Active SNMP' that solves these problems [4]. Active SNMP allows a Java applet to execute on a Java runtime 'near' the managed object, carrying out computation on MIB variables on behalf of the manager. For instance, the applet could monitor the number of IP packet losses and use this to adjust the advertised link weight. By placing processing near the MIB, Active SNMP allows us to perform fine-grained control, and avoid the 'horizon effect' where event information in a remote domain is invisible because of aggregation.

While Active SNMP solves some problems with monitoring, the question of deciding *which* nodes to monitor is still open. To solve this, we fit the traces collected from a particular site to an empirically determined mean. If a coarse-grained trace reveals that parameters of a fitted curve deviate substantially from normal, we automatically install an Active SNMP applet to more carefully monitor the situation.

2.3. Topology display

Discovery and monitoring create data sets that need to be efficiently and intuitively displayed. Instead of using a Visual Basic, Tcl, or Xlib GUI, we decided to use a standard Web browser as the user interface. We have created a scripting language that extends Javascript by wrapping methods in the Java AWT. Thus, window events, normally visible only to Java, are now exposed to Javascript. This allows us to rapidly create UI components in Javascript. Moreover, by writing a small windowing system in Dynamic HTML, we can represent multiple views of the topology, as well as current monitoring data within a browser. Our approach allows network management to be carried out from practically anywhere, and to display network performance information to interested users without a costly investment in a proprietary GUI.

2.4 Identifying performance problems

Performance problems in networks can be categorized roughly into Level-2 and Level-3 problems. Level-2 problems can be detected by monitoring collisions at routers and hubs, which we can do using the Active SNMP framework outlined above. (We also intend to come up with algorithms to map from collision statistics to probable causes of frequent collisions.) At Level-3, we can not only passively detect problems such as loss of connectivity using SNMP, but can *actively* probe the network in two ways. The first approach involves proactively testing paths in the network for routing loops and black holes with IP loose source routing. The second approach, which is more sophisticated, draws on network simulation techniques. We have designed and implemented an object-oriented network simulator called SurREAL that allows us to run multiple instances of the 4.4 BSD kernel networking code within a single Solaris or Windows NT address space. Moreover, existing applications can be ported to this environment with nearly no change. Our careful design allows the simulated network to appear to be equivalent to an actual network in every way. Thus, simulated nodes can be `ping`ed, and also support `ifconfig` scripts. This is an extremely powerful technique to test networks. For instance, we can create simulated test networks that generate and respond to routing updates to test the behavior of the actual network. As another example, we can simulate link and router failures to test its effect on the rest of the network. We can also deploy and test algorithms for differential service, resource reservation, and routing in a tightly controlled environment.

2.5 Correcting network performance problems

We plan to exploit SurREAL to correct network performance problems. Our approach is to use our discovery and monitoring tools to accurately model network topology, protocols, and workloads within a simulator, then use this to recreate observed performance problems. Then, network administrators can test out potential solutions (such as adding new hardware, or changing routing parameters) before implementing them in the network. To successfully carry out this phase of the work, we may need to simulate packet transmission and routing for networks with thousands of nodes and tens of thousands of simultaneous sessions. This is impossible with the existing packet-level simulation approaches (including SurREAL). Thus, we plan to come up with techniques for *session-level simulation* that will enable us to meet this goal. In this approach, we plan to create a map from the number of simultaneous TCP or UDP sessions to the expected performance of each such session from empirical observations. The map will therefore allow us to simultaneously compute the traffic flows traversing each router (from the routing simulation) and the performance of each flow (from the empirical map). While the details of this approach and its effectiveness still are under study, we believe that this will allow us to capture network performance at least to first order.

2.6 Configuring new hardware

One outcome of the previous steps may be the need to procure new hardware such as routers and hubs. These network components are not easy to configure. For instance, current routers require users to specify numerous interdependent configuration variables. Mistakes in router configuration are the most common reason for network outages. We plan to solve this problem in two ways. First, we plan to hook up a network component to SurREAL and put it through its paces before it is plugged in, thus identifying misconfiguration and potential performance problems before they occur. Second, we plan to build and test policy servers to centralize the administration of network components. This integration of policy management and simulation is a powerful tool for network performance management.

3. Conclusions

We believe that as the Internet becomes more and more important in daily life, there is a serious need not only to provide QoS guarantees in terms of delay and bandwidth for audio and video traffic, but also in terms of connectivity, robustness, and good performance for traditional best-effort traffic. We believe that current approaches to providing service quality are too narrow to achieve this goal. In contrast, our pragmatic approach combines tools for discovery, monitoring, display, and simulation that allow network administrators to build and maintain high-quality networks. While this paper describes work that is still in progress, our initial results are encouraging. Our experience leads us to believe that, in the long run, our approach may be only one actually able to deliver QoS to users.

4. References

[1] HP NetMetrix, http://www.tmo.hp.com/tmo/ntd/products/products.html

[2] Cisco Netsys, http://www.cisco.com/warp/public/734/toolkit/index.html

[2] V. Jacobson, “Pathchar” binary, ftp://ftp.ee.lbl.gov/pathchar/

[3] R. Sharma, S. Keshav, M. Wu, and L. Wu, **“**Environments for Active Networks,” *Proc. NOSSDAV ’97,* May 1997.

On the Effectiveness of Buffer in Deterministic Services *

Pawan Goyal and **Harrick M. Vin**

Pawan Goyal
goyal@research.att.com
AT&T Labs - Research
Networking and Distributed Systems Center
Florham Park, NJ 07932

Harrick M. Vin
vin@cs.utexas.edu
Department of Computer Sciences
University of Texas at Austin
Taylor Hall 2.124, Austin TX 78712

Abstract

We study the utility of buffer at switches in increasing the achievable utilization of a network providing deterministic guarantee. To determine the increase in utilization, we classify packet scheduling algorithms into two classes. Only one of these classes can utilize additional buffers to increase the achievable utilization. We experimentally determine the difference in achievable utilization of these classes. Our experiments demonstrate that contrary to intuition*, in most cases, additional buffers do not lead to higher achievable utilization of a network.*

1 Introduction

Integrated services networks support a diverse set of applications (e.g., data, audio, interactive video, stored video applications). The traffic characteristics as well as Quality of Service (QoS) requirements of these diverse applications vary significantly. To meet the QoS of requirements of the applications, a network has to manage two resources: link bandwidth and packet buffers. While several packet scheduling algorithms for managing link bandwidth have been developed and analyzed, the issues that arise in managing packet buffers for providing QoS guarantees have not been adequately investigated and are the subject of investigation of this paper.

To illustrate the issues that arise in managing buffers, consider a fluid flow model of the sources and the network. The switches in the network may either have no buffer or finite buffer. If the switches have no buffer, then fluid loss occurs whenever the aggregate arrival rate of flows at the output link of a switch exceeds the link capacity. In case of finite buffer, fluid is buffered when aggregate rate exceeds the capacity, and loss occurs only when the buffer overflows. Consider both types of networks when they provide deterministic guarantees on QoS parameters such as delay and loss. If the network does not have any buffers, then burstiness of a source may have to be smoothed at the edge of the network to ensure that loss does not occur. Such smoothing may introduce delay. Consequently, to meet the end-to-end delay requirements, resources greater than what is necessary may have to be reserved. On the other hand, if the network has buffers, then such smoothing and the consequent delay may not be necessary. This may enable the network to meet the delay requirement of a larger set of flows.

Thus, intuitively, it appears that use of buffer in a network should increase the achievable utilization of fluid flow networks. Hence, the question of interest is: Is the intuition correct for packet-switching networks? If yes, how much does the achievable utilization increase by adding buffer? Answers to these questions would determine the source rate control mechanism, which controls the smoothness of a flow, that should be employed in a network. If buffer in network does not yield higher utilization or the increase in utilization is not sufficient to offset the additional cost, a rate controller may completely smooth a flow to generate a constant bit rate (CBR) flow. If the buffer in the network yields increase in the achievable utilization, a source rate control mechanism may not smooth a flow at all or smooth it to appropriately tradeoff increase in utilization with increase in buffer requirement. Thus, the effectiveness of buffer in increasing achievable utilization would determine the appropriate buffering alternative, i.e., to buffer at source or in network.

In this paper, we take a step towards answering these questions. To determine if buffers at switches, henceforth referred to as servers, increase the achievable utilization

*This research was supported in part by IBM Faculty Development Award, Intel, the National Science Foundation (Research Initiation Award CCR-9409666, CAREER award CCR-9624757, and Research Infrastructure Award CDA-9624082), NASA, Mitsubishi Electric Research Laboratories (MERL), and Sun Microsystems Inc.

of a network providing deterministic guarantees, we classify the packet scheduling algorithms as either belonging to the class of Guaranteed Rate (GR) scheduling algorithms or Rate Controlled Service Disciplines (RCSD). Whereas the scheduling algorithms in RCSD can utilize additional buffers to possibly increase the achievable utilization, the achievable utilization of GR scheduling algorithms is unaffected by the available buffer space. We experimentally determine the difference in achievable utilization of RCSD and GR servers when RCSD servers employ additional buffers. Our experiments demonstrate that *contrary to intuition*, in most cases, additional buffers in a network of RCSD servers do not lead to higher achievable utilization. This experimental result indicates that if a source desires deterministic bounds on packet delay, then the traffic should be completely smoothed at the source.

The rest of the paper is structured as follows. We formulate the problem under investigation and present the background for it in Section 2. We present our experimental methodology in Section 3. The experimental results are presented in Section 4 and then discussed in Section 5. Finally, Section 6 summarizes our results.

2 Background and Problem Formulation

The sequence of packets transmitted by a source is referred to as a flow and is used synonymously with a connection. A flow is serviced by a sequence of servers. Let there be K servers on the path of a flow and let the i^{th} server on the path be server i. For simplicity of exposition, let the propagation delay between the servers be 0 [1]. To provide deterministic guarantees on packet delay and loss, each server employs a packet scheduling algorithm. The achievable utilization of a server depends on the packet scheduling algorithm. Hence, to determine the effectiveness of buffer at servers in increasing achievable utilization, let the scheduling algorithms proposed in the literature be classified into two classes:

- Guaranteed Rate scheduling algorithms: The class of Guaranteed Rate (GR) scheduling algorithms is defined based on *expected arrival time* of a packet. Let p_f^j be the j^{th} packet of flow f. Then, expected arrival time of packet p_f^j at server i, denoted by $EAT^i(p_f^j)$, is defined as:

$$EAT^i(p_f^j) = \max\left\{A^i(p_f^j), EAT^i(p_f^{j-1}) + \frac{l_f}{R_f}\right\} \quad (1)$$

where $j \geq 1$, R_f is the rate reserved for flow f, l_f is the length of packets of flow f [2], $A^i(p_f^j)$ is the arrival time of p_f^j at server i, and $EAT^i(p_f^0) + \frac{l_f}{R_f}$ is defined to be 0.

Scheduling algorithm at server i belongs to GR if it guarantees that packet p_f^j will be transmitted by $EAT^i(p_f^j) + \alpha_f^i$ where α_f^i depends on the scheduling algorithm as well as the server and the flow characteristics [6]. Several algorithms such as Virtual Clock [15, 18], Weighted Fair Queuing [1], Start-time Fair Queuing [8], Self Clocked Fair Queuing [4], Leap-forward Virtual Clock [14], Frame-based Fair Queuing [13], Delay EDD [7, 17] belong to the GR class. It has been shown in [6] that if each of the server on the path of a flow employs a scheduling algorithm in GR, then the end-to-end delay of packet p_f^j is given as:

$$d_f^j \leq EAT^1(p_f^j) - A(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n \quad (2)$$

where $A(p_f^j)$ is the time at which packet p_f^j is generated at the source.

- Rate Controlled Service Disciplines: The class of Rate Controlled Service Disciplines (RCSD) is defined based on the concept of shapers [3]. A shaper is a network element that ensures that its output traffic satisfies some burstiness constraints. If $AP^i(t_1, t_2)$ denotes the number of bits that are output from shaper S^i in time interval $[t_1, t_2]$, then shaper S^i delays the departure time of packets to ensure that $AP^i(t_1, t_2) \leq F^i(t_2 - t_1)$, where the function F^i characterizes the shaper S^i.

  Scheduling algorithm at server i belongs to the RCSD class with shaper S_f^i and delay d_f^i if it ensures that the departure time of a packet from the server is at most d_f^i more than its departure time from shaper S_f^i. RCSD class contains work conserving as well as non-work conserving algorithms (see [3, 16] for some examples). It has been shown in [3] that if each of the server on the path of a flow employs a scheduling algorithm in RCSD, then the network guarantees that the maximum end-to-end delay of any packet, denoted by d_f, is given as:

$$d_f = D_f^{\overline{S}} + \sum_{n=1}^{n=K} d_f^n \quad (3)$$

where $D_f^{\overline{S}}$ is the maximum delay incurred by the source traffic at shaper $\overline{S}_f$, which is a composition

[1] All the subsequent analysis is applicable when the propagation delay is non-zero but bounded. However, use of non-zero propagation delay would clutter the presentation without providing any insight into the problem under investigation.

[2] For ease of exposition, we have assumed that the length of the packets of a flow does not vary.

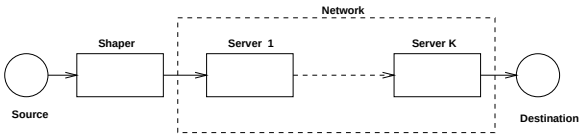


Figure 1: Network with Traffic Shaper

of shapers $S_f^1, .., S_f^K$. Intuitively, $\overline{S}_f$ is a shaper composed by sequentially connecting shapers $S_f^1, .., S_f^K$ (see [3] for a precise definition of composition of shapers).

Observe that the bound on the departure time of a packet in GR at server i can be interpreted as being at most α_f^i more than the departure time in a shaper that delays a packet till its expected arrival time. Hence, if scheduling algorithm at server i belongs to GR, it also belongs to RCSD with $d_f^i = \alpha_f^i$ and shaper S_f^i that delays packet till its expected arrival time. However, since arbitrary shapers may be employed in RCSD, the converse is not true, i.e., a scheduling algorithm in RCSD may not belong to GR. Hence, GR is a proper subset of RCSD.

The main difference between GR and RCSD class is that if the burstiness of a source is completely smoothed at the edge of a network to reduce the buffer requirement in the network, then whereas the end-to-end delay does not increase in case of a network of GR servers, it may increase in case of a network of RCSD servers. To observe this, let a shaper be employed at the edge of the network, i.e., between the source and the first server (see Figure 1). Let the departure time of packet p_f^j from such a shaper, denoted by $L^S(p_f^j)$, be equal to its expected arrival time, i.e., $L^S(p_f^j) = EAT^S(p_f^j)$. Such a shaper generates a smooth constant bit rate (CBR) flow. Now consider the end-to-end delay in such a system in a network of GR and RCSD servers.

- Network of GR servers: The maximum delay incurred by a packet consists of two components - the delay in the shaper and the delay in the network. The delay in the shaper of packet p_f^j is $L^S(p_f^j) - A(p_f^j) = EAT^S(p_f^j) - A(p_f^j)$. Using (2), we conclude that the maximum delay of packet p_f^j in the network is $\widehat{EAT}^1(p_f^j) - \widehat{A}^1(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n$ where $\widehat{EAT}^1(p_f^j)$ and $\widehat{A}^1(p_f^j)$ are the expected arrival time and actual arrival time of p_f^j at server 1 when the shaper is employed. Hence, we conclude:

$$d_f^j \leq EAT^S(p_f^j) - A(p_f^j) + \widehat{EAT}^1(p_f^j) - \widehat{A}^1(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n \quad (4)$$

Since propagation delay has been assumed to be 0, the arrival time of a packet at the shaper is the same as the arrival time at the first server in a network without the shaper. Hence, $EAT^S(p_f^j) = EAT^1(p_f^j)$. Furthermore, due to the use of shaper, $\widehat{EAT}^1(p_f^j) = \widehat{A}^1(p_f^j)$. Hence, we get:

$$d_f^j \leq EAT^1(p_f^j) - A(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n \quad (5)$$

Since (5) is same as (2), we conclude that the end-to-end delay does not change due to the presence of the shaper.

- Network of RCSD servers: Let the maximum delay incurred by packets in the shaper at the edge be denoted by D_f^C. Also, let the maximum delay incurred by the packets at shaper $\overline{S}_f$ after being shaped at the edge be $\widehat{D}_f^{\overline{S}}$. Hence, maximum end-to-end delay is given as:

$$\widehat{d}_f = D_f^C + \widehat{D}_f^{\overline{S}} + \sum_{n=1}^{n=K} d_f^n \quad (6)$$

Now, $D_f^C + \widehat{D}_f^{\overline{S}}$ may be greater than the delay $D_f^{\overline{S}}$ experienced by packets at shaper $\overline{S}_f$ when traffic is not shaped at the edge. To observe this, let $\overline{S}_f$ be a (kl_f, r_f) leaky bucket shaper and flow f conform to (kl_f, r_f) leaky bucket. A shaper is a (σ, r) leaky bucket shaper, where σ is the burstiness and r the average rate, if it ensures that for all intervals $[t_1, t_2]$:

$$AP(t_1, t_2) \leq \sigma + r(t_2 - t_1) \quad (7)$$

A flow conforms to (σ, r) leaky bucket if it does not incur any delay in a (σ, r) leaky bucket shaper. Then, since flow f conforms to the shape enforced by the shaper $\overline{S}_f$, $D_f^{\overline{S}} = 0$. However, since D_f^C is given as $\frac{(k-1)l_f}{r_f}$ [6], if $k > 1$, $D_f^C + \widehat{D}_f^{\overline{S}} > 0$. Hence, $\widehat{d}_f$ may be greater than d_f. Thus in case of RCSD, the end-to-end delay may increase due to the presence of the shaper.

Hence, if buffer requirement in the network is reduced to that of a CBR flow by employing a shaper at the network edge then:

- In GR networks the delay does not increase. Hence, increasing the available buffer space does not effect the achievable utilization.

- In RCSD networks the delay may increase. Hence, to ensure that end-to-end delay requirements of the flow are met, higher resources may have to be reserved which may result into loss in achievable utilization.

Hence, we conclude that the available buffer space does not effect the achievable utilization of GR networks but may effect that of RCSD networks. Therefore, the effectiveness of buffer in increasing the utilization of a network can be determined by evaluating the difference in achievable utilization of a network of RCSD and GR servers when RCSD servers employ additional buffers. Since RCSD requires additional buffers only when the shaper $\overline{S}_f$ is a non-CBR shaper, henceforth we will assume that non-CBR shapers are employed in RCSD.

To gain an intuitive understanding for the difference in achievable utilization of a network of GR and RCSD servers, let us first consider a single server scenario. Let the flows conform to leaky buckets, i.e., flow f conforms to a (σ_f, r_f) leaky bucket. Furthermore, for scheduling algorithms in GR, let $\alpha_f^i = \frac{l_f}{R_f} + \frac{l_{max}^i}{C^i}$ where C^i is the capacity of server i and l_{max}^i is the maximum length of packets served by server i (scheduling algorithms such as WFQ and Virtual Clock satisfy this assumption [6]). Since EDF has the largest schedulability region, let the RCSD scheduling algorithms employ EDF scheduler and a (σ_f, r_f) leaky bucket shaper for flow f.

Consider two flows f and m that conform to leaky bucket with parameters $(10pkt, 1pkt/s)$ and $(1pkt, 1pkt/s)$, respectively. Let both flows f and m require delay of $5.5s$ and be served by a single server with capacity $2pkts/s$. Then, from the results in [3], it can be shown that a RCSD server satisfies the deadline requirements of both the flows. On the other hand, in case of GR, from (2) and [6] we know that flow f would have to reserve at least rate R_f such that:

$$\frac{\sigma_f - l_f}{R_f} + \frac{l_f}{R_f} + \frac{l_{max}}{C} = 5.5 \tag{8}$$

Hence, $\frac{10}{R_f} + 0.5 = 5.5$ which yields $R_f = 2pkt/s$, which is the capacity of the server. Hence, the server would not be able to accept flow m. Thus, the achievable utilization is higher in case of RCSD.

Though achievable utilization is higher in case of a single server for RCSD, it is not clear whether the same result holds in a network of servers. To observe this , let flow f have an end-to-end delay requirement of d_f. Since we have assumed that the shaper employed is the same as the input traffic, $D_f^{\overline{S}} = 0$. Hence, from (3) we get:

$$d_f = \sum_{n=1}^{n=K} d_f^n \tag{9}$$

If we assume each of the server is equally loaded, then the delay of the flow at each server is $\frac{d_f}{K}$. Thus, *intuitively*, for a given end-to-end delay, the resource requirement at each server for a flow increases linearly with the number of servers on the path. On the other hand, in a network of GR servers, we know from (2) and [6] that the reserved rate $R_f \geq r_f$ should be such that:

$$d_f \geq \frac{\sigma_f - l_f}{R_f} + K\frac{l_f}{R_f} + \sum_{i=1}^{i=K} \frac{l_{max}^i}{C^i} \tag{10}$$

Then, if σ_f is large, i.e., when RCSD servers may be expected to have larger utilization, R_f increases sub-linearly with the number of servers on the path. Hence, intuitively, whereas resource requirement increases linearly with the number of servers in a RCSD network, it increases sub-linearly in a GR network. Consequently, RCSD networks will have higher utilization only if the reduction in resource requirement yielded by additional buffer is sufficient to offset the difference in the linear and sub-linear increase. Otherwise, RCSD networks may not have higher utilization and, contrary to intuition, may have lower utilization.

A theoretical analysis to determine the difference in achievable utilization of GR and RCSD networks has remained elusive. Hence, to determine the difference we conduct a large number of experiments. In the next section, we describe our experimental methodology and present the experimental results in Section 4.

3 Experimental Methodology

To experimentally determine the difference in achievable utilization of a network of GR and RCSD servers, we assume that all the flows conform to leaky bucket. (Since the traffic model used by the Internet and ITU standard bodies is leaky bucket, this is not a restrictive assumption.). Hence, flow f is characterized by the quintuple $(\sigma_f, r_f, l_f, d_f, K_f)$ where σ_f,r_f,l_f,d_f, and K_f denote the burstiness, rate, packet length, end-to-end delay requirement, and the number of servers on the path of flow f, respectively. We make the following assumptions for GR and RCSD networks:

- GR: We assume that for GR servers $\alpha_f^i = \frac{l_f}{R_f} + \frac{l_{max}^i}{C^i}$ where C^i is the capacity of server i and l_{max}^i is the maximum length of packets served by server i (we remove this restriction in Section 4.2). Hence, rate $R_f \geq r_f$ that would satisfy the end-to-end delay requirements of flow f can be determined by (10). We assume that R_f is reserved for flow f at each of the servers.
- RCSD: We assume that RCSD servers employ EDF scheduler and a (σ_f, r_f) leaky bucket shaper for flow

f. Furthermore, we assume each of the server is equally loaded and hence the delay of flow f at each server is $\hat{d}_f = \frac{d_f}{K_f}$.

We assume that the capacity of each server is C. The schedulability conditions that we employ for GR and RCSD are as follows:

- GR: We ensure that aggregate rate reserved by the flows at a server is less than the server capacity. Hence, if Q flows are served by a server, then the schedulability condition is $\sum_{i=1}^{i=Q} R_i \leq C$.
- RCSD: We use the schedulability condition presented in [2], which is a generalization of Theorem 1 of [3] for networks that have variable packet sizes, to determine the schedulability region. Specifically, flows $[1..Q]$ are schedulable if $\sum_{i=1}^{i=Q} r_i \leq C$ and the following set of inequalities hold:

$$\sum_{i=1}^{i=k} l_i + \sum_{i=1}^{i=k} (\sigma_i - l_i) + l_{max} \leq \hat{d}_k \left(C - \sum_{i=1}^{i=k-1} r_i \right) + \sum_{i=1}^{i=k-1} r_i \hat{d}_i \quad 1 \leq k \leq Q \quad (11)$$

Without loss of generality we have assumed that $\hat{d}_1 \leq .... \leq \hat{d}_Q$. If all the flows have the same packet length l, then we use Theorem 1 of [3] and ensure that the following set of inequalities hold:

$$\min\{(k+1)l, Ql\} + \sum_{i=1}^{i=k} (\sigma_i - l) \leq \hat{d}_k \left(C - \sum_{i=1}^{i=k-1} r_i \right) + \sum_{i=1}^{i=k-1} r_i \hat{d}_i \quad 1 \leq k \leq Q \quad (12)$$

To experimentally evaluate the difference in achievable utilization between RCSD and GR networks, we first assume that there are only two types of flow at the servers: low throughput and high throughput flows (experimental methodology for heterogeneous flows is presented in Section 4.3). Let $r_l < r_h$ and the quintuples $(\sigma_l, r_l, l_l, d_l, K_l)$ and $(\sigma_h, r_h, l_h, d_h, K_h)$ characterize the low and high throughput flows, respectively. We employ the schedulability conditions presented above to compute the schedulability region for RCSD and GR servers. The schedulability region is given by a set of tuples (n_h, n_l) such that n_h and n_l are the maximum number of high and low throughput flows, respectively, that can be admitted simultaneously (Figure 2 shows a hypothetical schedulability region for RCSD and GR servers). To compare schedulability regions of GR and RCSD and hence determine the difference in achievable utilization between GR and RCSD, we define two metrics: *average utilization gain* and *maximum utilization gain*. To define these metrics, we first define the functions $U^r(i)$ and $U^g(i)$ that compute the achievable utilization of a RCSD and GR server, respectively, when i high throughput flows are admitted.

Let $L^r(i)$ and $L^g(i)$ be the maximum number of low throughput flows that are admitted in RCSD and GR when i high throughput flows are admitted, respectively. Also, let n_h^r and n_h^g be the maximum number of high throughput flows that can be admitted by RCSD and GR server, respectively. We need to define $U^r(i)$ and $U^g(i)$ for $i \in [1..N]$ where $N = \max\{n_h^g, n_h^r\}$. Consider the definition of $U^r(i)$. Clearly, when $i \leq n_h^r$, $U^r(i)$ is:

$$U^r(i) = \frac{r_h i + r_l L^r(i)}{C} \quad i \leq n_h^r \quad (13)$$

If $n_h^r \leq n_h^g$, then we will require $U^r(i)$ for $i > n_h^r$ also. In such a case, the number of high throughput flows that should be used in determining utilization is n_h^r. Furthermore, since $L^r(i)$ is not defined for $i > n_h^r$, $L^g(i)$ should be chosen as the number of low throughput flows. However, since at most $L^r(n_h^r)$ low throughput flows may be admissible along with n_h^r high throughput flows, we conclude that $\min\{L^r(n_h^r), L^g(i)\}$ number of low throughput flows should be used in the computation of $U^r(i)$ when $i > n_h^r$. Hence, we get:

$$U^r(i) = \frac{r_h n_h^r + r_l \min\{L^r(n_h^r), L^g(i)\}}{C} \quad n_h^r < i \leq N \quad (14)$$

The function $U^g(i)$ is defined analogously. Note that in the definition of $U^g(i)$, the actual rates of the flows, and not the reserved rates, are used.

We now define the two metrics as follows:

- *Average utilization gain*: This metric, denoted by U, is defined to capture the expected increase in utilization yielded by a RCSD server. It is defined as:

$$U = \frac{1}{N} \left(\sum_{i=1}^{i=N} \frac{U^r(i) - U^g(i)}{U^g(i)} \right) \quad (15)$$

- *Maximum average utilization gain*: This metric, denoted by U^+ is defined to capture the maximum expected increase in utilization yielded by a RCSD server. It is based on the observation that since GR class is a proper subset of RCSD class, it is always possible for a RCSD server to achieve the utilization of a GR server. Hence, for determining maximum expected increases in utilization, any loss in utilization yielded by RCSD may be ignored. Specifically, it is defined as:

$$U^+ = \frac{1}{N} \left(\sum_{i=1}^{i=N} \frac{\max\{0, (U^r(i) - U^g(i))\}}{U^g(i)} \right) \quad (16)$$

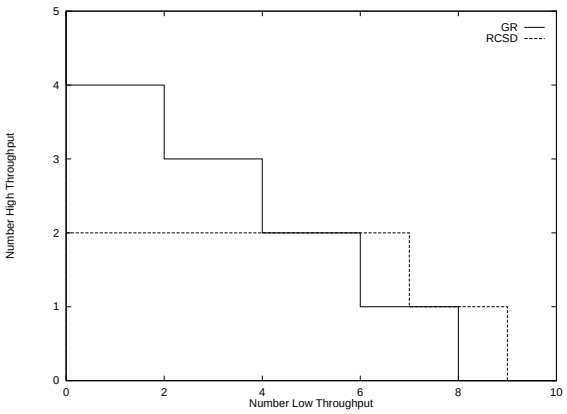


Figure 2: Example Schedulability Regions for GR and RCSD

We are now ready to define our experimental methodology. In an experiment, we choose a set of low and high throughput flows. For each pair of low and high throughput flows chosen from their respective sets, we determine the schedulability region in RCSD and GR. For each pair, we compute U and U^+. Thus, we get a set of values for the two metrics. We then determine a cumulative distribution function for the metrics and employ it to determine the difference in achievable utilization between GR and RCSD servers. In the next section, we present and analyze the results of some of the experiments that we have conducted.

4 Experimental Evaluation

4.1 Basic Experiments

We construct several sets of low and high throughput flows and choose several of their combinations. In a particular set, all the flows have equal number of servers on their path. Hence, let $S_l^{i,k}$ and $S_h^{i,k}$ denote the i^{th} low and high throughput set, respectively, with all flows having k servers on their path. Let l_l and l_h be the length of low and high throughput flow packets, respectively. Also, let Σ_l, Φ_l, and D_l^i be set of burstiness, rate, and delay values for the low throughput flows. Then, the set $S_l^{i,k}$, $i \in [1..3]$, is defined as follows:

$$S_l^{i,k} = \{(\sigma, r, d, l_l, k) |\ \sigma \in \Sigma_l \wedge r \in \Phi_l \wedge d \in D_l^i\} \quad (17)$$

The set $S_h^{i,k}$ is defined analogous to $S_l^{i,k}$. Observe that flows in set $S_l^{1,k}..S_l^{3,k}$ differ only in their delay values. The sets Σ, Φ, and D^i for low and high throughput flows are defined in Table 1. (The delay and throughput values for the low and high throughput flows are chosen so that they correspond to audio and video flows, respectively). We let the number of servers on the path be either $1, 3, 5, 10,$ or 15. Note that since the average number of hops in Internet is 17, the number of servers we choose is conservative [10]. We conduct three experiments:

- EXP1: $S_l^{1,k}$ and $S_h^{1,k}$ are chosen to be the low and high throughput sets. In this case, some flows in the low throughput set have higher delay requirement than the flows in the high throughput set, and vice versa.
- EXP2: $S_l^{2,k}$ and $S_h^{2,k}$ are chosen to be the low and high throughput sets. In this case, flows in the low throughput set always have lower delay requirement than the flows in the high throughput set.
- EXP3: $S_l^{3,k}$ and $S_h^{3,k}$ are chosen to be the low and high throughput sets. In this case, flows in the high throughput set always have lower delay requirement than the flows in the low throughput set.

We assume that the capacity of each server is $50Mb/s$ and conduct these experiments for network environments with fixed as well as variable size packets (leading to the computation of 52650 schedulability regions).

- Networks with fixed packet size: We choose $l_l = l_h = 512$ bits. Figures 3 and 4 plot the cumulative distribution function for the metrics U and U^+, respectively, for different values of k. Figure 3 shows that $70 - 80\%$ of the time, average gain is less than 0, and it is at most 0.15 in all the experiments when $k > 1$. Figure 4 shows that maximum average gain also is at most 0.15.
- Networks with variable packet size: In networks with variable packet size, we expect low throughput applications (such as audio) to have small packet size and high throughput applications such as video to have large packet sizes. Hence, we choose $l_l = 512$ bits and $l_h = 8000$ bits. Figures 5 and 6 plot the cumulative distribution function for the metrics U and U^+, respectively, for different values of k. Figure 5 shows that $85 - 90\%$ of the time average gain is less than 0 in all the experiments when $k > 1$. Figure 6 shows that 90% of the time the maximum average gain is less than 0.02 in all the experiments when $k > 1$. It also shows that sometimes, the maximum average gain is as high as 1.

From the above experiments, we conclude, for both type of networks, that: (1) as per the intuition presented in Section 2, the average gain and maximum average gain decrease with increases in the number of servers on the path; (2) $70 - 80\%$ of the time, the average gain is less than 0; (3) $85 - 90\%$ of the time, the maximum average gain is less

Type	Burstiness	Rate	Delay		
	Σ	Φ (Kb/s)	D^1 (ms)	D^2 (ms)	D^3 (ms)
Low	$\{l_l, 10l_l, 20l_l\}$	$\{32, 64, 128\}$	$\{25, 50, 75, 100, 200\}$	$\{25, 50, 75, 100\}$	$\{100, 125, 150, 175, 200\}$
High	$\{l_h, 10l_h, 20l_h\}$	$\{640, 1280, 2560\}$	$\{25, 50, 75, 100, 200\}$	$\{100, 125, 150, 175, 200\}$	$\{25, 50, 75, 100\}$

Table 1: Parameters for the experiments

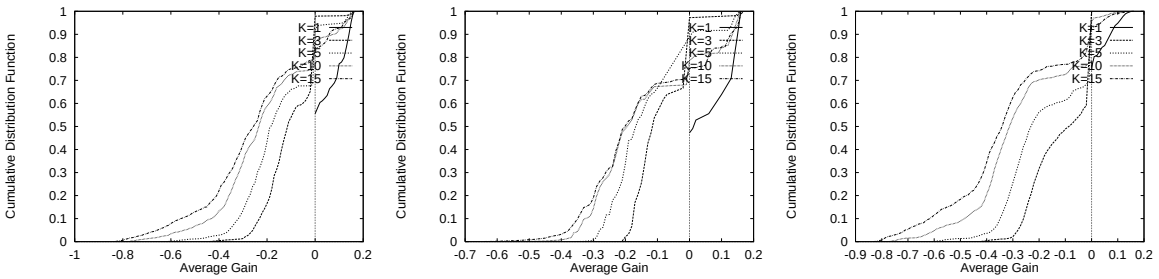


Figure 3: Fixed size packet networks: Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

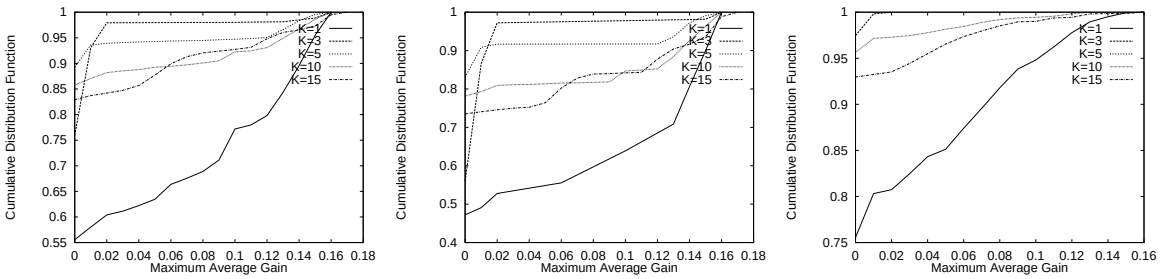


Figure 4: Fixed size packet networks: Maximum Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

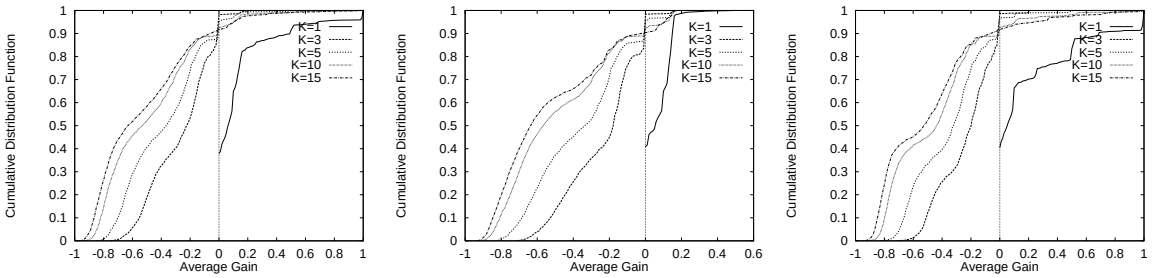


Figure 5: Variable size packet networks: Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

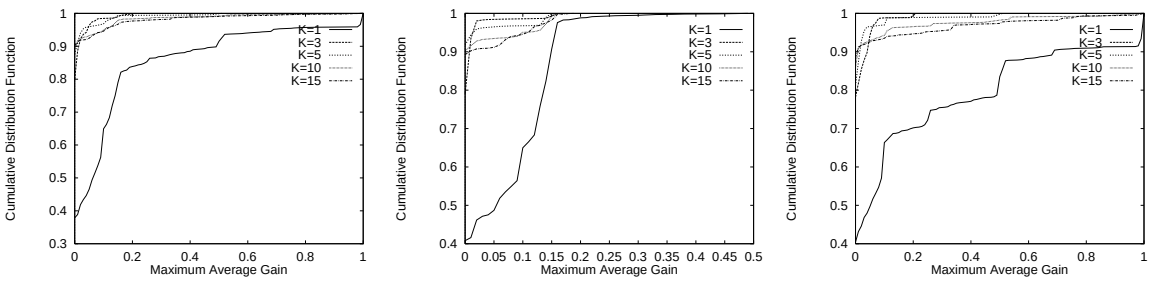


Figure 6: Variable size packet networks: Maximum Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

than $0.02 - 0.03$; and (4) $10 - 20\%$ of the time, the maximum average gain is non-negligible.

A further examination of the experimental results for the $10 - 20\%$ cases, in which the maximum average gain was non-negligible, demonstrated that most of the significant increase in utilization of RCSD occurred when both the low and high throughput flows have burstiness equal to the packet length, i.e., the flows are CBR. Since theoretically the achievable utilization of GR and RCSD should be same when all the flows are CBR, this seems to be a contradictory result. However, this is only an apparent contradiction.

To observe this, recall that we had restricted the GR class to algorithms for which α^i was given as $\alpha_f^i = \frac{l_f}{R_f} + \frac{l^i_{max}}{C^i}$. This restriction included algorithms that allocate only rate (e.g., WFQ, Virtual Clock). However, GR class contains algorithms such as Delay EDD that separate rate and delay allocation and can have $\alpha_f^i < \frac{l_f}{R_f} + \frac{l^i_{max}}{C^i}$, or $\alpha_f^i > \frac{l_f}{R_f} + \frac{l^i_{max}}{C^i}$. Such algorithms have higher achievable utilization than the algorithms that allocate only rate. The unrestricted GR class that contains Delay EDD has the same achievable utilization as the RCSD class when the flows are CBR. Thus, to determine whether the non negligible increase in utilization of RCSD server is due to the additional buffer employed by it or the restriction that we had placed on GR, in the next section, we repeat the experiments with the unrestricted GR class.

4.2 Experiments with Unrestricted GR Class

In the unrestricted GR class, the reserved rate $R_f \geq r_f$ that will meet the end-to-end delay requirement of flow f can be derived using:

$$d_f = \frac{\sigma_f - l_f}{R_f} + K\hat{d}_f \tag{18}$$

where we have assumed that $\alpha_f^i = \hat{d}_f$. Observe that there are several combinations of R_f and $\hat{d}_f$ that will satisfy (18). Hence, to determine both R_f and $\hat{d}_f$, we need one more constraint.

To develop this constraint, we will assume that the scheduling algorithm at each of the servers is Delay EDD. The schedulability conditions for Delay EDD for variable size and fixed size packet networks are given by (11) and (12), respectively, with l_f and R_f substituted for σ_f and r_f, respectively. From an examination of the schedulability conditions, we conclude that neither very high value of R_f nor a very small value of $\hat{d}_f$ are desirable. Hence, to balance these two tradeoffs we choose minimum value of $R_f \geq r_f$ that will satisfy (18) while ensuring that:

$$\hat{d}_f \geq \epsilon \frac{l_f}{R_f} \tag{19}$$

where $\epsilon \leq 1$.

We determine the schedulability region for Delay EDD as follows. Since the set of flows that are schedulable by rate only allocation algorithms are also schedulable by Delay EDD, we first determine a tuple (n_h, n_l) as in Section 3 for GR. We then choose $\epsilon = 0.1, 0.3, 0.5, 0.7$, and 0.9 and determine the maximum number of low throughput flows, $\hat{n}_l \geq n_l$, that can be accepted along with n_h high throughput flows. This procedure when repeated for all values of n_h gives the schedulability region for Delay EDD.

We repeat the experiments presented in Section 4.1 to compare the achievable utilization in GR and RCSD. We find that the average gain becomes less than 0 in all the experiments for $k > 1$. Hence, we only plot the maximum average gain for fixed and variable length packet networks in Figures 7 and 8, respectively. As the figures demonstrate, the maximum average gain reduces significantly for $k > 1$. It is always less than 0.01 for fixed size packet networks and 0.05 for variable size packet networks.

We repeat all the experiments that we have presented so far by increasing all the delay values by 100ms. Figures 9 and 10 plot the maximum average gain for fixed size and variable size packet networks, respectively. As the figures demonstrate, there is no significant change in the results.

4.3 Experiments with Randomly Generated Flows

In our experiments so far, we have assumed that there are only two types of flows at a server. In this section, we determine the difference in achievable utilization when the flows have heterogeneous characteristics. To achieve this objective, we generate a list of flows that have heterogeneous characteristics and then determine the subset of flows accepted in RCSD and GR using two methods:

1. We consider flows for acceptance in sequential order and determine the subset of flows that are accepted by a RCSD server using the schedulability conditions presented before. We then determine the subset of flows accepted by a GR server in two steps:
   - We determine the subset of the accepted flows that can be scheduled by a GR server using the schedulability conditions for Delay EDD and $\epsilon = 0.1, 0.3, 0.5, 0.7$, and 0.9.
   - If all the flows accepted by RCSD are accepted by GR, we determine the subset of the remaining flows that can be accepted by GR.

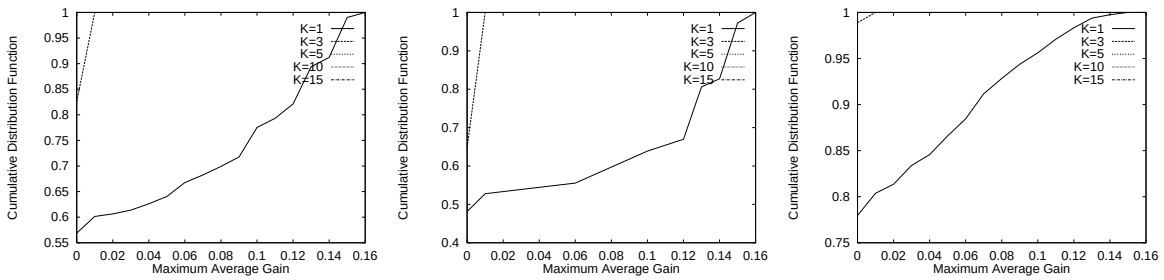


Figure 7: Fixed size packet networks: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

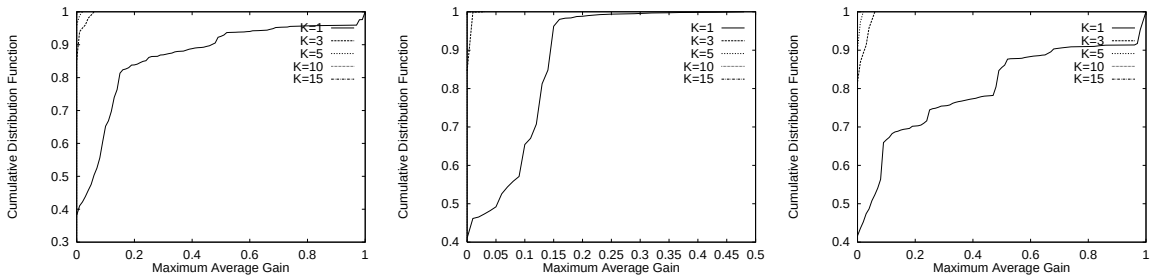


Figure 8: Variable size packet networks: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

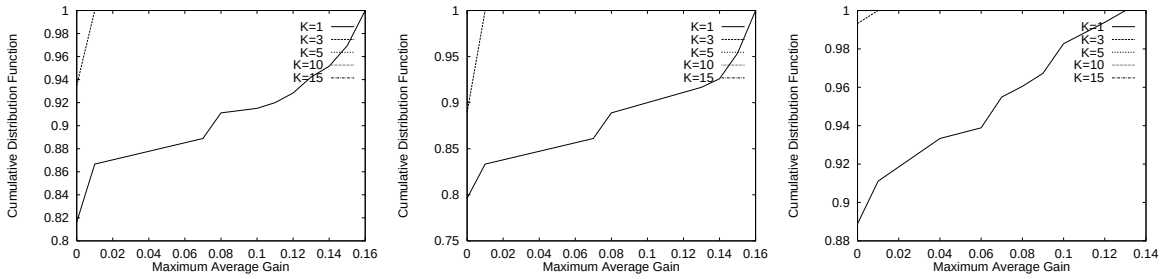


Figure 9: Fixed size packet networks with delay increased by 100 ms: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

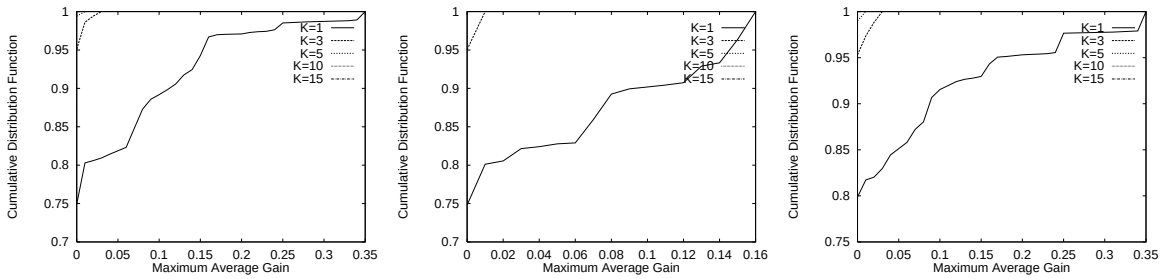


Figure 10: Variable size packet networks with delay increased by 100 ms: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

Let Q^r and Q^g be the set of accepted flows in RCSD and GR, respectively. Define utilization of RCSD and GR servers, denoted by u^r and u^g respectively, as $u^r = \frac{1}{C}\sum_{i \in Q^r} r_i$ and $u^g = \frac{1}{C}\sum_{i \in Q^g} r_i$ where C is the link capacity. Then RCSD utilization gain, U^r is defined as: $U^r = \frac{u^r - u^g}{u^g}$.

2. This method is the same as the previous with the only change that roles of GR and RCSD are reversed. In this case, we define GR utilization gain, U^g as: $U^g = \frac{u^g - u^r}{u^r}$.

In an experiment, we generate a large number of different list of flows, determine U^r and U^g for each list, and determine the cumulative distribution function of both the metrics. The lists of flows are generated as follows. We choose range of burstiness factor $[b_{min}, b_{max}]$, packet lengths $[l_{min}, l_{max}]$, rate $[r_{min}, r_{max}]$, delay $[d_{min}, d_{max}]$, and number of servers on the path $[k_{min}, k_{max}]$. We then generate the quintuple $(\sigma_f, r_f, l_f, d_f, K_f)$ for a flow by choosing r_f, l_f,d_f, and K_f from their respective ranges with uniform probability. We choose burstiness factor b_f uniformly from its range and set $\sigma_f = b_f l_f$. A list of flows consists of such randomly generated flows and has at most one flow more than that required to ensure that the aggregate rate of the flows in the list is at least that of the link capacity.

We conduct 6 experiments and generate $100,000$ lists in each experiment. The ranges for all the factors other than delay are kept same in all the experiments. The ranges that we choose are: $[b_{min}, b_{max}] = [1, 20]$, $[l_{min}, l_{max}] = [512, 8196]$ bits, $[r_{min}, r_{max}] = [32, 2056]$ Kb/s, and $[k_{min}, k_{max}] = [3, 15]$. The six different ranges that we choose for delay are: $[25, 200]$ ms, $[100, 200]$ ms, $[100, 300]$ ms, $[100, 400]$ ms, $[100, 500]$ ms, and $[100, 600]$ ms. Figures 11(a) and 11(b) plot the metrics U^r and U^g for these experiments, respectively. As both the figures demonstrate, in most cases, RCSD has lower utilization than GR, but the loss decreases with increase in the upper bound on delay. Observe that the distribution of the gain in utilization of RCSD is similar to the experiments conducted in the previous section. Hence, it appears that the results are not very correlated with the particular choice of parameters.

5 Discussion

The experiments in the previous section demonstrate that, in most cases, RCSD networks that employ non-CBR shapers have lower achievable utilization than GR networks. In few cases that they have higher utilization, the increase is small. Since RCSD networks with non-CBR shapers have higher buffer requirement than GR networks, we conclude that additional buffer does not increase the achievable utilization of a network providing deterministic guarantees. Furthermore, contrary to intuition, use of additional buffer may decrease the achievable utilization significantly.

It may be argued that since additional buffer does increase utilization sometimes, although by very small percentage, it may be used only when it increases the utilization and not otherwise. However, an algorithm for determining when to use additional buffers such that utilization always increases is not known. Furthermore, in many of our experiments in the previous section, the schedulability region is similar in structure to the hypothetical example shown in Figure 2. In such a case, the utility of additional buffer is dependent not only on the characteristics of the flows but also the specific combination of flows that have to be supported. Thus, it may be difficult to determine utility of additional buffers in a network where the set of flows may change dynamically.

In our experiments, we have focussed on the gain in maximum achievable utilization of a network. Though buffer does not help in increasing the achievable utilization, it may be argued that it may improve other measures of network performance such as call blocking probability. However, our experiments indicate that this is unlikely as most (99% or higher) of the times, the maximum average utilization gain is 0. Unless the flow arrival process is such that the network operates most of the time in the small region where the maximum average utilization gain is non-zero, buffer would not yield reduced call blocking probabilities.

Though additional buffer at the servers does not increase achievable utilization when a network provides deterministic guarantees, it may be desirable for other reasons. If the burstiness of the sources is not smoothed at the edge and additional buffer is employed at the switches to absorb the burstiness, then due to the resultant overall work conserving nature of the systems, the packets of sources will experience smaller average delay. Furthermore, a network may be able to achieve better statistical multiplexing of best effort packets and thereby achieve higher overall throughput [12]. Finally, a network that supports deterministic service as well as a service that provides guaranteed throughput, but not delay, may be able to employ additional buffer for deterministic service to increase its overall utilization [2].

6 Concluding Remarks

A theoretical investigation to determine whether source traffic should be completely smoothed or not when a network provides deterministic guarantees has been carried

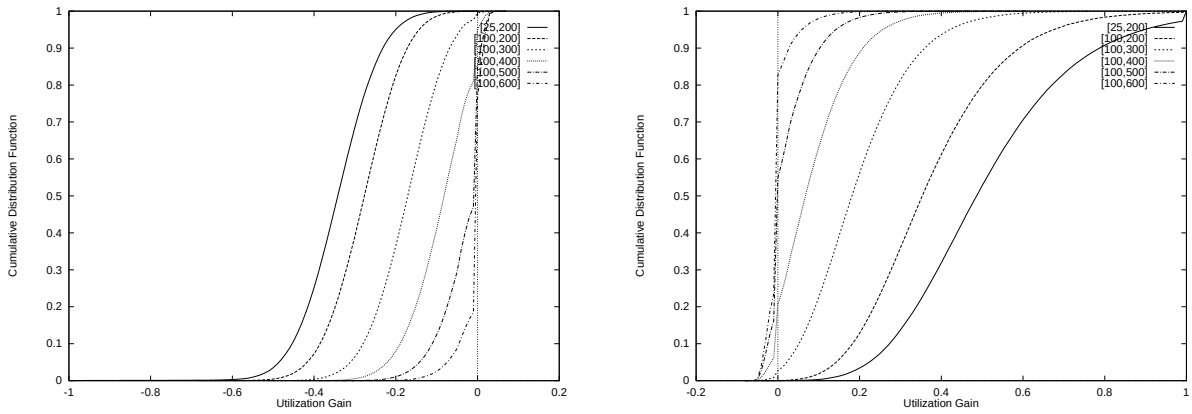


Figure 11: Randomly generated flows: (a) U^r (b) U^g

out in [5]. However, [5] makes several simplifying assumptions (for example, the differences in the delay requirements of flows are not accounted for) that make it inapplicable to the more general problem that we have investigated. We are not aware of any other theoretical or experimental study that investigates the utility of buffer in a network providing deterministic guarantees.

In this paper, we took a step towards determining the effectiveness of buffer in increasing achievable utilization of networks providing deterministic guarantees. To determine the increase, we classified the packet scheduling algorithms as either belonging to the class of Guaranteed Rate (GR) scheduling algorithms or Rate Controlled Service Disciplines (RCSD). We experimentally determined the difference in achievable utilization of RCSD and GR servers when RCSD servers employ additional buffers. Our experiments demonstrated that *contrary to intuition*, in most cases, additional buffers in a network of RCSD servers do not lead to higher achievable utilization. This experimental result indicates that if a source desires deterministic bounds on packet delay, then the traffic should be completely smoothed at the source [9].

Though our experiments explored large parameter space (more than 100,000 schedulability regions and 700,000 sets of flows were generated), they are not exhaustive. Hence, they should be considered as the first step in evaluating effectiveness of buffer in increasing achievable utilization of networks providing deterministic guarantees. We plan to extend this study in several ways. For example, we plan to evaluate the effectiveness of buffer for deterministic guarantees when the sources are characterized by models such as D-BIND [11].

References

[1] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of ACM SIGCOMM'89*, pages 1–12, September 1989.

[2] L. Georgiadis, R. Guerin, V. Peris, and R. Rajan. Efficient Support of Delay and Rate Guarantees in an Internet. In *Proceedings of ACM SIGCOMM'96*, pages 106–116, August 1996.

[3] L. Georgiadis, R. Guerin, V. Peris, and K.N. Sivarajan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. In *Proceedings of INFOCOM'96*, pages 102–110, March 1996.

[4] S.J. Golestani. A Self-Clocked Fair Queueing Scheme for High Speed Applications. In *Proceedings of INFOCOM'94*, pages 636–646, April 1994.

[5] S. Gorinsky, S. Baruah, and A. Stoyenko. Traffic Reshaping in Packet-Switched Virtual-Circuit Fixed-Packet Networks. In *Proceedings of Workshop on Resource Allocation Problems in Multimedia Systems*, December 1996. Available via URL: http://www.cs.unc.edu/ jeffay/mm-wrkshp96.html.

[6] P. Goyal, S. S. Lam, and H. M. Vin. Determining End-to-End Delay Bounds In Heterogeneous Networks. *ACM/Springer-Verlag Multimedia Systems Journal*, 5(3):157–163, May 1997. Also appeared in the Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video, Pages 287-298, April 1995.

[7] P. Goyal and H. M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. In *IEEE/ACM Transactions on Networking*, August 1997.

[8] P. Goyal, H. M. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM'96*, pages 157–168, August 1996.

[9] M. Grossglauser, S. Keshav, and D. Tse. RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic. In *Proceedings of ACM SIGCOMM'95*, pages 219–230, August 1995.

[10] J. D. Guyton and M. F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *Proceedings of ACM SIGCOMM'95*, pages 288–298, August 1995.

[11] E. Knightly and H. Zhang. Traffic Characterization and Switch Utilization Using a Deterministic Bounding Interval Dependent Traffic Model. In *Proceedings of INFOCOM'95, Boston, MA*, April 1995.

[12] S. Shenker. Contribution to the Int-serv mailing list. July 1995.

[13] D. Stiliadis and A. Varma. Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet Switched Networks. In *Proceedings of SIGMETRICS'96*, May 1996.

[14] S. Suri, G. Varghese, and G. Chandramenon. Leap Forward Virtual Clock: A New Fair Queuing Scheme with Guaranteed Delays and Throughput fairness. In *Proceedings of INFOCOM'97*, April 1997.

[15] G.G. Xie and S.S. Lam. Delay Guarantee of Virtual Clock Server. *IEEE/ACM Transactions on Networking*, 3(6):683–689, December 1995.

[16] H. Zhang and D. Ferrari. Rate-controlled Service Disciplines. *Journal of High Speed Networks*, 3(4):389–412, 1994.

[17] H. Zhang and S. Keshav. Comparison of Rate-Based Service Disciplines. In *Proceedings of ACM SIGCOMM*, pages 113–121, August 1991.

[18] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proceedings of ACM SIGCOMM'90*, pages 19–29, August 1990.

Design, Implementation, and End-to-End Evaluation of a Measurement-Based Admission Control Algorithm for Controlled-Load Service

Cheng Jin Sugih Jamin*
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109-2122
{*chengjin,jamin*}*@eecs.umich.edu*

May 29, 1998

Abstract

The Internet Engineering Task Force (IETF) is considering the adoption of controlled-load service as a standard for relaxed real-time services on the Internet. The ability of relaxed real-time services to achieve high level of utilization and still provide acceptable quality of service depends crucially on the use of measurement-based admission control (MBAC) algorithms. MBAC algorithms admit new connections based on measured bandwidth usage of existing connections. In this paper we describe the design and implementation of an MBAC algorithm for controlled-load service on a router running FreeBSD 2.2.2 kernel. We then present an end-to-end evaluation which shows that controlled-load service, supported by an MBAC, reduces the packet loss rate and delay variation of real-time traffic while maintaining network efficiency.

1 Introduction

With the growing popularity of the Internet, its use as a medium for real-time audio and video traffic has seen a rapid rise. Currently the Internet offers only a single level of best-effort (BE) service, which, though adequate for traditional traffic such as email and file transfer, does not provide service guarantees to preserve the fidelity of real-time audio and video sessions. While it has been largely accepted that real-time applications on the Internet must adapt to available bandwidth [AMK97], and adjust their quality accordingly, efforts to quantify the effect of network congestion on user-perceived quality has seen limited results. The authors of [MKT95, RKTS94] studied the design of adaptive algorithms given a packet stream with varying inter-arrival times. Aside from [BBS98] and the present paper, we are not aware of any other work that studied the effect of network design on adaptive playback applications.

Two parallel efforts are on-going at the Internet Engineering Task Force (IETF) to introduce service differentiation to the Internet that could potentially preserve the fidelity of real-time sessions in times of network congestion. *Differentiated Services* aims to allow network providers to sell virtual links of guaranteed rate ("premium service") to organizations such as other providers or institutional customers [CW97, NJZ97]; *Integrated Services* aims to allow per-flow negotiation of quality of service on a shared link. The integrated services working group at the IETF is working towards the standardization of two types of real-time service: Guaranteed Service [SPG97] and Controlled-Load (CL) service [Wro97]. In this paper we consider the use of CL service in a differentiated-services environment, for example, when an institution purchases a virtual link under the differentiated services archi-

*This research is supported in part by funding from MCNC/MCI and NSF Career Award ANI-9734145 and by equipment donation from Digital Equipment Corp., Sun Microsystems Inc., and Intel Corp.

tecture to carry its CL traffic.

Controlled-load service provides a relaxed real-time service with low loss and low delay without guaranteeing a delay bound. Given a fixed rate allocated to serve controlled-load traffic, admission control must be employed to control admittance to the service. A simple algorithm that ensures the sum of reserved rates to always be below allocated rate is an effective, though not always efficient, admission control algorithm. We refer to this as the Simple Sum Admission Control algorithm (SSAC) in the remainder of this paper. A Measurement-Based Admission Control algorithm (MBAC) controls admittance based on the declared rate of a flow and measured usage of existing flows. In this paper we show that the application of MBAC to control access to a virtual link purchased under the differentiated services architecture allows CL traffic to take advantage of statistical multiplexing, increasing utilization of the virtual link, without perceptible degradation in the quality of service received.

Our previous work on MBAC support for CL service has concentrated on studying the utilization gain achievable compared against that under the simple sum algorithm. In this paper, we look at the end-to-end service seen by adaptive real-time applications subscribing to CL service when an MBAC algorithm is used. We show that CL service with MBAC has the following characteristics:

- Controlled-load traffic receives "better" service than best-effort traffic, by which we mean that CL traffic sees lower packet loss rate and delay variation.
- When admittance to controlled-load service is controlled by a properly working admission control algorithm, best-effort traffic does not experience excessive loss in the presence of controlled-load traffic.
- MBAC allows higher utilization of the bandwidth allocated to controlled-load traffic than under SSAC with comparable application-perceived service in terms of loss rate and delay variation.

Extant studies of measurement-based admission control algorithm have mostly been based on formal and numerical analysis or simulations. All the results reported in this paper are from experiments with our in-kernel implementation of Measurement-based Admission control for Controlled-load sErvice (MACE). We describe the design and implementation of MACE in Section 3. In Section 4 we describe our experimental setup for results reported in Section 5. In both Sections 3 and 4 we describe some caveats on the implementation of MACE and the experimental setup on a network testbed. MACE has been implemented on Intel's PentiumPro machines running FreeBSD 2.2.2. We will make this implementation available on the Web. We close this paper by summarizing our results and lessons learned in Section 6. We start our discussion with an overview of current real-time adaptive playback applications in the next section.

2 Adaptive Playback of Real-time Applications

A typical real-time session today involves transmitting digitized audio/video samples from senders across the Internet to receivers. The current real-time applications send packets only when new data, e.g. human voice and body movement, is detected. Consequently, real-time packets tend to arrive at a receiver in bursts. We call the time between when a packet is transmitted by the sender and when it is delivered ("played back") to the user, the packet's *playback delay*. Without buffering at either the sender or receiver, a packet's playback delay is then the transmission, propagation, and queueing delays the packet sees on the network. To ensure smooth playback of a burst of packets, receivers can buffer incoming packets and smooth out variations in delay (jitter) that packets experience in the network by delaying the playback of the leading packet of the burst. Adaptive playback applications change the playback point for each burst of incoming packets, i.e. based on observed packet delay variance, the playback algorithm can adjust the playback delay of the first packet of the incoming burst.

It has become widely accepted that good programming practices of real-time Internet applications should incorporate some form of adaptive playback [AMK97]. The adaptive nature of such

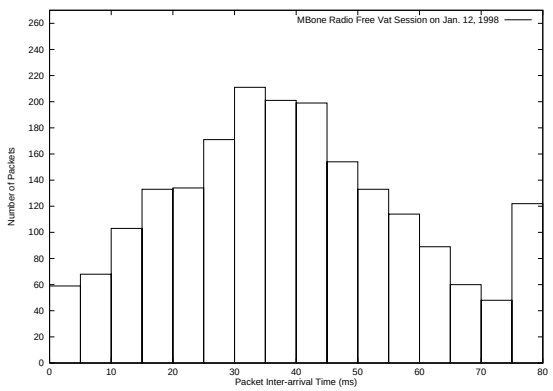


Figure 1: Vat Session with Poor Audio Quality.

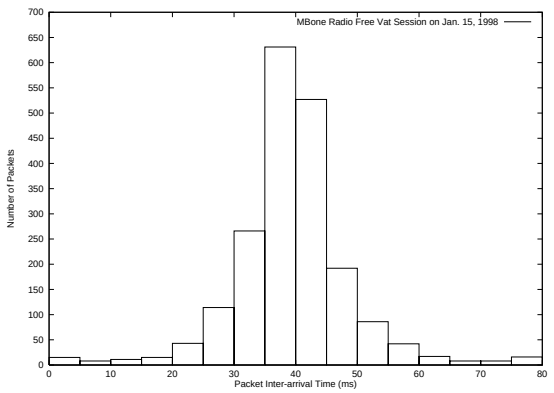


Figure 2: Vat Session with Better Audio Quality.

applications makes the specification of a metric to quantify the effect of network congestion on their performance that much harder to define. In [BBS98], the authors compute the *utility function* of adaptive applications as a function of delay variation. Users of an adaptive application gain utility from the use of the application if delay variation stays below a certain user-specified subjective threshold. Users get lower utility from signal reconstructed from packets with delay variation beyond this threshold. Delay variation aside, packet loss also effects signal reconstruction. Adaptive applications can tolerate some loss in their data stream; however there is no consensus to how much loss these adaptive applications can tolerate, though a low loss rate is generally desirable.

Figure 1 shows a trace of 2000 packets from a real-time multicast audio session on the MBone[1] captured with the `tcpdump` packet capture tool. The session captured was a continuous 64kbps stream of 320-byte UDP packets (excluding protocol headers) generated at a fixed 40ms inter-arrival time. The observed per-packet delay at the receiver is highly variable, as shown by the wide spread of the inter-arrival times in the figure. While the audio tool used, *vat*, was able to adapt and play the received audio stream, we subjectively found the quality of the resulting audio unsatisfactory. Figure 2 shows another trace of packets from the same audio session captured at a later date. Note that the delay variation has a much tighter spread compared to that in Figure 1. The tighter inter-arrival times enabled a much smoother playback of the received audio stream.

In the remainder of this paper, we study the end-to-end delay variation and loss rate seen by various flows under different network resource allocations. Lacking a quantitative metric to judge the services received, we can only provide comparative statements on how the delay distribution received under one service is tighter than that under another service. Given the subjective nature of perceived application performance, we do not make any claim that the difference in delay variation seen will translate into perceptible application performance.

3 Design and Implementation

This section describes in detail the design and implementation of our Measurement-based Admission control for Controlled-load sErvice module (MACE) in FreeBSD 2.2.2 kernel. Before we dive into the detail of MACE implementation, we first list our implementation guidelines:

- MACE implementation should provide only minimal service necessary to give priority to CL class traffic. We do not optimize our design for any particular transport protocol or specific application.
- MACE implementation should be modular such that each of its components can be replaced by a different implementation.
- The presence of MACE should be transparent to the rest of the system.

[1]The MBone is a virtual network on the Internet that supports multicast delivery.

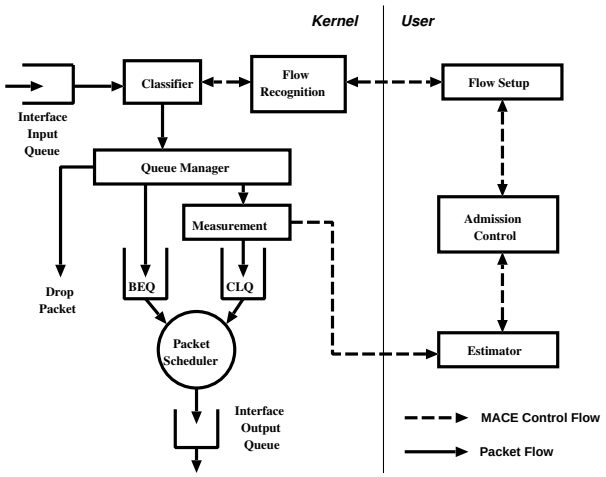


Figure 3: Overview of the Implementation

Figure 3 shows the schematic of our MACE implementation, which includes eight components. Four components: the packet classifier, the queue manager, the measurement unit, and the packet scheduler must be incorporated into the kernel because they are in the critical path of packet forwarding. The flow recognition unit promotes flows from BE class to CL class. It examines each packet that is not yet classified as a CL packet and promotes a flow if certain criteria are met. The flow recognition unit is placed in kernel to minimize the per packet processing time. The other three components, the flow setup unit, the estimator, and the admission control unit may be implemented as user level processes.

We focus our design on the output end of the packet forwarding path of a router. A two level priority queue is implemented to support CL service in addition to BE service available in the traditional one queue implementation. MACE implementation classifies and stores CL packets into a separate storage from BE packets. MBAC is applied to limit the aggregate CL traffic bandwidth usage to the allocated value. Our implementation does not use an explicit reservation protocol, e.g. RSVP for flow setup signalling; rather, network administrators use the flow setup unit to specify the criteria for flow setup. A per flow state is maintained for each recognized flow as in RSVP.

In the following paragraphs, we give a detailed description of each component of MACE.

Packet Classifier. The packet classifier determines the type of service each incoming packet receives. The packet classifier matches each incoming packet against a set of flow descriptors in the classifier table. If a match is found, the packet is considered a CL packet; otherwise, it is considered a BE packet. For the purposes of this paper, we use a single hash table for flow descriptor lookups. Assuming the network carries only IPv4 packets, we use the <source address, source port, destination address, destination port> fields of the IPv4 packet header as our flow descriptor. Our implementation allows for the use of the other fields in the IPv4 header, but they are not considered in this paper.

Queue Manager. The queue manager maintains a two-level strict priority queue, with CL traffic at the higher level. CL packets are therefore isolated and protected from BE packets.

The current FreeBSD 2.2.2 kernel implements a single interface queue of 50 packets for each Ethernet interface; we replace this single interface queue by our two-level priority queue. The total size of the priority queue is 50 packets. In the absence of CL traffic, a router with MACE implementation behaves exactly like a traditional router providing only BE service. When CL traffic is present and the MACE queue is filled, incoming BE packets are dropped unconditionally. CL packets are not dropped until the number of CL packets in the MACE queue reaches 50. We originally implemented a system with separate CL and BE queues of 50 entries each. This design was abandoned because the comparison is unfair in the sense that the queue size was twice that of the original FreeBSD kernel.

Measurement. The measurement unit keeps a counter for the number of bits transmitted on each network interface. Periodically, an average bandwidth usage is computed from this counter, and the counter is cleared. The counter is updated whenever a new packet is enqueued. Counting when packets are enqueued gives us the arrival rate, which could be higher than the service rate of the outgoing link. This makes MACE unit more conservative.

Packet Scheduler. The scheduler ensures that CL packets always get priority over BE packets. The packet scheduler is called by the device independent network output function and the device interrupt handler routine. As long as there are CL packets queued, the scheduler will dispatch the packet at the head of the CL queue; otherwise the packet at the front of the BE queue is dispatched only when the output interface queue is empty.

Flow Recognition. The flow recognition unit detects potential CL connections and attempts to add each one to the flow descriptor table of the classifier. Aside from the table used by the classifier, the flow recognition unit also maintains a separate flow recognition table to keep track of potential CL connections. This flow table is similar to the classifier flow table, with an additional packet counter field used to record the number of packets sent by a CL candidate. When the packet counter exceeds a predefined threshold, the flow recognition unit attempts to move the new flow to the CL service class. At this point, admission control must be performed to ensure that the addition of this flow will not violate the allocated bandwidth for CL class. If admitted, the flow descriptor for this flow will be added to the classifier's flow descriptor table. Since a flow's lifetime is generally not known *a priori*, MACE relies on a timeout mechanism to remove stale flows. If an admitted CL flow does not inject any packet into the network within a timeout period, we declare the flow "stale" and remove it from the classifier flow descriptor table.

Flow Setup. The flow setup unit assists the flow recognition unit in building the flow table. It also provides the admission control unit with a bandwidth usage estimate of each new flow.

To aid the flow recognition unit, the flow setup unit uses a list of descriptors of applications that may receive CL service. Each descriptor contains a port number and user specified token bucket parameters. This list is stored as a plain text file which network administrators can easily modify. The flow setup units export the list of port numbers that the flow recognition unit should monitor. The flow recognition unit keeps the list in kernel to speed up the recognition process. Whenever a CL flow is recognized (by having a specific port number and number of packets above the threshold), the flow recognition unit calls the flow setup unit. The flow setup unit finds the bandwidth usage estimate of the new CL flow from the list of descriptors and invokes the admission control unit.

Admission Control. The admission control algorithm used in this paper is the one studied in [JSD97]:

$$v\mu > \widehat{r} + \widehat{\nu} \tag{1}$$

where μ is the bandwidth allocated to CL traffic, v the target utilization, $\widehat{r}$ a rate estimate of the new connection requesting CL service, and $\widehat{\nu}$ the measured aggregate bandwidth of existing CL connections. A detailed explanation of the algorithm and recommendations on parameter choices can be found in [JSD97].

Estimator. The estimator computes the average bandwidth usage. It periodically samples the counter maintained by the measurement unit. We use the same moving time window mechanism presented in [JSD97] to obtain a bandwidth usage estimate from bandwidth averages. This moving time window mechanism uses a fixed window size expressed in number of samples. The maximum bandwidth during each measurement window is recorded by the estimator. The bandwidth usage estimate is updated on three occasions. At the end of a measurement window, we update the estimate to be the recorded maximum sample. Whenever an individual sample exceeds the estimate, we immediately update the estimate with the new sampled value. We also update the estimate when a new flow is admitted by adding the flow's reserved rate to the current estimate.

3.1 Caveat Implementor

Next, we present two caveats while implementing MACE. The first one is deciding where in kernel to implement CL service support, and the second is in deciding how to implement the packet scheduler.

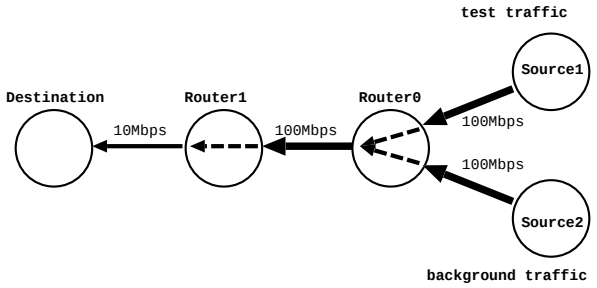


Figure 4: Experiment Setup

We want to implement CL service support in the critical path of packet forwarding to achieve efficiency and speed. There are two possible places to implement MACE, the packet input queue and the packet output queue. We first experimented with adding support for CL service at both the input queue and the output queue. We found out that the normal processing of input packets is done by Ethernet device interrupt handler, which must be fast to keep up with packet arrivals. Adding CL service support at the input queue slows down the interrupt handler because each packet must now be classified and put into the proper queue. The slow-down of interrupt handler causes more packets to be dropped at a busy router since the possibility of overrunning the Ethernet interface buffer is now higher. This is a well known problem of input-queueing [Kes97]. Furthermore, we observed little input queueing at the IP level, hence we do not add CL service support at the input queue. On the other hand, we observed more packet queueing at the output queue, and output queueing is done at an interrupt level lower than the device interrupt handler's so classifying and queueing each packet does not decrease the kernel's efficiency in receiving incoming packets. In our final implementation, MACE is done only at the packet output queue.

The second caveat comes from the implementation of the packet scheduler. In our implementation, outgoing Ethernet packets are always queued first in the MACE queue. Packets are then dispatched to the output interface by the scheduler according to their classification. In the first version, we dispatched BE packets to the interface as long as the CL queue was empty. This seems reasonable at first, but we then discovered that it was possible for BE packets to get queued on the device interface queue inside the kernel. On a busy router, the interface queue almost always contains more than one packet; often, the interface queue contains more BE packets than it can transmit between successive CL packet arrivals thus causing CL packets to be queued behind BE packets. As a result, CL packets on a busy router receive no better service. We revised this implementation so a BE packet is dispatched only when the interface queue is empty. A CL packet is dispatched to the interface if one is awaiting service; if no CL packet is awaiting service, then BE packets are dispatched one by one only when the device is ready. This ensures that in the worst case, a CL packet is delayed by at most one BE packet, in addition to CL packets ahead of itself. With this implementation, CL packets are no longer queued behind BE packets and indeed receive better service than BE packets.

4 Experiment Setup

Extant studies on the performance of measurement-based admission control and controlled-load service have mostly been done through theoretical analysis and simulation. In this paper, we look at the performance of MACE on a network testbed.

Our network testbed consists of four 200 Mhz PentiumPro PCs and one 167 Mhz UltraSparc Sun workstation. Each PentiumPro PC is equipped with three 100 Mbps Ethernet interface cards; the Sun workstation is equipped with five 10 Mbps interfaces. The testbed setup is depicted in Figure 4. All nodes in our testbed are connected with point-to-point links. The node labeled `Destination` in the figure is the Sun workstation. A 10 Mbps bottleneck link connects `Destination` and `Router1`. In all our experiments, the node labeled `Router0` never sees congestion; the node labeled `Router1` experiences congestion due to the slower speed of the bottleneck link and is the focus of our study. We send UDP traffic from the two source nodes to `Destination`.

There are two types of traffic in our experiments: CL and BE traffic (referred to as *testing traffic* from here on), whose behaviors we study in the paper, and background traffic, which creates the re-

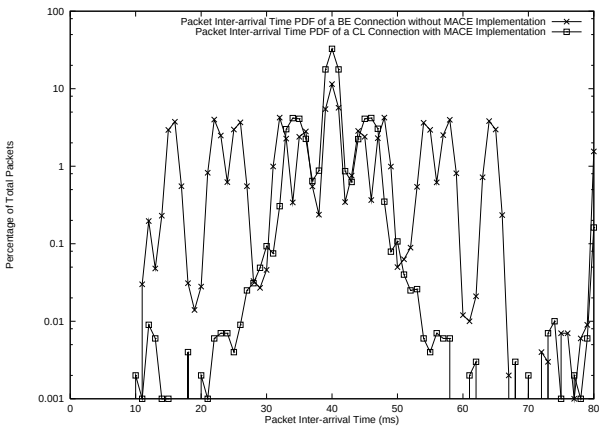


Figure 5: Unexpected Histogram of Packet Inter-arrival Time with Fixed Size Congestion Packet

quired network congestion for our experiments. We found that if CL and BE traffic sources were run side by side on the same workstation as the background traffic sources, background traffic sources could not receive enough CPU time to create the desired level of congestion. Thus, we send the two types of traffic sources from separate nodes in our testbed.

In the next two paragraphs, we describe the creation of different levels of network congestion. In section 4.1, we describe the source models used in generating flows that we want to study, and in section 4.2, the experiment scenarios used for data collection.

To create different levels of network congestion, we use a simple source model that continuously sends UDP packets. The source model has two tunable parameters: the average packet size and the packet generation interval. We change the average packet size to vary the average packet transmission time at a router; and we change the packet generation interval to vary the packet arrival rate at a route. We adjust both parameters to create different levels of network congestion. UDP connections are used instead of TCP connections for background traffic because a TCP connection reduces its sent rate when network is congested, i.e. when packet loss is detected.

While designing and testing our background traffic, we discovered two problems that could have invalidate our results if they were not addressed. We originally used fixed size packets, which gave rise to inter-arrival times of testing traffic as recorded in Figure 5. One would expect the histogram function to be a relatively smooth bell shaped curve; however, we see sharp "peaks" and "valleys" in the graph. Randomization of the congestion traffic packet size eliminates this problem as shown in Figure 9(Section 5.2). We also discovered another potential problem with the FreeBSD kernel. The FreeBSD 2.2.2 kernel implements ICMP source quench. Under congestion, routers with this implementation send messages to senders advising them to reduce their connection rates. As a result of receiving and processing many such packets, the senders are not able to generate packets very fast. This is inconsistent with our objective of creating network congestion, hence we disabled this feature of FreeBSD 2.2.2 in our experiments.

4.1 Source Models for Testing Traffic

In our experiments, we model a testing traffic source as either a constant bit rate (CBR) source or an ON/OFF process with constant peak rate. The CBR source sends fixed size packets at constant inter-arrivals. This is a model we use for pre-recorded music broadcasted over the MBone.

We use two types of ON/OFF sources: one with exponentially distributed ON/OFF times, the other with Pareto distributed ON/OFF times. During each ON period, packets are generated at a fixed rate. The EXP1 model shown in Table 1 is a model for packetized voice encoded using ADPCM at 32 kbps. Pareto distribution is a heavy-tailed distribution that can be described by two parameters: its location and shape (β). A Pareto shape parameter less than 1 gives data with infinite mean; shape parameter less than 2 results in data with infinite variance. The Pareto location parameter can be computed from the formula: $mean \times (shape - 1) \,/\, shape$. Each ON period of the Pareto-ON/OFF model is a Pareto distributed random number. During each ON period, packets are generated at a constant bit rate. The OFF times are also Pareto distributed. We use Pareto ON/OFF models with finite mean and infinite variance. Aggregation of Pareto ON/OFF sources models traffic exhibiting long range dependency (LRD) [WTSW95].

In our experiments, we only use one kind of

Source Model	$\overline{T}_{on}$ (ms)	$\overline{T}_{off}$ (ms)	Peak Rate	β
POO1	400	400	64 kbps	1.5
POO2	40	360	256 kbps	1.1
POO3	313	2925	64 kbps	1.1
EXP1	313	325	64 kbps	
EXP2	20	180	256 kbps	

Table 1: ON/OFF Source Models Used in Our Experiments

CBR source, a source sending at 64 kbps. We use several variations of the two ON/OFF source models as shown in Table 1. Each row in Table 1 corresponds to an ON/OFF source model. The columns in the table show various parameters for each model, with $\overline{T}_{on}$ being the average ON time, $\overline{T}_{off}$ the average OFF time, `Peak Rate` the connection data rate during ON times, and β the shape parameter of Pareto ON/OFF models.

We implement each source model with a program that sends UDP packets. Similar to the generation of background traffic, we can tune the testing traffic for packet generation intervals but fix the packet data size to be a constant 320 bytes. We achieve different connection rates by varying packet inter-arrival times, e.g. for connections with data rate of 64 kbps, we generate a packet every 40 ms, and for connections with data rate of 256 kbps, we generate a packet every 10 ms. In Table 1, the `Peak Rate` refers to the actual data rate, i.e. the amount of data sent in bits per second. The overall bandwidth consumption is higher than the data rate because each packet also carries protocol headers besides data. Thus, during ON times, a connection with 64 kbps peak rate sends 320 bytes of data every 40 ms, with 12 byte RTP header, 8 byte UDP header, 20 byte IP header, 26 byte Ethernet header and trailer. The total size of each packet is therefore $320 + 12 + 8 + 20 + 26 = 386$ bytes. The overall bandwidth consumption of a 64 kbps connection is therefore $386bytes/40ms = 77.2kbps$. Hence, a 64 kbps source sending 320 byte data appears to be sending 386 byte packets to the measurement unit and is considered to be generating 77.2 kbps by the admission control unit.

4.2 Experiment Scenarios

In this section, we describe the four scenarios used in our experiments. In the first and second scenarios, we start the congestion traffic and wait until it fills the pipe, then we start the testing traffic. In the third and fourth scenarios, which we use to investigate MBAC, we report only data collected after a warm-up period of 100 seconds. About 30% of the link bandwidth is allocated to CL traffic.

Before presenting the four scenarios, we list the values used for all MACE system parameters. For flow recognition unit, the packet counter has a threshold of 100 packets for 64kbps traffic and 10 packets for 256kbps traffic. The value of packet threshold changes the number of available flows that can be recognized. We want each CL flow to have statistically significant number of packets before we recognize it as a flow; however, we do no claim that our choices are optimal. We make this a tunable parameter so network administrators can tailor the flow recognition unit to their needs. The measurement unit samples bandwidth usage every 5 seconds, and the estimator computes the bandwidth averages every 100 seconds. The sampling period determines how sensitive the measurement unit is to the burstiness of network traffic. Small sampling period gives more accurate instantaneous bandwidth usage; large sampling period gives smoother averages. The estimator window size affects flow admittance dynamics. An admission control algorithm with small estimator window size could potentially admits a small number of flows continuously, and large window size causes flow arrivals to be bursty. We made the window size to be 20 sampling periods so the estimator has enough samples. Again these parameters are tunable.

In our first scenario, we use 40 64 kbps CBR sources to generate testing traffic. Of the 40 CBR sources, 20 of them request CL service, the rest BE service. We run 20 background traffic sources with 20 ms packet inter-arrival times and randomized data packet size. Each background traffic source has a lifetime of 10000 packets, and each testing connection has a lifetime of 5000 packets; therefore, both background flows and testing flows have the same lifetime of 200 seconds. We record the loss

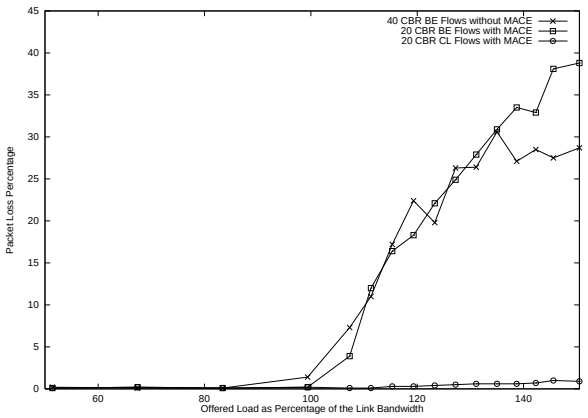


Figure 6: Packet Loss Comparison of CL and BE Connections with CBR Source Model

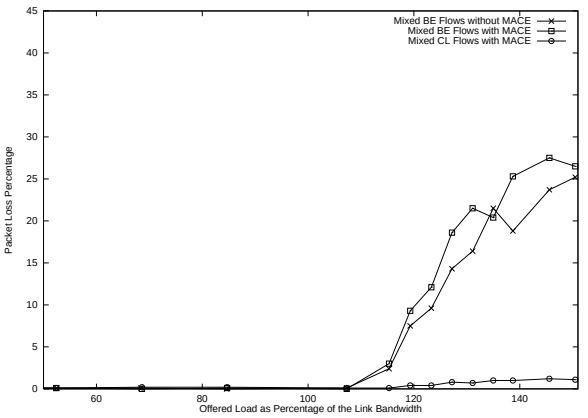


Figure 7: Packet Loss Comparison of CL and BE Connections with Mixed Source Models

rate and packet delay of testing traffic under different levels of network congestion with and without MACE implementation.

We pick the POO1 model in the second scenario to simulate more realistic audio sessions. We use 40 POO1 sources as the testing traffic, with 20 of them requesting CL service, and subject them to the same background traffic as under the first scenario. Again, under MACE, we admit all 20 CL connections. Since each source has a randomized lifetime, repeating an experiment with and without MACE yields two set of connections with different lifetimes. To ease comparison, we decide to use a record-and-replay scheme for this scenario. We first run an experiment for 200 seconds and record the lifetime and starting time of each of the 20 CL connection. We then playback this session with the same starting times and lifetimes with and without MACE.

To test the effectiveness of MBAC, we first use each of the source models in a separate experiment so only homogeneous sources under CL service are examined. In the third scenario, for each CL connection modeled by a source with data rate of 64 kbps, we reserve 77.2 kbps; for a CL connection modeled by a 256 kbps source, we only reserve 154 kbps, which, incidentally, corresponds to the bandwidth requirement of a connection with peak data rate of 128 kbps. We only reserve half the bandwidth requirement of 256 kbps sources to study the effect of statistical multiplexing gain on link utilization under MBAC. A properly working MBAC should bound the amount of CL bandwidth usage under the amount allotted to the CL service class. An MBAC that fails to bound CL traffic consumption to the alloted amount could potentially starve BE traffic. While we provide better service to CL traffic, we must still provide acceptable service to BE traffic.

In each experiment, we run a daemon process that forks out a fixed number of sources requesting CL service and maintains that number of CL connections by creating a new one whenever an old one departs. The number of connections that request CL service is so large that there are always connections waiting to be admitted into the CL class. All of these sources have exponentially distributed lifetimes with an average of 1000 seconds. Each experiment lasts three times the average flow lifetime. We record the bandwidth utilization in each experiment under both MBAC and Simple Sum Admission Control (SSAC).

In the fourth scenario, we mixed the source models of testing traffic. The source models in this scenario are: CBR sources with 64 kbps data rate, POO2, POO3, EXP1, and EXP2 sources. As in the third scenario, we reserve 77.2 kbps for each CL connection modeled by the CBR, POO2, and EXP1 source models, and 154 kbps for each CL connection modeled by the POO3 and EXP2 source models. We use background traffic with offered load of 130% of the 10 Mbps bottleneck link. Connections under this scenario all have average lifetimes of 1000 seconds, and each experiment is run for

3000 seconds.

5 Results

We present the results of our experiments on packet loss rate, packet delay, and MBAC in this section. For packet loss rate and packet delay experiments, we allocate enough bandwidth for CL traffic so MBAC is not used. Later experiments concentrate on MBAC studies. We show that with MACE, traffic under CL service experiences lower packet loss and more uniform packet delay than traffic under the traditional BE service; furthermore, the presence of CL traffic does not cause background traffic to lose excessive number of packets. We also show that under MBAC, the utilization of allocated CL bandwidth is higher than that under SSAC, with comparable delay and loss performance.

5.1 Packet Loss

Most interactive real-time applications designed for the Internet can adapt to varying network load and occasional packet losses. Under heavy network load, when packet loss rate can be higher than 10%, even if the application can reconstruct the signal, the resulting quality is likely to be unacceptable. By putting these interactive traffic into a CL class, we hope to keep the loss rate reasonably low. In this section we present results from experiments conducted under the first and fourth scenarios described in the previous section. We show that loss rates of connections under CL service in both scenarios are much lower than that of traditional BE connections.

We first compare the aggregate loss rate of 64kbps CBR connections under BE service against identical connections under CL service. Data was obtained from an experiment under the first scenario described in the last section. Shown in Figure 6 is the plot of aggregate loss rates of 20 BE connections with MACE, 20 CL connections with MACE, and 40 BE connections without MACE implementation. The x-axis is the average offered load of background traffic expressed as a percentage of the 10 Mbps bottleneck link speed. The y-axis is the packet loss percentage of the respective traffic types.

The loss rate of CL traffic with MACE is significantly lower than that of BE traffic with or without MACE. While BE traffic without MACE experiences loss rate as high as 30%, the worst loss rate of CL traffic is just over 1%. Packet loss of CL traffic occurs primarily during the flow recognition phase when the initial 100 packets from each CL connection are classified as BE packets. It is clear that the presence of a priority scheduler reduces CL packet loss. This improvement can be beneficial especially under highly congested network conditions. We also observe that the loss rate of BE traffic under MACE is not significantly worse than the loss rate of BE traffic without MACE.

We expect real-time audio and video traffic on the Internet to be better modeled by traffic with ON/OFF times instead of CBR traffic [Flo96, GW94, BSTW95]. We conducted an experiment under the fourth scenario with mixed BE and CL traffics modeled by CBR 64 bkps, POO1, POO3, and EXP1 source models. Figure 7 graphs the aggregate packet loss rate of BE and CL connections. The figure confirms that traffic under CL service indeed experiences lower loss rate than under BE service. Figure 8 shows that the packet loss rate of background traffic in this experiment, given the ON/OFF CL sources, is lower than that under the first scenario, where the CL sources generate CBR traffic, by around 5% to 10%.

5.2 Packet Delay

Low packet loss rates alone may not be sufficient to guarantee good audio and video quality on the Internet. Another important factor that influences the quality of real-time traffic is the delay experienced by individual packets. Real-time audio and video applications typically compute playback points based on individual packet delays [MKT95]. Large packet delay variations cause playback points to be postponed, resulting in loss of fidelity. Hence, one of the objectives in introducing CL service is to reduce the delay variations experienced by real-time audio and video traffic.

A straight forward way to compute packet delay is for each packet to carry a timestamp. The sender stamps each packet before transmission. The difference between the value stored in the times-

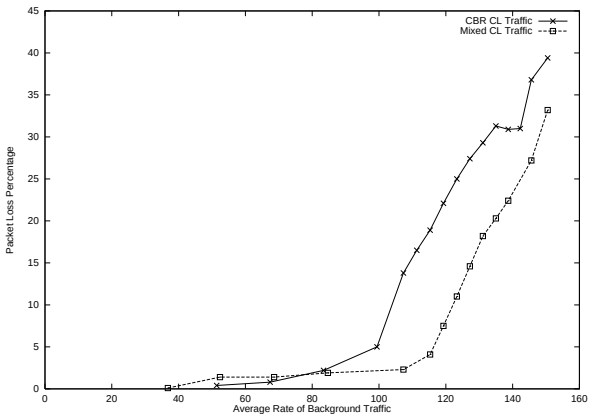


Figure 8: Packet Loss Comparison of Background Traffics with Two Types of CL Traffics

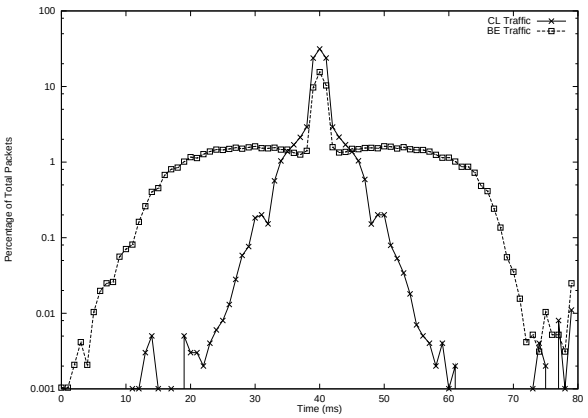


Figure 9: Histograms of Aggregate Packet Inter-arrival Time of CBR BE and CL Traffics

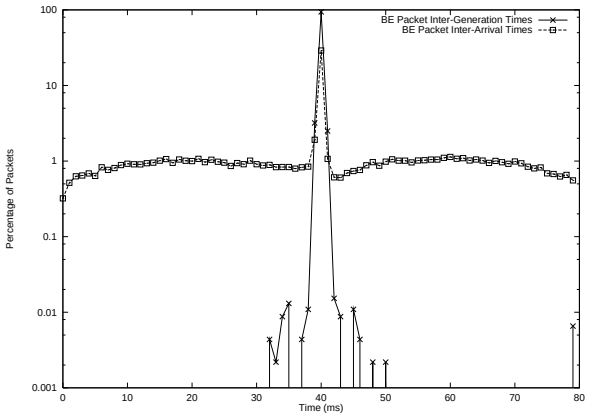


Figure 10: Histograms of Aggregate Packet Generation Times and Inter-arrival Times of BE Connections.

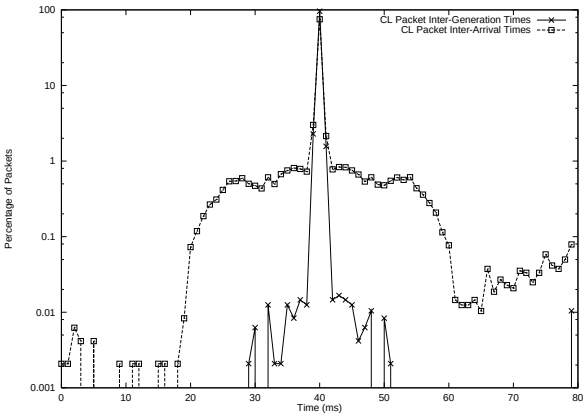


Figure 11: Histograms of Aggregate Packet Generation Times and Inter-arrival Times of CL Connections.

tamp and the time when the packet is received is the end-to-end delay. This calculation requires synchronized clocks at both ends of the transmission. Since we are only interested in delay variations, we only need to look at packet inter-arrival times. Since packets are generated with fixed inter-arrival at the sender, large variations of packet inter-arrival times seen by the receiver imply high delay variations.

In this section, we present results of experiments from the first, second, and fourth scenarios. In the first scenario, we study 64 kbps CBR traffic under CL service; in the second scenario, we study LRD traffic (POO1), and in the fourth scenario, we look at the highly bursty traffic, POO2 and EXP2, under CL service. We show that under CL service, 64 kbps connections all have significantly lower delay variations than under BE service, with 256 kbps connections seeing moderately improved delay variations.

Figure 9 shows the histogram of aggregate packet inter-arrival times for an experiment under the first scenario. The traffic we study is 64 kbps CBR connections under both CL and BE services. On the x-axis, we show the range of inter-arrival times, and on the y-axis, we show the percentage of packets with a particular inter-arrival time (note that the y-axis is in log scale). We removed all packets whose inter-arrival times are greater than 80 ms. Since packets are generated every 40 ms, and queueing delay of 40 ms is not possible in our experiment, the only way a packet can have inter-arrival times greater than or equal to 80 ms is when

one or more packets that immediately preceded this one was dropped, in which case, the 80 ms is not a true inter-arrival, and we discount them in our figure. This not only removes the effect of lost packets, but also removes the first packet from each ON time, which has an inter-arrival time that is actually the preceding OFF time. Under CL service, 99% of all packets have inter-arrival times that differ by less than 25% from the ideal inter-arrival time, while only 58% of BE packets are within this range.

Next, we present a back-of-the-envelop analysis of the delay BE traffic sees under congestion. In this experiment, there are 20 background traffic sources, each sending with a peak rate twice that of a testing traffic source. Without MACE, both CL and BE connections are treated as BE traffic. Each background traffic packet has 700 byte data (excluding all protocol headers) plus an exponentially distributed offset with a mean of 100 bytes for an average of 800 bytes per packet. Each testing traffic packet has 332 byte data (without protocol headers). On average, a router sees 2 incoming congestion packets for every 2 BE packets; hence, the average packet data size seen by a router is $800 \times .5 + 332 \times .5 = 566\, bytes$. Together with protocol headers, the average raw packet size is around 620 bytes. A BE packet arriving at a full queue (of 50 packets in this experiment) at a router with 10 Mbps outgoing link experiences a queueing delay of $50 \times 620 \times 8/10,000 = 24.8\ ms$ – a queueing delay that is certainly not negligible considering the packet inter-arrival time of 40 ms. Since the background traffic is sent twice as fast as the testing traffic, the interface queue is filled mostly with packets from these faster connections. An empty entry at the end of the queue has a higher probability of being occupied by a packet from faster connections. Thus packets from slower connections are likely to be queued after many packets from faster connections and have long queueing delays.

Researchers have observed LRD characteristic in Internet audio and video traffic [Flo96, GW94, BSTW95]. To see the effect of LRD, we use the POO1 model from the second scenario. We run an experiment from the second scenario and record flow arrival times and lifetimes of 20 CL connections. We then play the session back with the same flow arrival times and lifetime with and without MACE implementation. In Figures 10 and 11, we show the histogram of aggregate packet generation times at the sender and packet inter-arrival times at the receiver for both CL and BE traffic. Again we remove all packets with inter-arrival time of 80 ms or greater. Independent of objective perceptible difference in quality, we say that based on the Figures 10 and 11, an adaptive playback algorithm will be able to compute a smoother playback function with LRD connections under CL service.

In the fourth scenario, we use more complex traffic sources that include, aside from the simple CBR 64 kbps and EXP1 sources, LRD sources (POO3), highly bursty sources (EXP2), and highly bursty LRD sources (POO2). For this set of data, we show separate plots for connections with 256 kbps peak rate (POO2, EXP2) in Figure 12 and connections with 64 kbps peak rate (CBR, POO3, EXP1) in Figure 13. For 256 kbps connections, CL service provides marginal improvement over BE service in terms of packet delay. The marginal improvement here is the result of connections having fast transmission rate; connections with faster transmission rates fill up the queue with consecutive packets, packets from connections with slower transmission rates are likely to see a queue of packets from connections with faster transmission rates. While the improvement in packet delays of CL traffic over BE traffic is marginal, we note that the loss rates of both types of CL traffic is just under 1% while the loss rate for 256 kbps BE traffic is 20% and 24% for 64 kbps BE traffic.

5.3 Measurement-based Admission Control

In the last two subsections, we studied packet loss rate and packet delay without enabling MBAC on CL traffic; In this section we present our findings on the bandwidth utilization of CL traffic under both MBAC and Simple Sum Admission Control (SSAC). SSAC uses an admission control that limits the sum of reserved peak rates to be below the allocated bandwidth.

We present results of experiments from the third and fourth scenarios. We show that MBAC enables much higher levels of link utilization while maintaining packet loss rate and delay variations

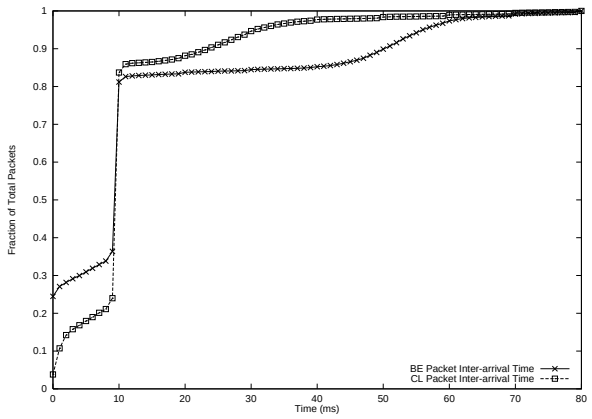


Figure 12: CDFs of Aggregate Packet Inter-arrival Times of Eight 256 kbps connections under CL and BE services

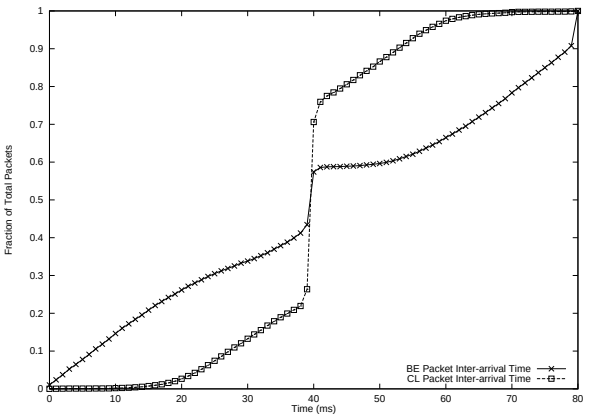


Figure 13: CDFs of Aggregate Packet Inter-arrival Times of Twelve 64 kbps connections under CL and BE services

Source Model	CL with SSAC		CL with MBAC	
	%Utilization	Flows	%Utilization	Flows
POO1	44.5	50	72.5	83
POO2	31	15	61	27
POO3	16	29	46	91
EXP1	49	50	89.6	93
EXP2	29	14	93	47
MIXED	36	19	74	34

Table 2: Comparison of SSAC with MBAC for Single Source Model

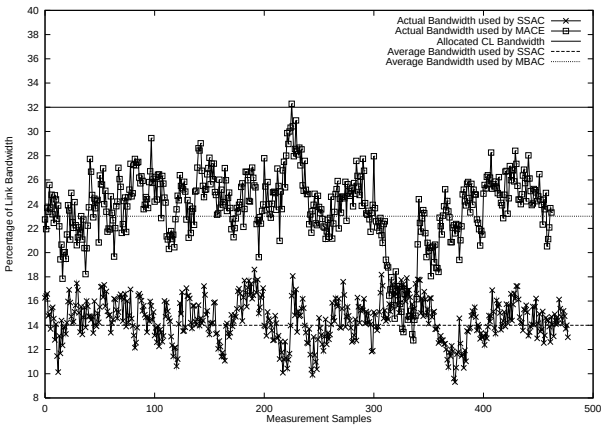


Figure 14: Aggregate Bandwidth Usage in 2500 seconds of POO2 Connections under both MBAC and SSAC.

comparable to that achieved under SSAC.

Table 2 shows that with the same amount of bandwidth allocated to CL traffic, MBAC outperforms SSAC in link utilization for all the source models. The `%Utilization` columns show the average bandwidth utilization under each admission control algorithm; the `Flows` columns show the number of concurrently admitted CL connections under each algorithm. For POO1 and EXP1, we reserved 39% of total link speed, and for the rest of the source models, we reserve 23% of the total link speed. Figure 14 shows a sample path bandwidth utilizations under MBAC and SSAC for POO1 CL connections. As expected, MBAC is very effective when source models are smooth (POO1 and EXP1). When sources are very bursty, such as in the POO3 case, where the peak to average ratio is 10:1, MBAC's effectiveness in increasing link utilization is limited. We also note that physical limitation of the machines prevented us from generating more than 100 offered POO3 flows.

Next, we show that the higher link utilization under MBAC does not drastically degrade the service CL traffic receives. We used mixed traffic with 5 source models: CBR with 64 kbps peak rate, POO2, POO3, EXP1, and EXP2. In the mixed traffic, the number of connections under each source model is roughly the same. We ran the experiment under both SSAC and MBAC with background traffic that is 130% of the 10 Mbps bottleneck link speed. For 256 kbps CL connections, only 8% more of total packets are received under SSAC with inter-arrival times of 10 ms or less as shown in Figure 15. When we compare the rest of CL traffic, the inter-

arrival time distributions are almost identical under both SSAC and MBAC. Figure 16 shows 64 kbps CL connections which ran side by side with the 256 kbps connections mentioned earlier. The performance is slightly worsened while the average link utilization under MBAC is 50% higher than under SSAC. Overall, MBAC delivers packet delays comparable to that under SSAC.

6 Conclusion

In this paper, we have presented our design and implementation of Measurement-based Admission Control for Controlled-load service. Results obtained from running experiments on actual network setup show that this implementation successfully accomplishes our design goals. While our experiments are of limited scale, the results give evidences that CL service can provide benefits to real-time audio and video connections on the Internet. We learned some important lessons from this implementation about good experiment setup (Section 3), programming techniques (Section 4), and data analysis.

In summary, we find that traffic under CL service has a much lower loss rate than that under BE service. While providing CL service does increase the packet loss rate of background traffic, this increase is marginal. Under CL service, traffic sees tighter packet inter-arrival time, and this holds true for homogeneous as well as mixed traffic sources. In MACE, CL service is complemented by MBAC so the membership in CL class is controlled, resulting in lower packet loss rate and tighter packet inter-arrival time for admitted traffic. It yields significantly higher bandwidth utilization at the cost of slightly worse packet inter-arrival times.

Acknowledgment

We thank the anonymous reviewers for their comments.

References

[AMK97] E. Amir, S. McCanne, and R. Katz. "Receiver-driven Bandwidth Adaptation for Light-weight Session". *ACM Multimedia*, Nov. 1997.

[BBS98] S. Bajaj, L. Breslau, and S.J. Shenker. "Is Service Priority Useful in Networks?". *Proc. of ACM SIGMETRICS*, 1998.

[BSTW95] J. Beran, R. Sherman, M.S. Taqqu, and W. Willinger. "Long-range Dependence in Variable-Bit-Rate Video Traffic". *IEEE Transactions on Communications*, 43:1566–1579, 1995.

[CW97] D. Clark and J.T. Wroclawski. *An Approach to Service Allocation in the Internet*. Internet Draft, Jul. 1997.

[Flo96] S. Floyd. "Comments on Measurement-based Admissions Control for Controlled-Load Service". Submitted to *Computer Communication Review*, 1996. **URL** ftp://ftp.ee.lbl.gov/papers/admit.ps.Z.

[GW94] M. Garrett and W. Willinger. "Analysis, Modeling and Generation of Self-Similar VBR Video Traffic". *Proc. of ACM SIGCOMM '94*, pages 269–279, Sept. 1994.

[JSD97] S. Jamin, S.J. Shenker, and P.B. Danzig. "Comparison of Measurement-based Admission Control Algorithms for Controlled-Load Service". *Proc. of IEEE INFOCOM '97*, Apr. 1997. **URL** http://netweb.usc.edu/jamin/admctl/info97.ps.gz.

[Kes97] S. Keshav. *"An Engineering Approach to Computer Networking"*. Addison Wesley, 1997.

[MKT95] S.B. Moon, J. Kurose, and D. Towsley. Packet audio playout delay adjustment algorithms: Performance bounds and algorithms. Technical report,

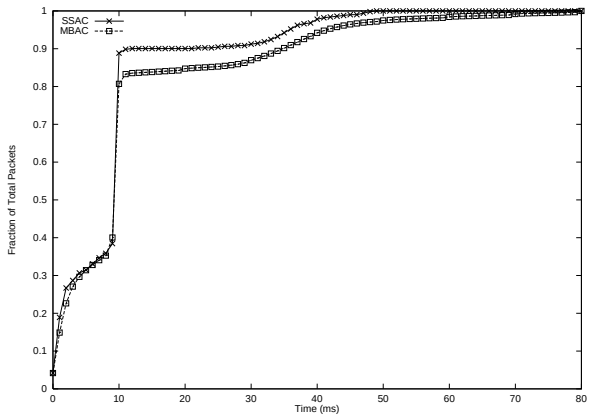


Figure 15: CDFs of Packet Inter-arrival Time under SSAC and MBAC for 256 kbps Connections.

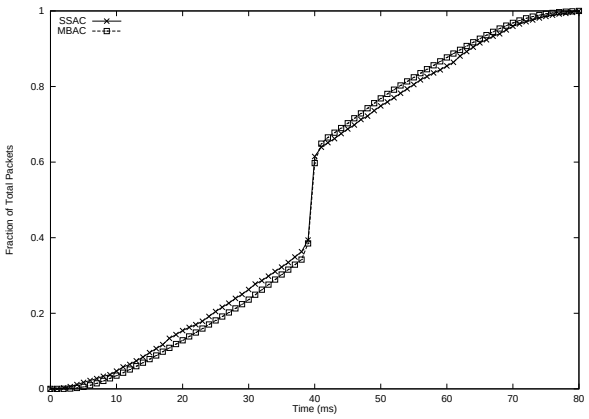


Figure 16: CDFs of Packet Inter-arrival Time under SSAC and MBAC for 64 kbps Connections.

Dept. of Computer Science, Univ. of Massachusetts, Amherst, Aug. 1995.

[NJZ97] K. Nichols, V. Jacobson, and L Zhang. *A Two-bit Differentiated Services Architecture for the Internet*. Internet Draft, Nov. 1997.

[RKTS94] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. "Adaptive playout mechanisms for packetized audio applications in wide-area networks.". *IEEE INFOCOMM*, pages 680–688, 1994.

[SPG97] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. RFC 2212, Internet Engineering Task Force, September 1997.

[Wro97] J. Wroclawski. Specification of the controlled-load network element service. RFC 2211, Internet Engineering Task Force, September 1997.

[WTSW95] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson. "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level". *Proc. of ACM SIGCOMM '95*, pages 100–113, Aug. 1995.

Charging of Multimedia Flows in an Integrated Services Network

Burkhard Stiller, George Fankhauser, Bernhard Plattner
Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, ETH Zürich
Gloriastrasse 35, CH–8092 Zürich, Switzerland
Phone: +41 1 632 7016, Fax: +41 1 632 1035
E-Mail: [stiller | gfa | plattner] @ tik.ee.ethz.ch

Abstract

Admission to communication services for multimedia applications does not imply purely technical control, it has to integrate economic admission policies as well. This is mainly due to the fact that multimedia communication services within the Internet cannot be assigned in a cooperative fashion as they represent bandwidth-hungry applications. Internet economics make various assumptions on pricing policies and services, encompassing features of the economic model, the pricing mechanism, the network model, and their practicability in the Internet today. Based on general architectural aspects for integrating and supporting charging and accounting properties of integrated services networks, the implementation of a flow-based approach to charge multimedia data in the Internet is presented. Ongoing work in the design, implementation, and evaluation of a control protocol for charging and accounting resource reservations in the integrated services Internet applies usage-based pricing models employing an auction-based pricing model (delta auction) and an adaptive, load-sensitive, volume pricing model. In contrast to packet-based charging schemes, the developed approach of flow-based charging features low processing overhead and low bandwidth requirements of less than 1% for fine-grained and frequent reservation setups are achieved.

1 Introduction to Flow-based Charging

Concerning networking traffic, todays Internet is able to support a best-effort type of traffic only. For various reasons, however, some applications, especially multimedia applications require networking support for at least a little better service assurances, may be even some guarantees. These service guarantees have to be provided by end-system resources, *e.g.*, supported by a QoS-aware operating system, and by the network itself, *e.g.*, supported by resource reservation protocols. Therefore, a first approach for charging deals with the flow-based approach, since it deals with a significant amount of resources that has to be reserved exclusively for a certain application. As these networking resources cannot be shared any more between other applications, an adequate policy has to ensure that this resource consumption is payed for. This policy includes as the main component economic incentives besides traditional protocol support as mentioned. Since the amount of required resources per application – or in general per flow – may vary, a flat-fee approach does not provide the right economic incentive. Therefore, usage-based pricing schemes need to be applied which take into account the actual amount of currently consumed resources.

Once this need for suitable methods for a usage-based service principle has been accepted, prerequisites to implement accounting, charging, and pricing have to be studied in detail. Generally required components for these tasks and their interdependency are depicted in Figure 1 which does not distinguish between end-systems and routers.

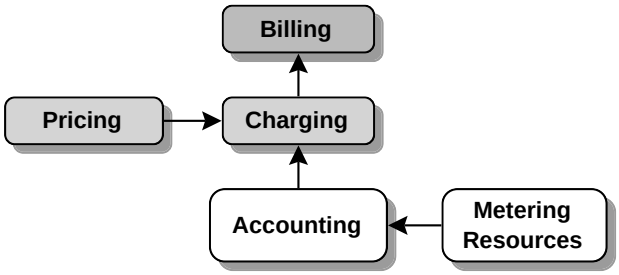


Figure 1: Functional Components of an Architecture for Flow-based Charging.

At a first step, the particular utilization of resources is metered or monitored and has to be accounted for. Accounting data can be available for end-system resources, *e.g.*, in terms of utilized central processing unit time, memory, or special devices. However, accounting data for network resources and network access are more important for the networking perspective of evaluating communication costs. These resource data may encompass, *e.g.*, the acquired network access bandwidth, duration of a communication, expected delays, loss rates, or time-of-day information. To support multimedia flows, accounting data must include Quality-of-Service (QoS) specifications for resources that are utilized to provide a service.

In a second step, this accounting information has to be collected for a particular user or application. Based on different networking infrastructures, communication scenarios, or network provider concepts a single or multiple collecting centers exist within an administrative networking domain. For that reason, a general accounting protocol has to be integrated on top of or within the networking

architecture, such as the Internet protocol suite and service architecture, to exchange well-defined accounting records. Accounting information even may be piggy-backed on protocols already in use, like a resource reservation protocol.

In the third step, a charging scheme calculates the currently valid price for these utilized resources which are precisely determined by the collected accounting records. A pricing scheme specifies in a generic and flexible manner prices for certain units of goods, *e.g.*, per one megabyte of data at noon over a distance of ten hops. This pricing information and the real use of resources determines the amount of money to be payed for the usage of these resources. The local pricing settles base prices per unit which are dependent on many factors, such as time-of-day, thresholds, or discounts. In turn, these calculated charges are determined in charging records and may be transferred and charged directly to the user, directly to the application, indirectly via host institutions, or indirectly via the utilized workstation. It has to be mentioned that charging may take place in the same system where the accounting took place. In this case the system itself is the accounting and charging center which may be designed for every involved system in the network, resulting in a completely distributed and decentralized approach.

In the final step, these charging records are handled by a billing system. The billing system decides on a provider-based or on a user-requested policy, when and how the final communications bill is delivered to a user or the application. Normally, billing is done in a more centralized manner, based on transmitted charging records. In addition, interoperator agreements may be determined to allow for a collective billing between single operators.

Obviously, these three steps involve a well-structured interaction between these components in a distributed manner, where the information on their location is of secondary interest, but the accounting, charging, and pricing information and their handling in terms of new or enhanced control protocols is of primary interest. In addition, of overall interest in this case is the secure exchange of accounting and charging records by these control protocols and a secure handling of corresponding payments.

2 Designing a Charging Protocol for Flows

Retaining the old flat-fee model for charging and accounting of multimedia flows does not reflect the new service classes being offered and it would lead to a fatal admission policy problem. As it can be assumed that the problem of reserving resources will be technically solved in the near future, a consistent and stable policy to grant admission to these resources is still missing. At least for expensive high-end communication services, such as multimedia data flows, this policy should be implemented using financial incentives. Focusing on reservations has the advantage of treating large bandwidth quantities that last over a longer period of time than packet-based charging schemes can. Such reservations for flows build the basis for an efficient charging mechanism with low overhead which is required to make usage-based pricing mechanisms economically feasible. Concerning QoS requirements and data rates generated by multimedia applications, it is very likely that service classes needed are based on reservations. Since they will occupy a significant part of the network bandwidth, they should be the premier target for usage-based pricing.

Using multimedia flows based on resource reservations as basic units for charging and accounting, the reservation protocol transports signaling information. A simplified version of RSVP [4] including basic functionality only, is used to carry charging relevant information. As shown in Figure 2, the basic receiver-oriented control messages are extended. PATH messages convey additional payment information and quote requests. This information is echoed by the receiver and returned to the sender in RESV messages containing receiver payment information and quotes. This round trip messaging can be used to have either sender or receiver pay for the reservation. To allow for locally different pricing models, the willingness to pay is reduced at each hop and price quotes are added up by the local market price. Payments are requested for bandwidth used on each hop's outgoing link over a short period of time.

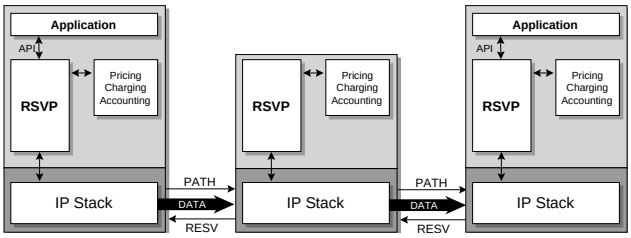


Figure 2: Applying RSVP to Charging Tasks

The developed approach supports local pricing decisions to enable easier interconnection of multiple providers which is based on peer agreements at aggregated links. It minimizes charging protocol overheads and allows for splitting payments between senders and receivers.

2.1 Pricing Models

Two pricing models have been used to evaluate the charging protocol. The major difference between these models is the way how they calculate prices. The delta auction approach calculates market prices based on consumers' bids and has been shown to be incentive-compatible [3]. Consumers announce their willingness to pay by bidding for the bandwidth they currently request. All requests are split into small sized units in order to perform an auction on bandwidth only. In situations, where no congestion occurs and all the bids fit into the link's offered bandwidth, no second price is available and the service can be given for free or optionally for a low volume-fee in case of non-zero marginal costs.

The other approach, a dynamic volume price model, is provider oriented with base-prices that are modified according to demand. Although it calculates prices dynamically according to the load on the network, the base-prices are set by the operator of each node [2]. The model features different service classes, best-effort and reserved bandwidth have been considered. The basic unit sold is a bandwidth reservation over a fixed period of time. This model provides two options which control the price when congestion is about to occur. One is a function that describes the increase in price when demand exceeds supply provided by the link bandwidth. As a parameter to this function, the actual demand/supply ratio is used. The other option is a parameter which specifies at what load the before mentioned function is applied to increase the price.

3 Implementing the Charging Protocol

Simplified RSVP messages used for reservation setup, such as PATH and RESV messages, are enhanced by adding payment information and price queries. Depending on the economic models used at each router the necessary information is extracted from a uniform message and processed. For example, the willingness-to-pay field of a RESV message is used to transmit bids in case of an auction. For more provider-oriented models with predefined prices, the reservation is only granted when the required amount is available from this field. PATH messages are used to pin a path and setup a state to make sure that RESV messages follow the same route back to the sender. PATH messages may contain a request to quote market prices or a sender provided payment. All the messages share the same protocol data unit (PDU) structure with type fields and flags indicating the operation. Figure 3 shows the basic PDU of the reservation protocol. The highlighted parts have been added for the charging and accounting protocol.

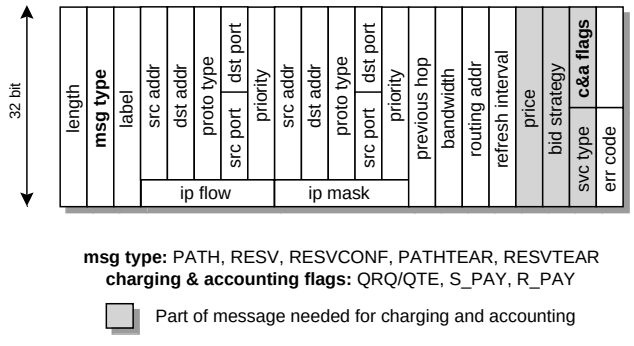


Figure 3: Enhanced Reservation Message Format for Charging Resources.

Normal operation includes quote requests and reservation requests that are initiated by applications via an extended reservation protocol API. Applications are expected to react in an adaptive fashion to the prices quoted and reservations performed. Based on minimal user input, such as profiles and budget constraints, middleware modules are responsible for negotiation of details with the extended API. Failures, such as unavailable resources or insufficient payments, are signaled via the standard PATHTEAR and RESVTEAR messages of the reservation protocol.

4 Results

Concerning first evaluations of our prototypical implementation, the general overhead of the charging and accounting protocol as described above was measured in relation to the requested resources. Primarily, bandwidth is used as a measure, but also the requited memory for accounting and extended flow and reservation tables has been studied. Furthermore, the delay introduced by the charging protocol and the scalability for large numbers of flows have been investigated. Interworking of different pricing models and payment modes still needs to be assessed. Measurements were performed on a flexible testbed of Crossbow routers and hosts [1]. Machines used included PCs with 233 MHz AMD K6 processors, running NetBSD 1.21, having 512 kByte of second level cache and sufficient amounts of memory to hold large flow tables.

As an overview and comparison of memory needed to store flow states extended by charging and pricing information at each router, some figures specify the requirements of total 194 Byte in the RSVP daemon in user space: The flow state requires 120 Byte, pricing 14 Byte, and the account 12 Byte. Basic authentication data (28 Byte) is also included in the form of an integrity object with an MD5 checksum, sequence number, previous hop, and time stamp. The charging relevant data uses 28% of the space for the total flow state information.

The protocol overhead needed to transport charging relevant information is most significant (cf. Table 1) since it is using up resources that could be otherwise sold. As an example we are using an IP telephony stream with very fine grained reservation periods of 5 seconds and a constant bit rate of 64 kbit/s (40,000 Byte for the 5 second period). Charging information for this stream uses only 0.75% of the flow's total bandwidth. During one reservation period 500 data packets are processed and a message round trip for querying the price and refreshing the reservation is needed.

Table 1: Protocol Overhead for IP Telephony Stream.

	[Byte/ packet]	Packets []	Sum [Byte]	[%]
Payload	80	500	40000	73.2
IP/UDP Headers	28	500	14000	25.6
Simplified RSVP Msg	96	2	192	0.4
Charging Information	12	2	24	0.1
Price Queries	108	2	216	0.4
Authentication Data	28	2	56	0.1
Optional RESVCONF	136	1	136	0.2
Total	-	-	54624	100.0

Processing at routers to perform price queries, the delta auction, and authentication requires for a long living flow on the average 322 µs per packet on a router which in total encounters for 2.3% for the sample scenario of IP Tele-

phony flows. In order to put a realistic load on the system, flow tables had to maintain 100,000 flows concurrently in a single router while performing this measurement and the Crossbow traffic control functions to setup or remove flows were used.

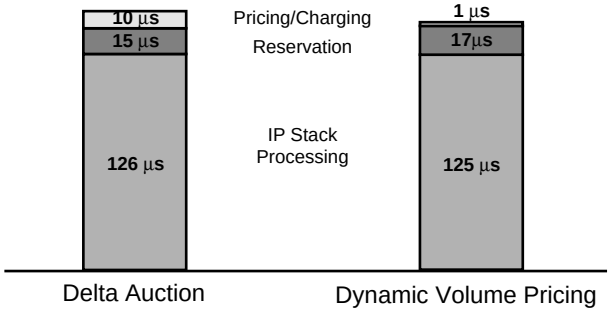


Figure 4: Processing Times for Pricing Schemes

Figure 4 shows the processing times for a single reservation request including processing of the charging information for different pricing schemes. In contrast to the above stated average numbers, it does only include processing figures for the reservation and charging protocol and not for traffic control. These numbers have measured interface-to-interface delays on a router connected to two 10 Mbit/s Ethernet subnetworks. Total delays for a reservation message encountered 151 µs and 143 µs for the delta auction and the dynamic volume pricing model, respectively. While the latter does not add significantly to the processing delay (<0.1%), the more complex delta auction uses up less <7% of the overall delay. However, it must be noted that the IP protocol stack processing delay including the protocol overhead, the required socket processing, and the use of timers, remains similar at 126 µs and 125 µs for the delta auction and the dynamic volume pricing scheme, respectively. In addition, the reservation fraction includes RSVP processing and the management of time-outs for reservation periods.

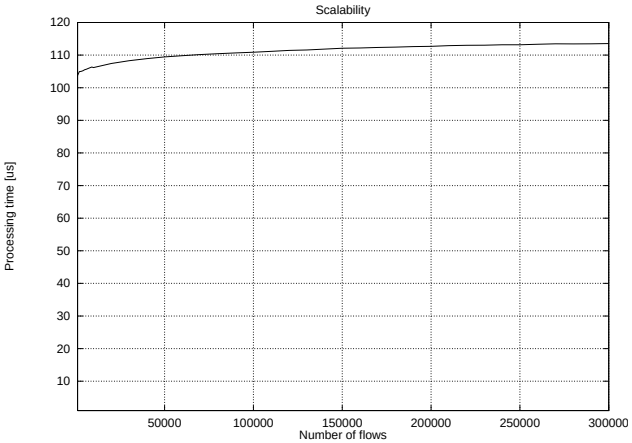


Figure 5: Reservation Request Processing Times for the Delta Auction Pricing Scheme.

While simple pricing models, such as volume pricing (even if the price is dynamic), can be implemented with simple lookup functions and do not involve complex sorting or searching procedures, the presented delta auction has to maintain proper data structures to process incoming bids and to decide whether they can be admitted or have to be rejected. Searching can affect the scalability of such a solution. For the prototype implementation, balanced trees were used to keep processing complexity at the *O(log n)* where *n* is the number of flows. Using load generators for *n* up to 300,000 flows shows the desirable behavior for reservation processing with the delta auction (cf. Figure 5). Balancing trees with information about the ongoing auctions is a strict requirement, since bidding strategies of participating end-users (or applications and daemons that act on behalf of them) will lead to degenerated trees.

5 Conclusions and Future Work

Adding charging and accounting functionality to the integrated services network architecture is feasible. By applying flows instead of packets, it is suitable to provide charging and accounting functionality without prohibitive overhead. For a sample scenario of 100,000 IP Telephony flows at one router the processing overhead for the auction-based, dynamic pricing and the associated charging functions was only at 2.3%. The most important result determines that the introduced protocol overhead was determined by 0.75% only for a reservation interval of 5 s, handling sender as well as receiver payments and supporting local pricing methods concurrently.

Future work includes new routing schemes considering QoS and pricing information to enhance the currently static routing approach. In addition, an interesting economic problem is how to integrate best-effort flows into the architecture. The presented current approach focuses deliberately on reservations only, under the assumptions that the revenue from reserved flows will exceed that from best-effort traffic. Future empirical studies will also be needed to address the reliability of this assumption.

References

[1] D. Decasper, M. Waldvogel, Z. Dittia, H. Adiseshu, G. Parulkar, B. Plattner: *Crossbow – A Toolkit for Integrated Services over Cell-Switched IPv6;* IEEE ATM'97 Workshop, Lisboa, Portugal, June 1997.

[2] G. Fankhauser, B. Stiller, B. Plattner: *Arrow - A Flexible Architecture for an Accounting and Charging Infrastructure in the Next Generation Internet*; Accepted for publication in Netnomics – Economic Research and Electronic Networking, Baltzer, The Netherlands, 1998.

[3] W. Vickrey: *Counterspeculation, Auctions and Competitive Sealed Tenders*; Journal of Finance, 1961.

[4] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala: *RSVP: A New Resource Reservation Protocol;* IEEE Networks Magazine, Vol. 31, No. 9, pp 8-18, September 1993.

A Better-Than-Best-Effort Service for Continuous Media UDP Flows*

Mark Parris Kevin Jeffay F. Donelson Smith Jan Borgersen

University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC 27599-3175 USA
{parris,jeffay,smithfd,borg}@cs.unc.edu

http://www.cs.unc.edu/Research/dirt

Abstract

The Internet community is very interested in addressing congestion with active queue management mechanisms like Random Early Detection (RED). Recent proposals extend these mechanisms to actively penalize "misbehaving" flows. These methods favor TCP and TCP-like flows strongly over other flows like UDP. In this paper we propose extensions to active queue management disciplines which continue to address congestion and reward TCP-friendly flows while minimizing impact on continuous media flows using other protocols. Our mechanism, Drop Preference Management (DPM), recognizes tagged flows and manages their latency while constraining the bandwidth they consume. We present empirical results of experiments comparing our mechanism to plain RED.

1. Overview

As the Internet continues to evolve, increasing attention is being given to recognizing and addressing congestion within the network. Particularly, there is increasing focus on recognizing "well-behaved" flows, those that respond to congestion by reducing the load they place on the network. Both Braden, *et al*. and Floyd, *et al*. recognize TCP flows with correct congestion avoidance implementations as well behaved [2, 6, 7, 1]. They argue that these flows, as "good network citizens," should be protected and isolated from the effects of "misbehaving" flows. Misbehaving flows include incorrect implementations of TCP, unresponsive but lightweight UDP connections, and high bandwidth flows that aggressively retransmit when drops occur. A recent internet draft considers the problem of congestion in the current internet and makes two recommendations [2]. First, they recommend deploying active queue management schemes, specifically Random Early Detection (RED) to more effectively notify responsive flows of congestion [5]. Active queue management refers to extending the queueing discipline in the router beyond simple enqueue and dequeue with drop-tail when full. For example, RED does not wait until the queue is full to drop packets. Instead, it probabilistically drops incoming packets when the queue's average size exceeds a particular threshold and automatically drops a random packet when the average exceeds a higher threshold. This provides earlier feedback, before the queue overflows, and causes higher bandwidth flows to see a greater number of drops. Second, they recommend continued development of mechanisms to deal with flows that do not respond to congestion in TCP-friendly manner. To date "dealing with" these other flows has centered on how to constrain or penalize those flows [6, 9]. We recognize the need to protect well-behaved flows but also recognize that many applications choose unresponsive protocols, such as UDP, because they are concerned with throughput and (especially) latency rather than reliable delivery. Since reliable delivery in TCP depends on timeouts, feedback, and retransmissions, it can be incompatible with performance goals. Multimedia applications are a prime example of applications that avoid TCP for performance reasons.

Simply penalizing these non-TCP flows leaves application developers with some unattractive options. With the deployment of RED in many routers, application developers must realize UDP flows will be subject to more aggressive drop policies than in the past. The developer could use TCP and incur overhead for features she may not want. Or, she could use another protocol and be subject to aggressive drop policies. Another alternative would be to use a protocol that implements TCP-like congestion management without the other undesired features such as reliable

* Supported by grants from the National Science Foundation (grants IRIS-9508514 & CCR-9510156), the Advanced Research Projects Agency (grant number 96-06580), the IBM Corporation, and a graduate fellowship from the Intel Corporation.

delivery [3]. The correct long-term solution is to develop adaptive applications that respond to congestion notification (as well as application level feedback). Our approach recognizes all of these solutions. Our solution is orthogonal, providing better support for continuous media flows in the router.

To mitigate the impact of active queue management on UDP flows, we are working on queue management policies for routers that attempt to balance the concerns of congestion avoidance and the requirements for continuous media applications using UDP. Specifically, we are experimenting with extensions to the Random Early Detection (RED) packet discard mechanism for providing better performance for UDP flows without sacrificing performance for TCP flows. The following briefly outlines the design of our RED extension, called RED with Drop Preference Management (RED-DPM), and presents some early empirical results that suggest the mechanism performs effectively.

2. Active Queue Management

RED was designed as a mechanism to notify responsive flows of congestion, either explicitly, by marking packets, or implicitly, by dropping packets of a flow. The likelihood of a flow being notified is directly related to its (average) bandwidth utilization. In this way RED avoids global synchronization as many flows back off at the same time, and avoids unfairly penalizing bursty traffic. Instead, those flows consuming the most bandwidth see the most drops. If responsive, the flow will reduce the load it generates and the number of drops on that flow will decrease, allowing it to reach an equilibrium state. If the flow is unresponsive, the flow will continue to see a high number of drops, corresponding to its utilization of a large part of the queue. However, RED assumes that traffic is responsive and thus is vulnerable to misbehaving unresponsive flows. A non-responsive flow could consume a large share of the bandwidth while other flows decrease their utilization in response to congestion notification. While RED does drop packets in ratio to the flow's arrival rate, it still drops an equal percentage of arriving packets from all flows. So even well behaved flows continue to see drops if one misbehaving flow keeps the queue full. As a result of these drops, well-behaved flows can back down to extremely low (or no) throughput. Both RED with Penalty Boxes and FRED address this deficiency in RED [6, 9]. Floyd and Fall propose mechanisms for identifying several classes of flows: TCP-friendly, unresponsive, and very-high bandwidth flows [6]. They suggest simply that those flows which are not TCP-friendly should be "regulated" (without clearly defining a mechanism for regulation) to prevent them from dominating network resources while TCP-friendly (*i.e.*, responsive) flows continue to be notified of congestion via a RED-like mechanism.

FRED is an extension to RED that uses per-active-connection accounting to ensure fairness and isolation between all active flows [9]. While RED drops an equal percentage of packets across all flows, FRED increases the likelihood of drops for high bandwidth flows. It does this by concentrating drop actions on those flows that are exceeding a threshold based on the average *per flow* logical queue length. Further, flows whose logical queue utilization is below average will not be subject to these drops. It also concentrates drop actions on those flows that have failed to respond to notification in the past. Both FRED and RED with penalty boxes take a harsh stance on unresponsive flows, constraining them tightly by dropping potentially all arriving packets once a queue in a router is above threshold.

3. RED with Drop Preference Management

We propose a mechanism to constrain misbehaving flows while attempting to provide them with the best performance possible under those constraints. A key observation is that latency may be as important a consideration for these flows as overall bandwidth. Thus we trade-off bandwidth for (lower) latency. We seek to minimize the latency of those packets that do arrive at the end system and avoid delivery of *stale* packets, packets that are queued in the network and are not likely to arrive in time to be useful.

In our scheme we identify multimedia flows with a tag[1] and maintain per flow statistics for those flows only. Non-tagged flows are subject to the standard RED queue management policy. Packets in tagged flows are subject to drops based on queue length thresholds similar to FRED but also are subject to drops if packets for that queue exceed a "staleness" threshold. The staleness threshold is based on the age and depth of the oldest packet for this flow currently enqueued on the router. Further, anytime a decision to drop a packet is made, an alternative drop mechanism, *delete and advance*, is used. This mechanism drops the oldest packet (at the head of the flow's logical queue) so newer, *fresher* packets are given priority. It also reduces the time the new packets spend in the queue by

[1] Any of several proposed tagging mechanisms in the IP header could be used, *e.g.*, the Clark or Nichols schemes [4, 10].

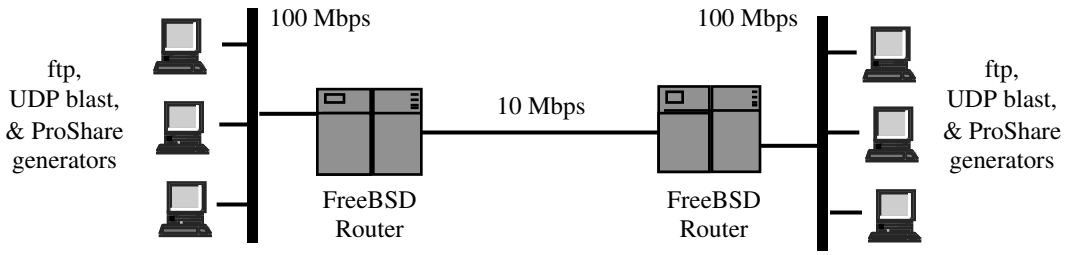


Figure 1: Experimental network configuration.

promoting them into their logical predecessor's location in the main queue. The depth of the queue elements for all other flows is unchanged.

With drop preference management we expect to reduce the impact of misbehaving tagged flows on responsive flows by performing more aggressive dropping of non-TCP-conformant flows. These flows are subjected to a RED test on their (potentially much shorter) logical queue as well as a staleness test. However, UDP flows, and in particular responsive UDP flows, benefit overall as their packets tend to stay closer to the head of the router queue and thus are delivered with lower latency. DPM should benefit both responsive and non-responsive continuous media flows. Non-responsive flows will see a large number of drops, just as they would in the face of most congestion, but the packets that are delivered will have lower latency (because of the delete and advance drop policy). Responsive flows will see this benefit but they will also be notified sooner of congestion because the oldest, not the newest, enqueued packet will be the one dropped. And most importantly, a responsive flow will see no drops once it has adjusted its load so that it consumes a fair share of the queue. Low bandwidth non-responsive flows will see this benefit as well.

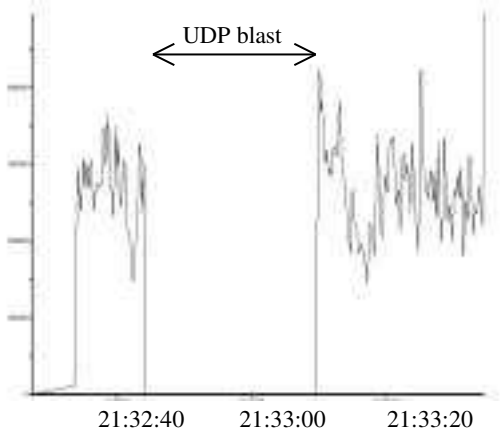
4. Empirical Results

We have implemented RED-DPM within the ALTQ version of the FreeBSD [8]. The RED-DPM implementation maintains a logical queue for each tagged flow and performs a threshold and staleness test on each tagged flow's logical queue immediately prior to the normal RED test.

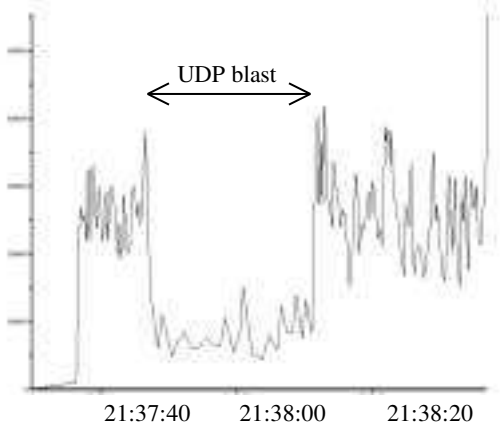
To test the implementation we have constructed a simple network consisting of a two switched 100 Mbps Ethernet LANs that are interconnected by a 10 Mbps Ethernet. FreeBSD routers route traffic between the 100 Mbps Ethernets across the 10 Mbps Ethernet as shown in Figure 1. This 10Mbs network creates a bottleneck to serve as a point of congestion. A series of machines at the edges of the network establish a number of connections to machines on the opposite side of the network. Connections include a mix of TCP connections and UDP connections. In particular, several of the UDP connections are videoconferencing flows generated by a version of the Intel ProShare system (6 flows at approximately 200Kbps each). We also generate a "UDP blast" of unresponsive UDP traffic sent from one machine at the maximum rate available.

Figure 2 reports some of our early results. It compares the performance of TCP (3 machines sending bulk data at the maximum rate available) and UDP connections (Proshare) when the routers use RED queue management and RED-DPM management. Figures 2a and 2b, show the throughput of the TCP connections, first with just TCP and Proshare, then during a period when an unresponsive UDP blast occurs, and finally back to just the TCP and Proshare. Figure 2a shows the performance under RED queue management. Figure 2b shows the performance under RED-DPM queue management. As mentioned above, RED does not effectively protect responsive flows from non-responsive ones. During the UDP blast period, the TCP flows see so many drops that they back off entirely, getting no throughput. Figure 2a confirms this shortcoming. RED-DPM does provide protection by limiting the number of buffers that the unresponsive stream can occupy, thereby ensuring a minimal amount of space exists for TCP flows. The result, as illustrated in Figure 2b is improved throughput for TCP.

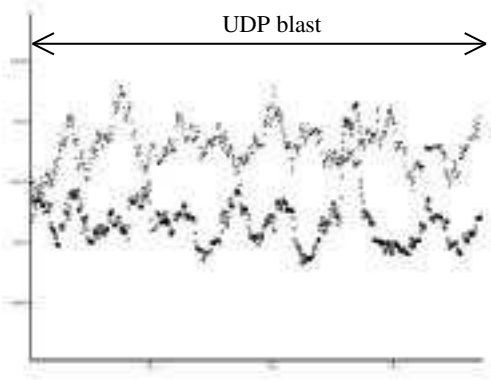
Figure 2c illustrates the impact of RED-DPM on the ProShare (UDP) flows. This shows the latency of the Proshare streams during the time the UDP blast is running. The top set of points is the latency with RED, while the lower, darker points are the latency with RED-DPM. In the case of RED-DPM, although the streams exhibit loss the delete-and-advance mechanism results in lower end-to-end latency than when plain RED is used.



2a) TCP performance under RED. (throughput *v.* time)



2b) TCP performance under RED-DPM. (throughput *v.* time)



2c) ProShare performance. Top line = RED performance, bottom line = RED-DPM (end-to-end latency *v.* time)

Figure 2: Experimental results. Performance of TCP and UDP flows during a UDP "blast."

5. Conclusions and Future Work

We have shown preliminary results which indicate that current active queue management mechanisms, such as RED, can be extended to provide better support for continuous media flows while maintaining or exceeding the current performance offered to "well-behaved" flows. We intend to conduct further comparisons between DPM and FRED, conduct more experiments with different network configurations and traffic patterns, as well as exploring the design space available for thresholds and staleness values.

6. References

[1] R. Braden, Ed., *Requirements for Internet Hosts-Communication Layers*, RFC-1122, October 1989

[2] B. Braden, *et al., Recommendations on Queue Management and Congestion Avoidance in the Internet*, Internet draft draft-irtf-e2e-queue-mgt-01, Feb. 17, 1998.

[3] S. Cen, C. Pu, J. Walpole, *Flow and Congestion Control for Internet Streaming Applications*, Proc. SPIE/ACM Multimedia Computing and Networking '98, San Jose, CA, January 1998, pages 250-264.

[4] D. Clark, J. Wroclawski, *An Approach to Service Allocation in the Internet*, Internet Draft draft-clark-diff-svc-alloc-00, July 1997

[5] S. Floyd, & V. Jacobson, *Random Early Detection gateways for Congestion Avoidance*, IEEE/ACM Trans. on Networking, V.1 N.4, August 1993, p. 397-413.

[6] S. Floyd, S., & K. Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*, February 1998. (Under submission to IEEE/ACM Trans. on Networking.)

[7] V. Jacobson, *Congestion Avoidance and Control*, ACM SIGCOMM '88, August 1988.

[8] C. Kenjiro, *A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers*, Accepted to USENIX '98, Annual Technical Conference, New Orleans, LA, June 1998.

[9] D. Lin & R. Morris, *Dynamics of Random Early Detection*, Proc. SIGCOMM '97.

[10] K. Nichols, V. Jacobson , & L. Zhang, *A Two-bit Differentiated Services Architecture for the Internet*, INTERNET-DRAFT, draft-nichols-diff-svc-arch-00.txt, Nov, 1997.

Adapting Fair Queueing to Improve Loss, Delay, and Admission Control Performance

N.G. Duffield† T. V. Lakshman* D. Stiliadis*

† AT&T Laboratories
Rm A175, 180 Park Avenue
Florham Park,
NJ 07932-0971 USA

* Bell Labs
Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733, USA

Abstract

Fair queueing provides minimum rate guarantees and is fair by being rate proportional, i.e, excess bandwidth due to non-backlogged connections is distributed in proportion to rate guarantees. State dependence of the scheduler is restricted to whether specific connections are backlogged or not. In this paper, we propose three alternate scheduling mechanisms that can provide better service, in terms of losses and delay variation, than fair queueing. The mechanisms preserve the minimum rate guarantees of fair queueing but distribute the excess bandwidth in a manner much more adaptive to instantaneous needs than fair queueing. We compare the performance of the proposed schemes to that of fair queueing using different traffic sources such as voice and video, as well as sources which have aggregate long-range dependent behavior. We find that the proposed schemes, in comparison to packet generalized processor sharing (PGPS), reduce packet losses and curtail the tails of delay distributions for real-time traffic and hence permit the use of significantly smaller playout buffers for the same network load. We argue that these schemes add robustness in measurement-based admission control by providing adaptation for connections whose demands have been underestimated during the measurement process.

I. Introduction

Fair queueing has been the focus of much recent research attention [6], [10], [20], [1] and fair-queueing variants are being increasingly being implemented in switches and routers. Two reasons for use of fair queueing are its ability to provide rate guarantees and its rate-proportionality which ensures fairness. The constraint of fairness and so rate-proportionality limits fair-queueing schedulers in the degree to which they can be state-dependent. The only state dependence in fair-queueing schedulers is on the information about connections being backlogged or not. The price for fairness is reduced state dependence which can translate into increased losses and delay variation. This paper is based on the premise that fairness is not a relevant metric for many applications and instead the relevant metrics are delay bounds, losses, and delay variation. With this in mind, we propose alternate scheduling mechanisms that while preserving the rate guarantees of fair queueing (and therefore preserving its ability to provide worst-case delay bounds to traffic with bounded burstiness) are far more state-dependent. This increased state-dependence causes these schemes to have lower losses and delay variation than fair queueing. The schemes proposed adaptively redistribute only the excess bandwidth, i.e., each connection receives its guaranteed bandwidth even if it is in no danger of violating delay bounds. Three schemes are for excess bandwidth redistribution are proposed in the next section.

The adaptive redistribution gives more of the available bandwidth to connections with the most instantaneous need. Unexpectedly high demands can be made by connections whose bandwidth has been underestimated by measurement during admission control. This can occur because of underlying variability of the demand, or due to sampling error in the measurement process itself. We argue that adaptive redistribution effectively identifies and directs resources to connections whose badly characterized demands would otherwise lead to buffer overflow under purely rate proportional scheduling.

II. Redistributing Excess Bandwidth

Simple Longest Queue First has been shown to minimize the overall packet losses in a system with finite buffers [19]. Here, the scheduler always picks a packet from the longest queue for transmission. However, if the arrival rates and guaranteed shares are such that the expected delays are are not the same for all connections then the longest queue first scheme will always favor high delay (and generally high bandwidth connections). Also, if buffer allocations are not the same, serving the longest queue first may lead to buffer overflow for connections with a low buffer or low relative bandwidth guarantee.

To account for these differences in buffer sizes and bandwidth requirements, we can simply weight the queue size by the allocated bandwidth to make the scheduler a Longest Delay First scheduler. The scheduler then uses the excess bandwidth to try and equalize the delays of all connections in the system and minimize the packet losses if buffer sizes have been allocated proportional to bandwidth requirements. We will refer to this algorithm as FQ-LDF.

Note that this still does not take into account the maximum allowed delay for each connection. An enhancement to FQ-LDF is to explicitly take into account the different delay requirements that applications may have. We would like to be able to allow low maximum delays to voice sources, somewhat higher maximum delays to video sources, and much higher maximum delays for data sources. Also, constant bit rate sources may require extremely low delays, even when their bandwidth allocation is very low. To account for these differences, a maximum delay, that is usually a function of the buffer allocated to a connection, is associated with each connection. The scheduler serves connections that are most likely to exceed their allocated maximum delay. Let D_i be the maximum delay allocated to connection i. Let d_i be the delay that the last packet of the queue associated with that connection at time t may see. The connection that should be served is the one that is more likely to overflow and exceed its delay bound. Thus, connections should be served in increasing order of the $(D_i - d_i)$ values. We call this fair-queuing with least-time to overflow (FQ-LTO) redistribution of excess bandwidth.

The above schemes have not taken into account other characteristics of the source that may be known during admission control. Characteristics could be the maximum burst that can be transmitted by a connection or its peak arrival rate. For example, if sources are leaky bucket shaped, as is the case for sources in the Controlled Load traffic class, we could use this information to predict future arrivals from sources and calculate more precisely the connections that are likely to overflow or see delays higher than the allocated. In systems, where a dual, or multiple leaky bucket is used to shape the input traffic, a similar approach can be used. In fact, we do not require that a connection is shaped by a leaky bucket. All we need is that there be a function specified that provides information as to the maximum burst that can arrive for a connection, as well as the worst-case time that this burst may arrive.

The main idea consists of tracking the state of the connection based on its packet arrivals. This tracking is similar to the policing function that is used to drop packets from connections when they exceed their traffic specification. Based on the state of the connection we can estimate the worst-case arrivals for a given connection and use this to serve connections which are most likely to suffer losses.

An argument that can be made against the proposed adaptive policies, is that a greedy user will always receive more bandwidth. In such a case, users may request less bandwidth and try to receive more by sending more bursty traffic. Limiting such behavior can be explicitly done in many ways. The simplest is policing. If policing is not a viable option, the adaptive policies can themselves be tailored to counter greedy behavior. Instead of picking the user with maximum queue length or the minimum time to overflow, we can, for example, pick the second such candidate. In this case, users that consistently overflow the buffers cannot depend on receiving any excess bandwidth; it will not be in their best interests to be excessively bursty.[2]

III. Performance Studies

We compared by simulation a Packet Generalized Processor Sharing (PGPS) scheduler with our adaptive schemes using a variety of sources to generate traffic. For each connection, the scheduler maintains a separate buffer whose drain time at the guaranteed rate is equal to the connection's maximum allowed delay D_i. The sources used are:

1. Two-state exponential On-Off sources: We used these to emulate compressed voice sources and so used small maximum allowed delays for these sources. The mean load generated per connection is 1/32 of the link capacity.
2. Video teleconference sources: We used the DAR(1) model used in [9] with a mean rate of 1.5 Mbps, a peak-to-mean ratio of 5, and .98 as the one-lag correlation.
3. CBR sources.
4. Data sources. These were modeled as On-Off sources where the on-time is Pareto distributed, and the off-time is exponential. The density of the Pareto distribution is given by

$$f(x) = a^{k-1}(k-1)/x^k, x \geq a = (k-2)/(k-1)\mu \quad (1)$$

We set $k = 5/2$, resulting in infinite variance of the on-times and long-range dependence in the aggregate traffic. The mean on-time was set to 6 packet transmission times and the mean off-time to 200.

We first compared the performance of FQ-LDF with PGPS (which is simulated exactly) using only one type of source. Table I shows packet loss rates for different small buffer sizes from a simulation with 32 voice sources. All connections were served with equal weight and the utilization is about .98. At this high load, PGPS typically has twice the losses of FQ-LDF and redistribution of excess bandwidth to the longest delay queue reduces the aggregate loss rate. We find similar results of lower loss rates for FQ-LDF with video sources as well. With Pareto sources, the loss rates are mostly similar for FQ-LDF and PGPS. This can be accounted for by the heavy-tailed distributions of the on periods.

Apart from packet losses, we also compared the delay distribution resulting from FQ-LDF to that of PGPS. For simulations with 32 voice sources with large enough buffers to avoid packet losses, we find that FQ-LDF needs significantly smaller playout buffers than PGPS. We see the same effect for video sources (plots are not shown for conciseness).

Next, we compare the performance of FQ-LTO with PGPS. FQ-LDF uses the excess bandwidth to equalize delays and it works very well when all sources have the same allowed delays. When different sources have different maximum allowed delays, the FQ-LTO scheduler which takes delays into account is expected to perform better than FQ-LDF

Buffer size in pkts/conn	Cell Loss Ratio:PGPS	Cell Loss Ratio:FQ-LDF
75	13.18e-3	8.3e-3
100	8.198e-3	4.938e-3
125	5.61e-3	3.61e-3
150	4.245e-3	2.849e-3
250	2.061e-3	1.025e-3
350	9.614e-4	1.054e-4
425	4.876e-4	$< 10^{-6}$
500	2.8478e-4	$< 10^{-6}$

TABLE I
CELL LOSS RATIOS FOR 32 EXPONENTIAL ON-OFF SOURCES WITH SHORT BUFFERS

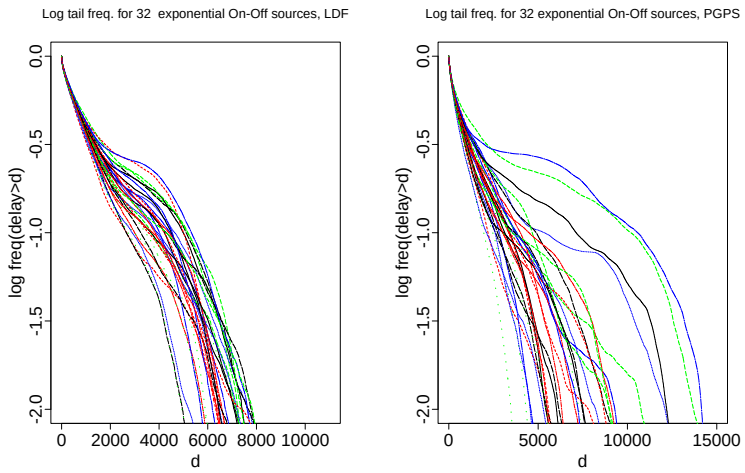


Fig. 1. Log tail frequencies for 32 exponential On-Off sources, according to service discipline (FQ-LDF or PGPS).

and PGPS. We use 32 sources with 8 sources from each of the types described before. CBR sources are guaranteed their mean rate. They generate at most 2 back-to-back packets and have a small buffer allocation of 5 packets. Voice sources are also guaranteed their mean rate and have a maximum buffer allocation of 1200 packets (about 10 ms delay at their guaranteed rates). Video sources are guaranteed two times their mean rates and have a buffer of 12000 packets. Data sources are given a large buffer of 40000 packets. Since they are not delay sensitive, they are guaranteed a rate below their mean rate. Data sources are expected to use some the excess bandwidth from the other classes to make up for their allocation below their means.

The PGPS scheduler gives excess bandwidths to the data class in proportion to the weights. Since the data class has a very large delay allowance, the FQ-LTO scheduler gives priority to the low delay classes in the distribution of excess bandwidth. Consequently, the lower delay classes have very compact delay distributions whereas the very high delay data class (which also has underallocated guaranteed bandwidth) has better tail behavior with PGPS. The FQ-LTO scheduler achieves what one would want in this mixed scenario. Worst case delay bounds are always met even for data (if it is given appropriate guaranteed rates). The lower delay classes have very small tails and hence need drastically lower playout buffers than when using PGPS.

IV. ADAPTATION TO ERRORS IN ADMISSION CONTROL

The predominant approach to admission control has focused on characterizing future bandwidth demands of a connection. If the source of a flow and the network can agree on the characterization, resources can be allocated by the network in order that the performance obtained for the source meets its expectations. Characterization can be done on the basis of declared parameters, or by measurement, or by some combination of these. Many applications, variations and combinations of measurement-based characterization are possible. These include measurement of a single connection, or an aggregate; for admission to a dedicated resource, or a shared one; measurement of admitted connections, or those seeking admission. We refer the reader to [2], [4], [5], [11], [14], [15] for examples of work in this area.

Measurement-based characterization of the bandwidth can fail to be accurate for three reasons: systematic errors, errors due to high variability, and measurement errors. Systematic error occurs when an underlying nonstationarity causes traffic characteristics to depart from their (initially) measured values. High variability, as manifest by stationary long-range dependent processes (e.g. fractional Brownian motion can lead to local trends away from measured values; [18]. Such LRD traffic models have been proposed on the basis of empirical studies; see e.g. [17]. In practice it may be difficult to distinguish between non-stationarity and long-range dependence. The consequences for admission control are, however, much the same: long periods for which the offered load exceeds that expected from measurements. (Here "long" is with respect to the drain-time of a full buffer). Measurement errors occur because measurement entails sampling and is hence subject to statistical error. Such errors can lead to overadmission through underestimation of the required bandwidth, resulting in an attained quality lower than the target. For bufferless resources, Grossglauser and Tse [12] have recently established the extent to which admission controls must be conservative in order that measurement errors do not compromise the target quality with greater than the target frequency. However, such errors will still occur. An analysis of the buffered case is presented in [7].

Whatever the source and frequency of errors in bandwidth characterization, it is desirable to have bandwidth allocation mechanisms which can adapt to provide or increase robustness. Scheduling can provide this, and we believe in particular that adaptive redistribution of excess bandwidth can be an effective means of adapting to badly characterized traffic. Consider, for example, when individually measured sources are each to be provided a guaranteed rate out of individual or shared buffers. If some true source rates are greater than expected, then adaptive excess redistribution policies described above will selectively direct resources to these connections. Thus we expect them to perform better than FQ in preventing the buildup of large occupancies and delays, in the buffers used or shared by these connections. We shall report detailed results elsewhere.

V. Concluding Remarks

Unless the fairness requirement of GPS is required for policy reasons, fair queueing systems can improve their performance by adaptive redistribution of excess bandwidth without losing their worst case fairness property or their ability to guarantee worst case delay bounds. For many applications, it is sufficient that schedulers provide minimum rate guarantees and use state-dependent excess bandwidth allocation so as to curtail delay-distribution tails or offset measurement-errors in admission control.

References

[1] J.C.R. Bennett, H. Zhang, "WF^2Q: Worst-case Fair Weighted Fair Queueing", *Proceedings of INFOCOM'96*, pp. 120-128, 1996.

[2] C. Casetti, J. Kurose, D. Towsley, "An Adaptive Algorithm for Measurement-based Admission Control in Integrated Services Packet Networks," Int. Workshop on Protocols for High Speed Networks, (Sophia Antipolis, Oct. 1996).

[3] D. Clark, S. Shenker, L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", *Proceedings of ACM SIGCOMM 1992*, pp. 14-23.

[4] C. Courcoubetis, G. Kesidis, A. Ridder, J. Walrand and R. Webber, "Call Aceeptance and Routing using Inferences from Measured Buffer Occupancy," IEEE Trans. Comm., Vol. 43, No. 2/3/4, pp. 1778-1784, 1995.

[5] S. Crosby, I. Leslie, J. Lewis, R. Russell, F. Toomey & B. McGurk, "Practical Connection Admission Control for ATM Networks Based on On-line Measurements", Proceedings IEEE ATM '97, June 1997, Lisbon.

[6] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm" *Internetworking: Research and Experience*, pp, 3-26, vol. 1, 1990.

[7] N.G. Duffield, "A large deviation analysis of errors in measurement based admission control to buffered and bufferless resources", AT&T Laboratories, 1998.

[8] N.G. Duffield, T.V. Lakshman, D. Stiliadis "On Adaptive Bandwidth Sharing with Rate Guarantees," *Proceeding of Infocom 1998*

[9] A. Elwalid, D. Heyman, T. V. Lakshman, D. Mitra, A. Weiss, "Fundamental Bounds and Approximations for ATM Multiplexers with Applications to Video Teleconferencing", *IEEE Journal on Selected Areas in Communications: Special Issue on Fundamental Advances in Networking*, 13:1004–1016, 1995.

[10] S. J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications", *Proceedings of INFOCOM'94*, pp. 636-646., 1994.

[11] R.J. Gibbens, F.P. Kelly & P.B. Key, "A decision-theoretic approach to call admission control in ATM networks" IEEE Journal on Selected Areas of Communications: Special Issue on Fundamental Advances in Networking, 13:1101–1114, 1995.

[12] M. Grossglauser and D.N.C. Tse "A Framework for robust measurement-based admission control", In Proc. ACM SIGCOMM 97, Cannes, France, September 1997.

[13] P. Goyal, H.M. Vin and H. Chen, "Start-time Fair Queing: A Scheduling Algorithm for Integrated Services Packet Switching Networks", *Proceedings ACM SIGCOMM'96*, pp. 157-169.

[14] S. Jamin, P.B. Danzig, S. Shenker & L. Zhang, "A measurement based admission control algorithm for integrated services packet networks", in Proc. ACM SIGCOMM '95, 1995.

[15] E. Knightly, "Second Moment Resource Allocation in Multi-Service Netw orks," in: Proceedings of ACM SIGMETRICS '97, Seattle, WA, June 1997.

[16] D. S. Lee, "Weighted Longest Queue First: An Adaptive Scheduling Discipline for ATM Networks", *Proceedings INFOCOM'97*, 1997.

[17] Leland, W. E., Taqqu, M. S., Willinger, W. and Wilson, D. V. (1994) On the self-similar nature of Ethernet traffic, *IEEE/ACM Trans. Networking*, 2, 1–15.

[18] B.B. Mandelbrot and J.R. Wallis (1968). "Noah, Joseph, and Operational Hydrology". *Water Resour. Res.,* **4** 909–918

[19] S. Panwar, D. Towsley, J. Wolf, "Optimal scheduling policies for a class of queues with customer deadlines to the beginning of services", *Journal of ACM*, vol. 35, no. 4, pp. 832—844, 1988.

[20] D. Stiliadis and A. Varma, "Design and analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks", *Proceedings of ACM SIGMETRICS '96*, pp. 104-115, May 1996.

An Empirical Study of Client Interactions with a Continuous-Media Courseware Server *

Jitendra Padhye Jim Kurose
{jitu,kurose}@cs.umass.edu
Department of Computer Science
University of Massachusetts
Amherst MA 01003 USA

Abstract

While considerable research has gone into investigating various networking and operating system mechanisms for supporting the transfer and playout of stored continuous media (e.g., audio and video), very little information is available about how users actually use such systems. Understanding how users interact with such a system – i.e., developing a user workload characterization – is crucial in designing and evaluating efficient continuous media (CM) resource allocation and access mechanisms. We have designed and built an interactive WWW-based, multimedia, client/server application, known as MANIC (Multimedia Asynchronous Networked Individualized Courseware), that streams synchronized CM (currently audio) and HTML documents to remote users. MANIC was used by more than 200 users during the Spring 1997 semester to listen to, and view, the stored audio lectures and lecture notes for a full-semester senior-level course at the University of Massachusetts. In this paper we provide empirical and analytic characterizations of observed user behavior in MANIC. We characterize both session-level behavior (e.g., the length of individual sessions) as well as interactive user behavior (e.g., the time between starting/stopping/pausing the audio within a session; and the "distance" of rewinding, fast forwarding and indexed jumping within the audio stream). We also examine how well our measured data can be fit analytically by various distribution functions. Finally, we consider the possible implications of our results for CM system architecture.

*This material was supported in part by the National Science Foundation under Grant No. CDA-9502639, NCR-9508274, and the University of Massachusetts through an Instructional Technology Grant. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the University of Massachusetts.

1 Introduction

Efficient delivery of continuous media (CM) such as audio and video, is one the important goals of current networking and operating systems research [6, 10, 17]. Considerable research has gone into investigating various networking and operating system mechanisms to make continuous media transfer more efficient. However, little information is available about how users actually *use* such systems. Understanding how users interact with such a system – i.e., developing a user workload characterization – is crucial in designing and evaluating efficient CM resource allocation and access mechanisms. For example, understanding how often users fast-forward, rewind, pause or make indexed-jumps within a continuous media stream (and how "far" they move within the media stream when they do so), can help in designing and evaluating call admission strategies [3], scalable compression techniques [18], and client/server buffer management strategies [4] for CM systems.

In this paper we provide an empirical characterization of observed user behavior with MANIC (Multimedia Asynchronous Networked Individualized Courseware) [8], a WWW based multimedia client/server application, that streams synchronized CM (currently audio) and HTML documents to remote users. MANIC was used by more than 200 users during the Spring 1997 semester to listen to, and view, stored synchronized audio lectures and lecture notes for a full-semester senior/graduate-level course at the University of Massachusetts. Our particular focus here is on both session-level characteristics (e.g., the length of the individual sessions) as well as *interactive* user behavior (e.g., the time between starting/stopping/pausing the audio within a session; and the "distance" of rewinding, fast forwarding and indexed jumping within the audio stream). Our goal here is *not* to evaluate the performance of the MANIC system. Instead, our goal to characterize user behavior and the resulting load placed on the network and server resources in such a CM system. We also consider the architectural de-

sign choices implied by such loads.

Our measurements indicate a high degree of user interactivity. We find that once audio playout begins, in more than 45% of the cases it lasts less than 3 minutes before the user pauses or jumps elsewhere in the audio stream; the average length of time spent listening to an audio stream before a pause or jump, though, is slightly over 8 minutes. From a modeling perspective, we find that session lengths, and the length of time a user spends in continuous audio playout (before a stop, pause, or jump) or with playout halted (before resuming audio playout) has a "heavier" tail than the exponential distribution that is typically assumed in previous CM analytic modeling [3, 18]; both the lognormal and gamma distribution are shown to provide a closer fit to our observed data than the exponential distribution.

Surprisingly, we also find that forward jumps in the audio stream are more than seven times more likely than backwards jumps (we had expected to see a relatively larger number of short backward jumps, as a student reviewed a not-understood portion of the lecture). We also find that while there are a significant number of jumps over relatively short periods of time (e.g., approximately a third of the jumps were less than 3 minutes forward in the audio stream), the average length of a jump was much larger (over two thousand seconds). Finally, at the session level, we find that the average session length is slightly less than half an hour, and that the vast majority of the sessions (more than 80%) are less than an hour in length.

We believe that while these measurements are themselves of independent interest, they are also of broader value. From a system architecture standpoint, the level of observed user interactivity suggests that mechanisms designed specifically to support the uninterrupted, long-term playout of CM (e.g., VoD servers for entertainment movies) may not be well-suited for the more interactive educational/training environments. From a modeling perspective, our results question the accuracy of some previously adopted modeling assumptions [3, 18] and provide specific "real-world" parameters values that can be employed in future studies. We stress, however, that our results are for a specific application, a specific set of multimedia course material, and a specific user population. Other populations and other servers with different CM applications and/or content may well exhibit significantly different user behavior. Nonetheless, the present work does provide an important first data point characterizing the interactive behavior of users in an operational CM server.

The remainder of this paper is organized as follows. Section 2 provides an overview of the MANIC architecture, describing the CM content and the user population. In Section 3, we describe how user interactions are captured and present empirical session-level characteristics. In section 4, we examine the finer-grained behavior of users within a session. Section 5 discusses implications of our measurements and Section 6 concludes the paper.

2 Overview of the MANIC Architecture and Content

In this section, we overview the MANIC architecture and discuss the structure of the multimedia course content that was accessed by students. For a detailed description of MANIC, the reader is referred to [8]. The CM course material that was accessed during this study can itself can be accessed via: `http://gaia.cs.umass.edu/cs653-1997/`

The architecture of the MANIC system is depicted in Figure 1. In its primary mode of operation, the system allows the user (client) to listen to pre-recored lectures with accompanying text. Pages of text ("slides") are displayed as HTML files in the browser window. Audio is delivered by the server using the RealAudio Server, and played out at the client side user the RealAudio browser plugin [14]. Slide display and audio playout are synchronized i.e., the displayed HTML contents in the WWW browser window change automatically as the audio progresses. This is achieved using the synchronized multimedia facility provided by the RealAudio server/player. Details about this synchronization mechanism, and how it is exploited in MANIC, can be found in [8]. In addition, MANIC can highlight (and then de-highlight) different portions of the text as the audio progresses, as discussed below.

Several CGI scripts running on the WWW server allow the user to perform various control/browsing actions by clicking on the control buttons provided within each slide. These commands include stopping and restarting the audio, moving to the previous or the next slide and jumping to arbitrary slides via an index. A detailed description of the implementation is provided in [8]. Here, we only describe what these commands do, and how they affect audio playout.

A screen shot of the MANIC user interface displaying a typical slide is shown in Figure 2. The slide contains two bullet points and a diagram. As noted above, the contents of a slide are partitioned into predefined groups that can be individually highlighted as the corresponding audio is played out. Highlighting is achieved rather crudely in this version of MANIC by loading a new HTML page whenever a different highlight is to be displayed. All HTML pages associated with a given slide have the same text and graphics content, but different font color tags depending upon which portion is to be highlighted. In Figure 2, there are two such "highlights", each corresponding to a bullet point[1]. The audio corresponding to the first highlight was playing when the

[1]This fact is not evident from the black and white screen shot. A highlight can contain any number of bullet points and/or arbitrary HTML code.

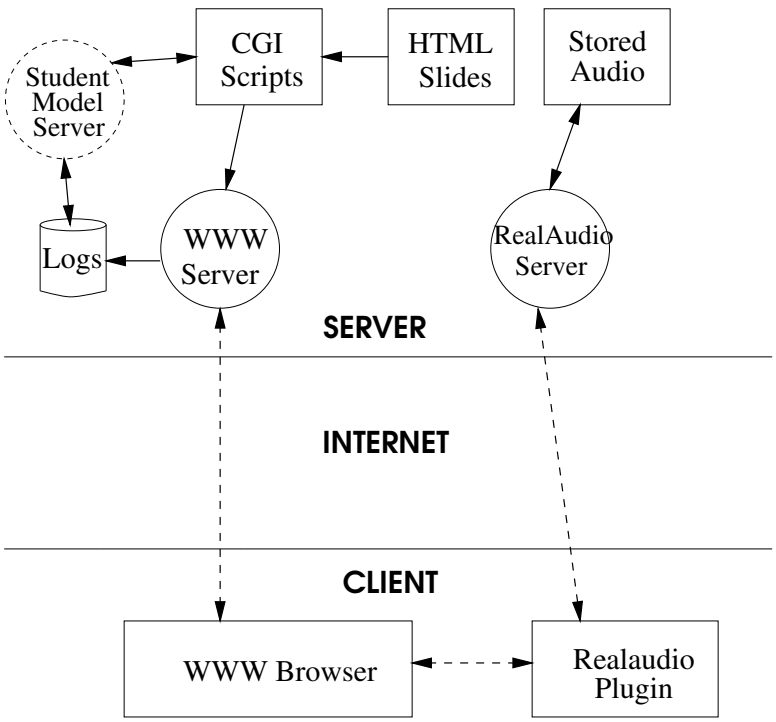


Figure 1: MANIC architecture

screen shot was taken. A highlight is the smallest unit of user interaction with MANIC system. That is, the smallest granularity with which a user can index or jump through the audio stream is by one highlight at a time.

Let us now consider the user commands in MANIC. As shown in Figure 2, the following commands are available:

- **Continuous Play:** This command starts the audio playout at the current highlight. The audio will continue to play uninterrupted until the end of the current lecture, unless stopped early by the user. Slides and highlights change as the audio progresses. Once the end of the lecture is reached, the audio stops playing[2].
- **Play Current Slide:** This command starts audio playout at the beginning of the current highlight. Audio playout stops once the end of the current slide is reached.
- **Next, Previous:** These commands stop the audio if it was playing, and display the next or previous slide in the lecture in the browser window, respectively.
- **Up, Down:** These commands stop the audio if it was playing, and move the highlight to next or previous highlight within the current slide, respectively. The Up and Down commands provide the means for "fast forwarding" or "rewinding" through material at the granularity of a highlight - the smallest possible unit of forwards or backwards jumping available within MANIC[3].
- **Index:** This command brings up a window that displays an index of course material by lecture number, table of contents or keyword search. The audio continues to play while the index is displayed. However, if the user clicks on an index entry, the audio stops playing and the slide corresponding to the table entry is displayed in the browser window.
- **Stop Playing:** This command stops the audio playout. The slide displayed in the browser window remains unchanged.

In each case when the audio stops, the user must explicitly restart playout by using one of the two "play" commands. We note that while MANIC provides the capability to start/stop and index through audio (i.e., by using the

[2]Slides are grouped together into "lectures", corresponding to the actual lectures given live to in-class students. Each lecture is about 70 minutes in length. Each lecture was stored in a separate RealAudio file for ease of management. Since audio playout stops at the end of a lecture, the user must click on the "Next" button and then use one of the two "play" commands to restart the audio. Note that this limits the maximum length of continuous audio playout to the length of lecture.

[3]MANIC also provides a "non-highlight" mode [8] for users with slower network connections. In this mode, the user can FF/rewind only at the granularity of a single slide. Approximately 10% of the downloaded slides were viewed in non-highlighted mode.

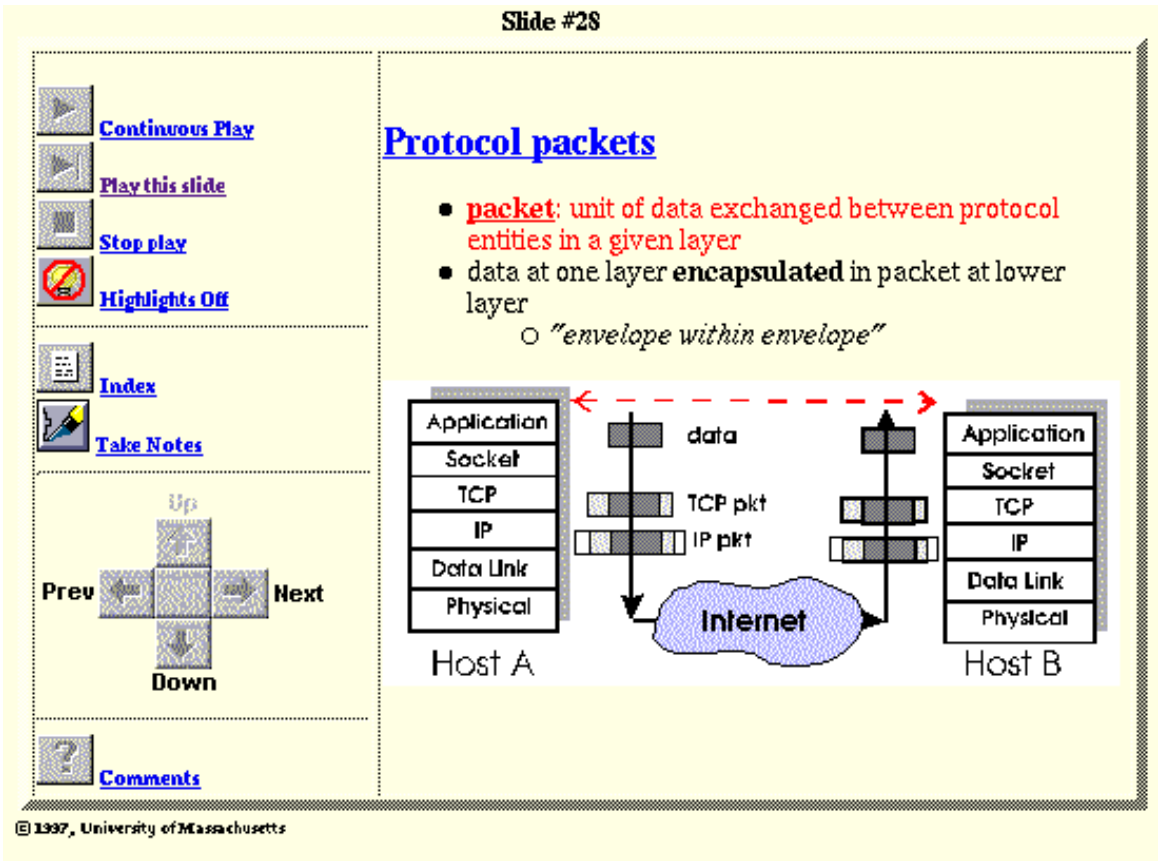


Figure 2: MANIC user interface

commands described above), a student can also manipulate audio via a slider bar in the Real Audio pop-up window (although we explicitly encouraged students not to do so); we discuss the difficulties posed by this second way to control audio shortly.

The brief description above has discussed MANIC's functionality as a CM-on-demand server. The client can start and stop the audio playout. Fast-forward, rewind and indexed-jumps are affected by jumping to predefined, fixed points in the audio stream, using the Next, Previous, Up, Down, and Index controls.

2.1 System Parameters and User Population

The current version of MANIC is designed around a 25-connection RealAudio server and a NCSA WWW server. Both the audio and the WWW server run on a Silicon Graphics *Challenge* workstation with two CPUs, 256 MB of RAM and 64 Gigabytes of hard disk space. Users can access course material from a variety of PC and workstation platforms using Netscape with a RealAudio plugin. While the audio server is capable of handling 25 simultaneous connections, we have seldom seen more than 5-7 users active at the same time.

The system was used to deliver a 3-credit, senior/grad level course on Computer Networks taught at the University of Massachusetts at Amherst, during the Spring 97 semester. The course itself was offered live to approximately 120 registered on-campus students and (by satellite) to 40 off-campus students. The audio and video of the live presentation of the course (instructor lectures, questions and answers, discussion) were professionally recorded by the UMass Video Instructional Program and the audio encoded in RealAudio format for 14.4 Kbps access (in a few cases, lectures were encoded for 28.8 Kbps access). Each lecture was made available on-line within 1 or 2 days after it was delivered in the classroom. Overall, the course consisted of 25 lectures, each of which was approximately 70 minutes in length. There were approximately 400 HTML slides.

It is important to note that the course was offered live, at the same time it was being made available to students via MANIC. It is likely, therefore, that a student using MANIC to view part of a lecture may well have already attended that lecture live. We note that the behavior of students viewing material for the first time may well be different from students reviewing material. We have anecdotal evidence,

however, that MANIC was also used to make up for missed classes, and several students admitted (after the class was over) to having "attended" more classes via MANIC than in person. Since the course was made available on the Internet free of charge, there were also additional users who had not officially registered for the class.

Before accessing MANIC for the first time, each user was required to select a unique (potentially anonymous) username. The users were expected to employ the same username each time they subsequently accessed the system. During the period of the Spring semester, we recorded 220 unique usernames. For each username, a separate activity log was maintained. While it is possible that some students may have used multiple aliases and changed their identity from one session to the next, we believe that the effect of any aliasing is minimal, particularly because we characterize aggregate behavior rather than the behavior of a particular user.

3 Measurements and Session-level Characteristics

The activity of each MANIC user (for the particular course described above) was logged by the MANIC server throughout the semester. These per-user logs were then analyzed off line to produce the user characterizations described in this section and the next. The 215 user logs that were analyzed contained more than 33,000 such log entries[4]. In addition to the logs kept by the MANIC server, the Real Audio server also kept its own logs of audio-only activity. As discussed in the Appendix, users occasionally directly manipulated the audio using the Real Audio pop up window (rather than the MANIC audio controls). In such cases, we thus had to coordinate the Real Audio logs with our MANIC logs.

Let us begin by describing the format and the contents of the MANIC activity logs. Recall that each user is identified by a unique username. This username is transmitted in all HTTP requests through the use of *cookies*. This allows the WWW server to maintain logs of actions performed by individual users. Each log entry specifies either a user command (such as stop, play, previous/next/up/down) or the slide and highlight sent by the server to the user's browser window. The log entries are timestamped with a granularity of 1 second.

The log fragment shown in Figure 3 is a typical example of the logs generated by the MANIC system. Lines 1 and 2 indicate that the user started on the first highlight of slide 1. The audio played continuously (line 3) until partly into the second highlight of slide 2. After that, the user used the index (line 7) to jump to slide 10 (line 8). Note that the audio stops playing at this point. The user then moved forward two slides by using the "next" button (lines 9-12). The audio was restarted from the first highlight of slide 12 (lines 12-13), and it was stopped shortly thereafter (line 15). The last line (line 16) simply indicates that when the user hit the "stop" button, slide 12 was reloaded. This is an artifact of the way we update highlights and maintain log entries.

3.1 Session Length

Perhaps the most important session-level user characterization is the length of each session itself, i.e., the length of time the student is sitting at the computer using MANIC before he/she leaves. This value is easy to measure if there are explicit log-in and log-off procedures. Although we instrumented MANIC with a log-on procedure for this study, this explicit login procedure could be bypassed by the savvy user. More importantly, we chose *not* to have an explicit log-off procedure in MANIC, primarily because we could not ensure that students would indeed explicitly log-off when done. (They could simply walk away from, or turn off, their PC or workstation when their session was "over.")

Without a specific logout procedure, it is difficult to measure a session's length, or indeed even define what is meant by a "session." We thus adopted the following approach to *estimate* session length. In order to define a session, we first define a Session Gap Threshold (SGT).

The SGT is a length of time such that if there is no activity (no user commands, no audio playing) by a given user for this length a time, that user's "session" is considered to have terminated. To determine a reasonable value for the SGT, we plot the number of "sessions" that would be present in the logs for various values of SGT, as shown in Figure 4. From Figure 4, it can be seen that a reasonable value of the SGT would be approximately 30 minutes, as the slope of the graph becomes very close to zero after that. This implies that a user that is inactive for more than 30 minutes is likely to remain inactive for a much longer. Thus, for all further measurements, we assume that an SGT of 30 minutes defined "sessions."

Having selected a value for the SGT, we were able to measure the lengths of individual sessions. The distribution of session lengths indicates the pattern of user "holding times" i.e., the amount of time each user spends in the system. In Figure 5 we plot the distribution of session lengths with an SGT of 30 minutes. The leftmost graph in Figure 5 plots the distribution of session lengths as a histogram. We have normalized the heights of the bars so that the area under the histogram is one. The average session length is 1731 seconds (approximately a half hour); the standard deviation is 2662.

The leftmost graph in Figure 5 also plots the three fitted

[4]This data is available on-line. See:
`ftp://gaia.cs.umass/edu/pub/jitu/MANIClogs/README`

```
 1 Sat Apr 26 16:00:42 1997- -start-1
 2 Sat Apr 26 16:00:42 1997-slide=1&buttons=N&highlight=1-1
 3 Sat Apr 26 16:00:47 1997- -continuous_play-1
 4 Sat Apr 26 16:00:48 1997-slide=1&buttons=N&highlight=1-1
 5 Sat Apr 26 16:02:14 1997-slide=2&buttons=DPN&highlight=1-1
 6 Sat Apr 26 16:04:34 1997-slide=2&buttons=UDPN&highlight=2-1
 7 Sat Apr 26 16:04:42 1997- -index-1
 8 Sat Apr 26 16:04:43 1997-slide=10&buttons=PN&highlight=1-1
 9 Sat Apr 26 16:04:47 1997- -next-1
10 Sat Apr 26 16:04:47 1997-slide=11&buttons=DPN&highlight=1-1
11 Sat Apr 26 16:04:49 1997- -next-1
12 Sat Apr 26 16:04:49 1997-slide=12&buttons=DPN&highlight=1-1
13 Sat Apr 26 16:04:52 1997- -continuous_play-1
14 Sat Apr 26 16:04:59 1997-slide=12&buttons=DPN&highlight=1-1
15 Sat Apr 26 16:05:12 1997- -stop-1
16 Sat Apr 26 16:05:13 1997-slide=12&buttons=DPN&highlight=1-1
```

Figure 3: Example of MANIC user logs

distributions: an exponential distribution of the form

$$f(x) = \lambda e^{-\lambda x},$$

a gamma distribution of the form

$$f(x) = \frac{\lambda_g e^{-\lambda_g x} (\lambda_g x)^{\alpha - 1}}{\Gamma \alpha}$$

and a lognormal distribution of the form

$$f(x) = \frac{1}{x(2\pi\sigma^2)^{0.5}} e^{\frac{-(ln(x)-\mu)^2}{2\sigma^2}}.$$

The distribution parameters $\lambda, \lambda_g, \alpha, \sigma^2, \mu$ were obtained by the method of moments - matching the mean (in the case of the exponential distribution) or the mean and variance (in the case of the gamma and lognormal distributions) of the fitted distributions to their empirically observed values. By informal inspection, we note that gamma distribution most closely fits the measured data. We also note that the exponential distribution tends to underestimate the probability mass for both short and long sessions.

The right graph in Figure 5 plots $1 - F(x)$ (where F(x) is the CDF of the probability distribution) on a logarithmic y-axis to highlight the tail behavior. Here we see that both the gamma and lognormal distributions tend to match the tail well, while the exponential distributions tends to underestimate the tail of the distribution. Unlike the heavy-tailed phenomena observed in physical systems such a reservoir levels and network traffic [12, 7], there is a very simple explanation for the heavy tail observed here – some users simple chose to view course material for a relatively long period of time.

Figure 6 presents the same data from another perspective. Here, we divide the sessions into two groups, depending on whether or not a session played any audio at all. (We will see shortly that that slightly under half of sessions did not play any audio at all.) Figure 6 shows that sessions with audio tend to last much longer session than those without audio. The mean session length for those with with audio is 2827.22 seconds, while the mean session length for those without audio is 304.52 seconds.

4 User Activity Within a Session

Having examined the length of a user session, we next characterize user behavior within a session.

Perhaps the coarsest characterization of user behavior with respect to the CM content is the fraction of time within a session that the user actually utilizes (listens to) the CM content. Recall that within MANIC, a user need not always be receiving streamed audio. For example, a user may choose to turn audio off while browsing through the HTML lecture slides (or other WWW documents). For example, the log fragment shown in Figure 3 indicates that the user had the audio on between 16:00:48 and 16:04:42 (lines 4-7), off (as the result of the indexed jump to a new slide in line 8), and then on again between 16:04:59 and 16:05:12 (lines 13-15).

We define a session's "audio utilization" to be the ratio of the amount of time the audio is actually being played during a session to the session length. The histogram in Figure 7 plots the fraction of sessions (y-axis) that were observed to have a given audio utilization (x-axis). The average audio utilization is 67.59% and the standard deviation is 27.50. These numbers are calculated only for those sessions that include at least some audio playout. Recall that almost 50% of the sessions contain no audio playout at all. The average audio utilization of *all* sessions is 37.50% and the

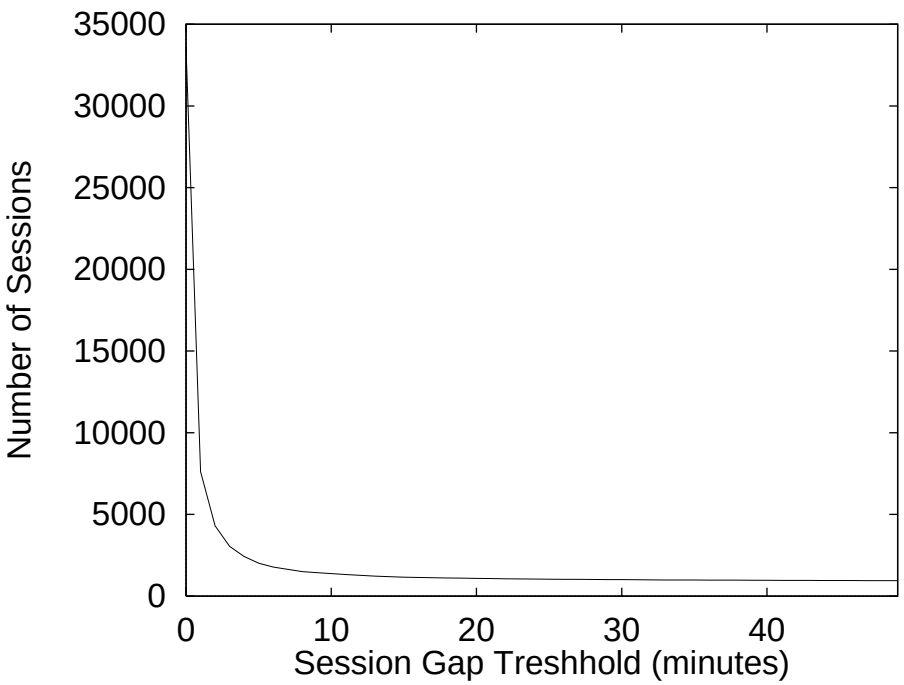


Figure 4: Number of sessions as a function of SGT

standard deviation is 39.34. We note that the relatively low average utilization indicates that a CM server that allocates resources on the basis of worst case demand (e.g., assumes that each session will always be playing audio) will severely over-reserve resources.

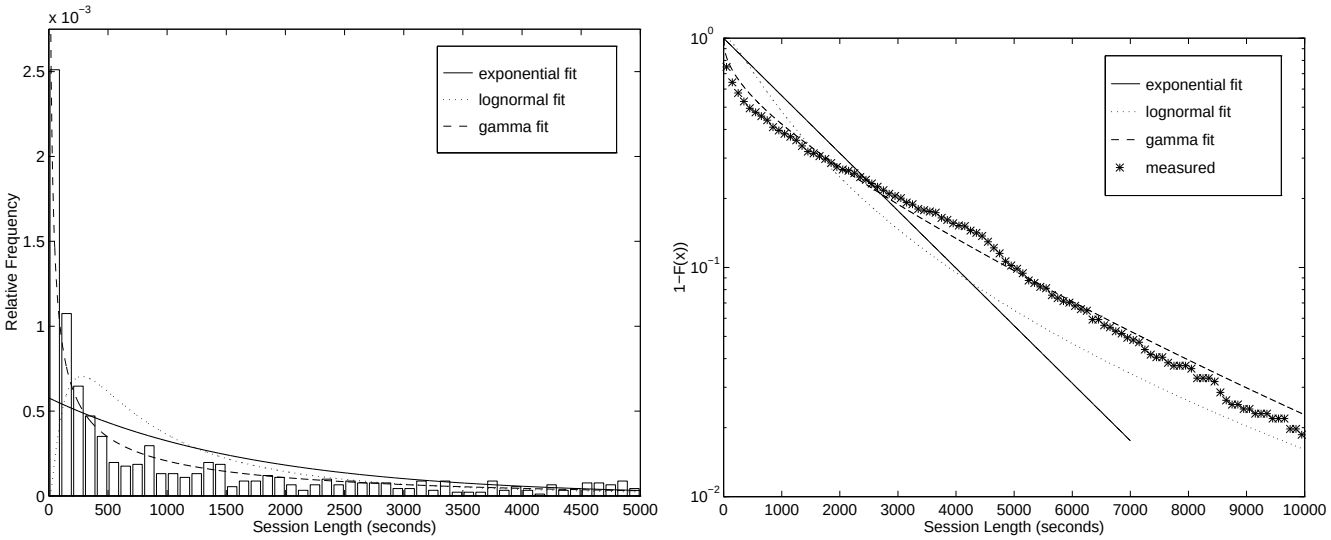
As discussed above, an individual user's CM activity within a session will consist of alternating intervals of having the audio on and off. Indeed, a number of analytic models of users in interactive CM systems explicitly model this on/off behavior [3, 18]. Figure 8 plots the distribution of the length of the individual audio ON periods. Once again, the heights of the histogram bars have been normalized so that the area under the histogram is one. Interestingly, more than 45% of the audio ON periods are less than 180 seconds in length. The average length of an audio ON period is 509.50 seconds and the standard deviation is 815.92. The bump at 4200 seconds corresponds to approximately 70 minutes of continuous audio playout, the approximate length of an individual lecture.

Figure 8 also plots fitted exponential, gamma, and lognormal distributions. We see again that the exponential distribution appears to provide the poorest fit, while both the lognormal and gamma distributions match the distribution of the observed data over a wide interval.

Figure 9 plots the distribution of the time spent between periods of audio playout – time when no CM is being streamed to the user. During this time, users may be browsing through class slides (as discussed below), browsing other WWW documents, thinking, or sleeping. The average length of silence between successive audio subsessions is 171.91 seconds; the standard deviation is 392.24. The fitted distributions again indicate that the gamma distribution is a particularly good fit to the observed data (except at very large tail values), as is the lognormal distribution. Once again, we see that the exponential distribution is a poorer fit for both small and large values of the audio OFF period.

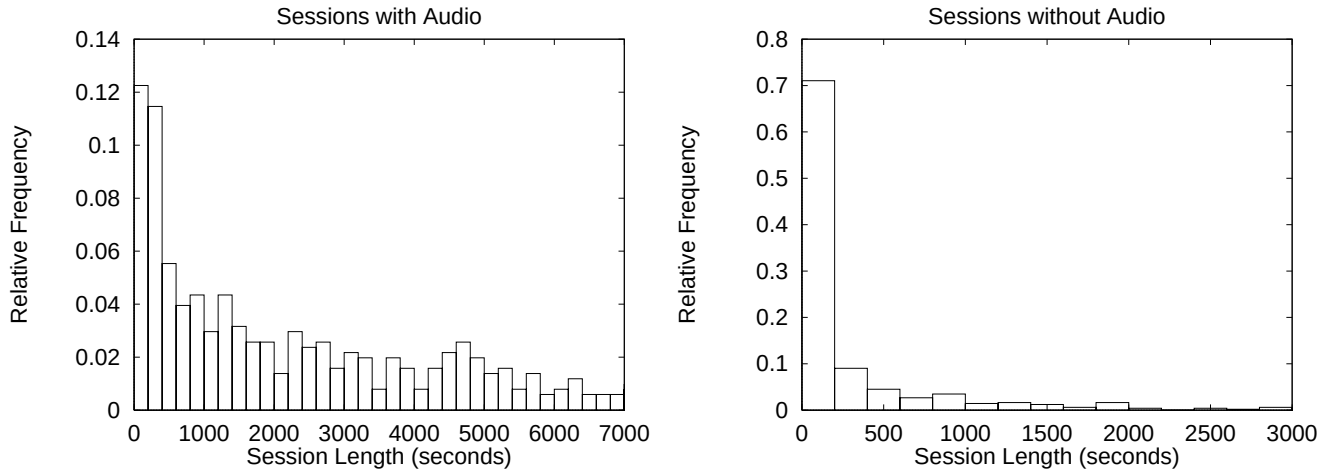
As discussed above, after a user ends an audio ON period, another audio ON period may eventually be initiated (following the audio OFF period). When re-initiating audio playout, a user may continue the audio where he/she had left off, or may choose to jump to a new point in the audio stream (perhaps, for example, via the use of the index). As an example, consider the log fragment shown in Figure 3. The audio-off period begins at clock time 16:04:42 (line 7), ends at clock time 16:04:52 (line 13) with a forward jump in the audio stream.

We now define the length of a "jump" in the audio stream more precisely. Consider the log fragment in Figure 3. In that example, the user has jumped from slide 2, highlight 2 (line 6) to slide 12, highlight 1 (line 12), between two audio subsessions. To measure the length of this jump, we construct a timeline as follows: we consider slide 1, highlight 1 to begin at time 0, and measure the time continuously, so that the last highlight of the last slide of the last lecture (highlight 1, slide 393, lecture 25) is considered to end at time 101685.9. On this timeline, slide 2, highlight 2 begins



(a) (b)

Figure 5: Session length characteristics: empirical measurements and fit data



(a) (b)

Figure 6: Sessions with and without audio playout

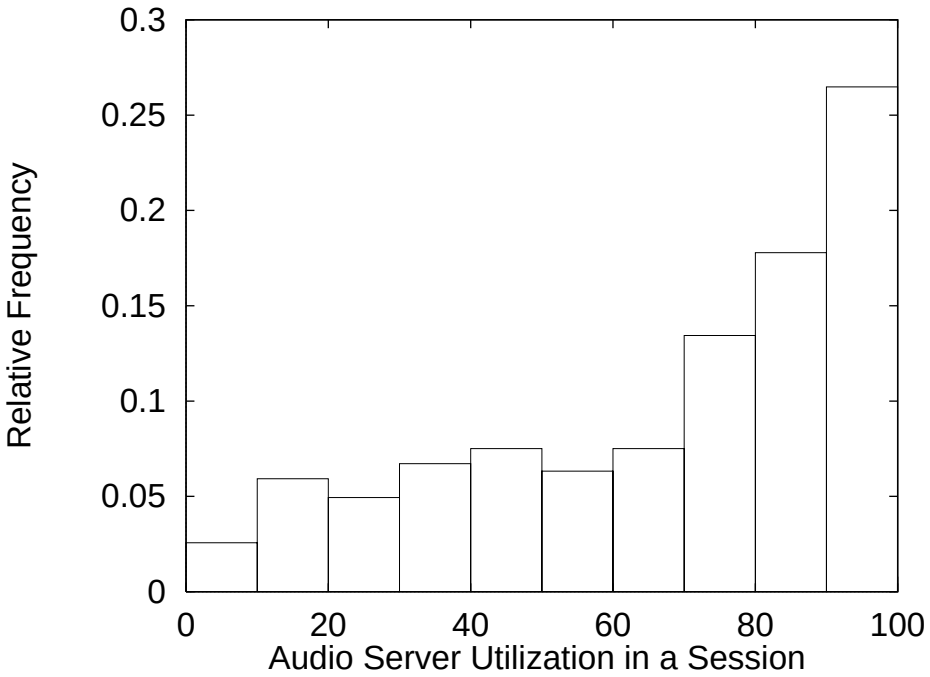
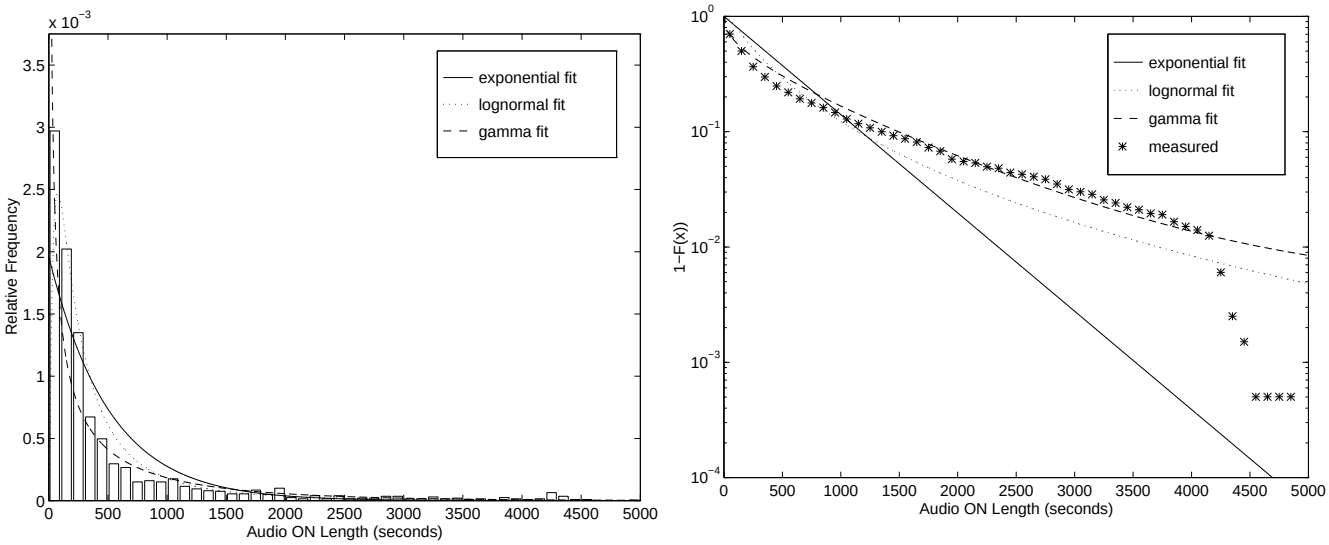
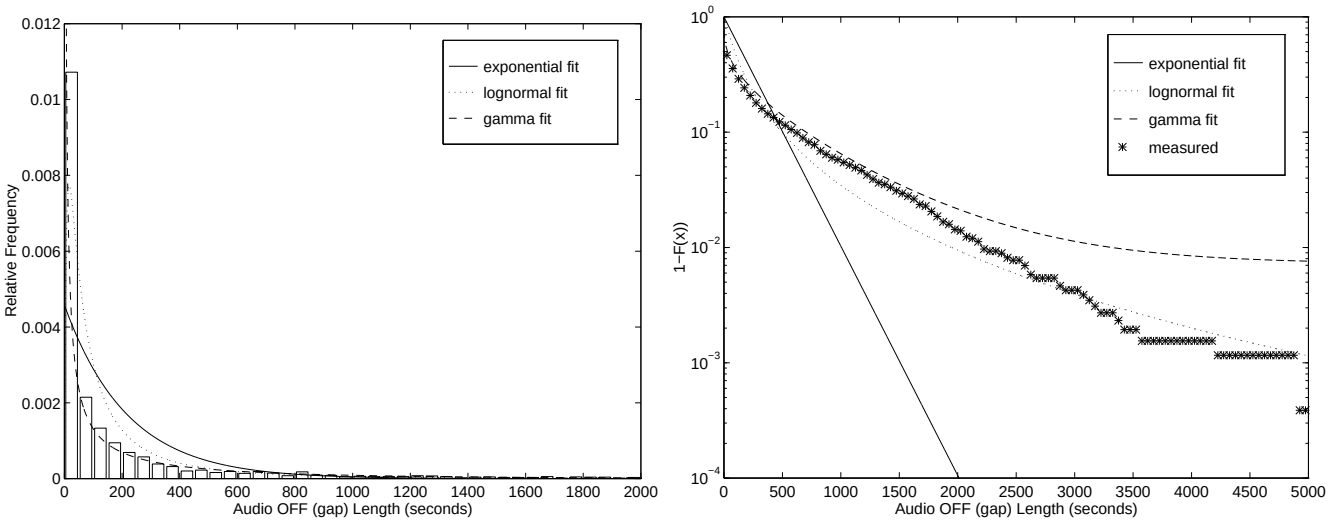


Figure 7: Audio utilization in a session



(a) (b)

Figure 8: Audio ON lengths: empirical measurements and fit data



(a) (b)

Figure 9: Audio OFF gap lengths: empirical measurements and fit data

at 219.9 seconds [5] and slide 12, highlight 1 begins at 2795.9 seconds. Thus we consider the length of jump to be the difference between these two values, i.e. 2576 seconds. Note that we ignore the fact that the user has played out part of slide 2, highlight 2 before making the jump. Thus, we measure the length of jumps only at one highlight granularity. This scheme of measurement reflects the fact the MANIC user interface provides fast forward/rewind capabilities only at one highlight granularity – i.e. using MANIC controls, a user cannot jump into the middle of a highlight. This results in slight overestimation of jump lengths. However, the error is no worse than length of the audio segment associated with one highlight.

We are interested in the length of these jumps for two reasons. First, if backward jumps are short, it might be possible to begin playout after a short backward jump without reloading new audio data from the server; playout could begin immediately using the already downloaded (and previously played-out and stored) audio data. Second, if forward jumps are short, playout can also begin immediately if the audio data has already been preloaded into the buffer as a result of server workahead [16, 13, 5].

In Figure 10, we plot the distribution of length of forward and backward jumps. Interestingly, there are relatively few backward jumps. We had expected the students to make frequent backward jumps to consult previously viewed material. (This unexpected result may be due to the fact that many students might have been using MANIC for review purposes, rather than first-time viewing of the material.) The average length of a jump is 2137.62 seconds in case of forward jumps and 3394.84 seconds in case of backward jumps. Approximately 34.37% of the forward jumps and 33.33% of the backwards jumps were less than 3 minutes in length. Quite interestingly, Figure 10 indicates only a limited amount of "locality" in the jumps.

[5] Note that the value 219.9 (not shown in line 6 in Figure 3) is not at all related to the wall time of 16:04:34. This latter time is simply the time of day the user happened to listen to this particular audio segment.

5 Consequences for CM Architectures and Modeling

Given the empirical and analytic results of the previous sections, we now consider the implications of these results for continuous media architecture and modeling.

Perhaps the most striking observed behavior is the high degree of user interactivity. Recall, for example, that once audio playout begins, in 45% of the cases it lasts less than 3 minutes before the user pauses or jumps elsewhere in the audio stream. This observed user behavior differs significantly from that of a "couch potato" user who initiates a CM playout that continues uninterrupted for a long period of time. From a resource allocation standpoint, a high level of interactivity would argue against prefetching significant amounts of CM material before its playout. While some amount of workahead is needed to remove network jitter [15] and reduce burstiness in the CM stream [16], downloading CM data too far in advance of its playout time can waste network and server resources by downloading data that is never actu-

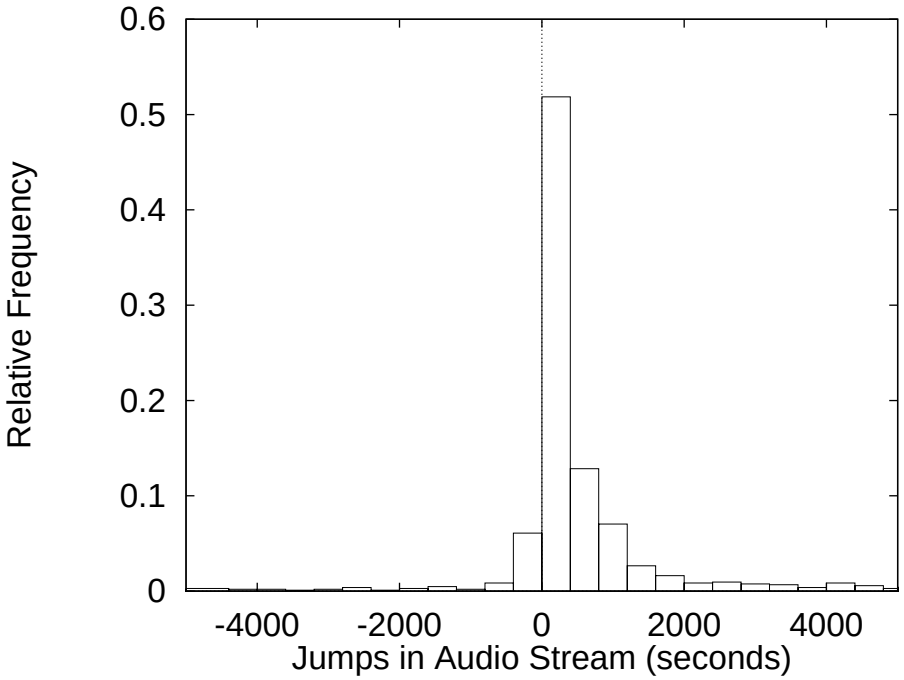


Figure 10: Distribution of lengths of jumps in the audio

ally played. From a network protocol standpoint, this would also argue against a rate control protocol (e.g., TCP) that simply downloads (prefetches) data as fast as the network will allow.

Two surprising observations in our data were the relatively small number of backward jumps ("rewinds") and the relatively large distance covered by jumps in the CM stream. The relatively small number of backward jumps suggests that relatively few interactive requests (fast forward/rewind/index) could possibly be satisfied by replaying previously played (but still buffered) CM. The latter observation suggests that it might also be difficult to satisfy a forward jump using prefetched (but not yet played out) CM. For example, recall that only a third of the observed forward jumps were less than three minutes forward. Even with a low bandwidth CM application such as 14.4K audio, three minutes of prefetched data is often non-trivial amount of data (more than a Megabit of data in this example), particularly for low bandwidth connections. Clearly, the advantage of potentially masking the latency of a forward jump by prefetching data must be weighed against the cost of prefetching data that is never played. This is an interesting question for future research. We also note that while the population as whole exhibits relatively little locality in their jumps, certain individuals may exhibit high locality in their jumping. In such cases, a model-based prefetching strategy that considers past user-specific actions [19] or makes content-specific prefetching decisions [11] may be able to successfully hide jump latency by buffering old data and/or prefetching future CM data. This too remains an issue for future research.

We noted that almost half of the MANIC sessions do not use audio and among those sessions that use audio, it is used only approximately two thirds of the time. Several observations can be made here. First, a CM server that allocates resources on the basis of worst case demand (e.g., assumes that each session will always be playing audio) will clearly over-reserve resources. A more reasonable basis for resource allocation would be the so called on/off model [3, 18] in which a user moves between "on" periods (where audio is streaming to the user) and "off" periods (where no audio is being streamed). A call admission policy based on such a model could then take advantage of the statistical multiplexing gains inherent in any population of users that are not always "on." The empirical data presented in this paper provides parameters for such models. The fact that half of the sessions use no audio at all also makes a compelling argument for handling the components of the CM stream (text, audio, video) as separate elements, allowing the heterogeneous client end systems to request and process/synchronize these streams according to their own desires.

From a modeling standpoint, our empirical data can be used in future simulation and analytic studies of CM systems at the session level, as well as at the detailed level of the activity within a session. We observed that the exponential distribution was a poorer fit to the empirical data than either the lognormal or gamma distributions. We also noted

that the performance measures we have analytically characterized (the length of a session, the length of time spent in the audio "on" and "off" states) have a heavier tail than that of the exponential distribution.

6 Conclusions

In this paper we have provided both empirical and analytic characterizations of observed user behavior with MANIC (Multimedia Asynchronous Networked Individualized Courseware), a WWW-based multimedia client/server application, that streams synchronized CM and HTML documents to remote users. Our goal was to characterize how people actually *use* such a CM system. Understanding how users interact with such a system – i.e., developing a user workload characterization – is crucial in designing and evaluating efficient CM resource allocation and access mechanisms. We examined both session-level characteristics (e.g., the length of the individual sessions) as well as *interactive* user behavior (e.g., the time between starting/stopping/pausing the audio within a session; and the "distance" of rewinding, fast forwarding and indexed jumping within the audio stream). We also considered the consequences of our measurements for CM architectures and their models.

A number of avenues for future work remain. We are currently investigating network-level congestion control protocols for CM, and the effect of client interactivity on the amount of CM data that should be transmitted in advance to the client. We are also currently integrating video into the MANIC system. More rigorous goodness of fits studies [2] can be performed to quantify the benefits of one analytic distribution over another. Finally, even though one distribution may more accurately model the workload than another, the overall effect of using different workload models in various performance studies is another topic for future research.

Acknowledgments

The MANIC system is the collective effort of many people, including the authors. Hu Im Lee, Jesse Steinberg, Mia Stern, and Gary Wallace have been instrumental in the design and implementation of MANIC; without their work these measurements would not have been possible.

References

[1] Anderson D. P., Osawa Y., Govindan R., "A File system for continuous media," *ACM Transactions on Computer Systems 10,* 4 (Nov. 1992), 311-337

[2] R. D'Agostino, Goodness of Fit Techniques, Marcel Decker, 1980.

[3] J. Dey Sircar, J Salehi, J.F. Kurose, D. Towsley, "Providing VCR-like Capabilities in Large-Scale Video Servers", in *Proc. ACM Multimedia 1994,* Oct 1994.

[4] J. Dey, S. Sen, J. Kurose, D. Towsley, J. Salehi, "Playback Restart in Interactive Streaming Video Applications," *Proc. IEEE Conference on Multimedia Computer and Systems," (June 1997 Ottawa Canada).*

[5] W. Feng, F. Jahanian, S. Secrest, "An Optimal Bandwidth Allocation Strategy for the Delivery of Compressed, Prerecorded Video," *Multimedia Systems,* Sept. 1997, pp. 297-309.

[6] J. Gemmel, H. Vin, D. Kandlur, P.V. Rangan, "Storage Servers: A Tutorial and Survey", *IEEE Computer,* May 1995.

[7] M. Grossglauser, J. Bolot,"On the Relevance of Long Range Dependance in Network Traffic," *Proc. ACM SIGCOMM 96.*

[8] J.F. Kurose, H.I. Lee, J. Padhye, J. Steinberg, M. Stern, 'MANIC: Multimedia Asynchronous Networked Individualized Courseware," Technical Report 96-72, Department of Computer Science, University of Massachusetts, Amherst MA 01003. Oct. 1996. *The MANIC homepage is:* `http://gaia.cs.umass.edu/nmis/manic.html`

[9] T. Kwon, Y. Choi, S. Lee, "Disk Placement for Arbitrary Rate Playback in an Interactive Video Server," *Multimedia Systems,* July 1997, pp. 271-281.

[10] K. Nahrstedt, R. Steinmetz, "Resource Management in Networked Multimedia Operating Systems," *IEEE Computer,* May 1995, pp 40–49.

[11] V. Padmanabhan and J. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency", *Proceedings of ACM SIGComm*, 1996, pp. 22-36.

[12] V. Paxson and S. Floyd, "Wide Area Traffic: the Failure of Poisson Modeling,"*IEEE/ACM Transactions on Networking,* Vol. 3, No. 3, (June 1995), pp. 226 - 244.

[13] M. Reisslein and K. Ross, "Join the Shortest Queue Prefetching for VBR Encoded Video on Demand," *Proc.19997 Int. Conference on Networking Protocols,* (Oct. 1997, Atlanta GA).

[14] Real Audio homepage: http://www.realaudio.com.

[15] K. Rothermel, T. Helbig, "An Adaptive Protocol for synchronizing Multimedia Systems," *Multimedia Systems,* Sept. 1997, pp. 324-336.

[16] J. Salehi, Z. Zhang, J. Kurose, D. Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-end Resource Requirements through Optimal Smoothing," *Proc. 1996 ACM Sigmetrics Conference,* (May 1996, Philadelphia PA).

[17] P. Shenoy, P. Goyal, H. Vin, "Issues in Multimedia Server Design," *ACM Computing Surveys,* December 1995.

[18] P. Shenoy, H. Vin, "Efficient Support for Scan Operations in Video Servers," *Proc. ACM Multimedia 95,* (Nov. 5-9, San Francisco).

[19] Mia K. Stern, Beverly Park Woolf, and James F. Kurose, "Intelligence on the Web?", *Proceedings of the 8th World Conference of the AIED Society,* August 18-22, 1997, Kobe, Japan.

Appendix

Factors affecting measurement accuracy

Our measurement accuracy is affected by several factors. In this appendix, we describe these factors and discuss their impact on our work.

- Use of Netscape and RealAudio plugin controls

  Recall that as discussed in Section 2, MANIC provides its own controls to both navigate through the class notes and control the audio. The CGI scripts that maintain the MANIC logs were based on a (naive, in hindsight) assumption that users would refrain from using browser and Real Audio plugin controls to move through the course materials[6] since this functionality is explicitly provided by MANIC. There were, however, users that used the browser/plugin controls (e.g,. forward and backward buttons on the browser, bookmark lists, history index, "pause" buttons on the Real Audio plugin etc.) – actions that would not appear in the MANIC logs but (for audio-related actions) would appear in the Real Audio logs. In order to account for this in the audio ON and OFF length results presented in section 4, we explicitly correlated the MANIC logs with the separate logs maintained by the RealAudio server to account for such actions.

- Measurement of forward and backward jumps

  As noted above, users could potentially make small forward and/or backward jumps in the audio stream using the controls on the RealAudio plugin. The RealAudio logs do not contain sufficient information to determine the lengths of these jumps. It is also difficult to determine lengths of such jumps from the logs maintained by our CGI scripts. Thus, in our analysis of jump lengths, we have only considered the jumps made using MANIC-provided controls.

- Network and processing delays

  Because of network and processing delays, the log entries for various events are written slightly later than when the events actually occur. For example, a MANIC log entry may indicate that the user requested audio playout to start at time t, while the Real Audio logs indicate that audio data was not sent to the client until time $t + \delta$. Most of these delays (δ) are less than 10-15 seconds.

- Small breaks in the audio session can be hard to attribute

  Such breaks can arise due to several factors, including temporary network congestion and multiple clicks by the user on the "Play" buttons. It is hard to distinguish between the various factors that may cause of such breaks, especially since they are all of the same magnitude as the network delay. For example, if the RealAudio logs show that the playout started, say 10 seconds, after the time indicated by the timestamp of the corresponding MANIC log entry, we can not conclude if this was due to some or all of the following factors: network delay, processing overhead at the RealAudio server, user clicking on "pause" button just after he clicked on "Play" etc. This results in our slightly overestimating the length of audio subsessions. We estimate that the magnitude of the overestimate is less than 4%.

- Use of multiple IP addressed and proxies by students

  If a user used multiple IP addresses simultaneously, or used a proxy to receive RealAudio content, it became difficult to match the MANIC and RealAudio logs. We had to exclude logs of 5 users from our consideration, as their logs contained such discrepancies on an extensive scale.

- Use of CGI scripts instead of java for audio and highlight control

  The version MANIC used for these measurements used CGI scripts instead of java applets (that were not available then for use with the Real Audio Plugin) to control audio and FF/Rewind. This resulted in a presentation that is not "smooth," e.g., a highlight change is accompanied by some "flashing" of the page. It is possible that user FF/Rewind behavior might have been different (e.g., used more extensively) if such operations resulted in smoother playout. Java-based

[6] This problem gave rise to almost all the issues discussed here.

controls, which do result in smoother playout, have been incorporated in versions 1.5 and 2 of MANIC. We are investigating whether these enhancements will affect user behavior in any significant way.

The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme*

Dorgham Sisalem
GMD-Fokus, Berlin
sisalem@fokus.gmd.de

Henning Schulzrinne
Columbia University, New York
schulzrinne@cs.columbia.edu

Abstract

Many distributed multimedia applications have the ability to adapt to fluctuations in the network conditions. By adjusting temporal and spatial quality to available bandwidth, or manipulating the playout time of continuous media in response to variations in delay, multimedia flows can keep an acceptable QoS level at the end systems. In this paper, we present a new scheme called the loss-delay based adjustment algorithm (LDA) for adapting the transmission rate of multimedia applications to the congestion level of the network. The LDA algorithm was designed to reduce losses and improve utilization in a TCP-friendly way that avoids starving competing TCP connections. It relies on the end-to-end Real Time Transport Protocol (RTP) for feedback information. In addition, we enhanced RTP with functionalities for determining the bottleneck bandwidth of a connection. The bottleneck bandwidth is then used for dynamically determining the adaptation parameters. Simulations and measurements of the LDA algorithm suggest the efficiency of the scheme in utilizing network resources, decreasing loss ratios and its fairness towards competing TCP traffic. Our investigations also suggest that the report intervals and feedback information of RTP require some enhancements to accommodate the needs of adaptive QoS control schemes.

1 Introduction

A large part of the multimedia conferencing applications used currently in the Internet, such as VIC [17] and VAT [14] are based on the UDP transport protocol. However, UDP offers no quality of service control mechanisms and can therefore not guarantee any level of QoS. Fluctuations of network conditions combined with the inability of those protocols to support QoS control often render multimedia applications useless.

In addition, deploying non-congestion-controlled UDP traffic results in extreme unfairness towards competing TCP traffic. Sending best-effort traffic without any consideration of the network congestion state can easily result in packet losses. In response to that, TCP connections sharing the same bottleneck would reduce their transmission rates. However, without any rate reduction on behalf of the non-congestion-controlled traffic, the TCP connections which constitute around 95% of the Internet traffic today [25] would starve and receive much less than their fair bandwidth shares. Therefore, UDP sources need to be enhanced with quality of service control mechanisms that not only aim at reducing loss ratios and improve bandwidth utilization but also are fair towards competing TCP connections, i.e, be *TCP-friendly*.

Currently, different approaches are being discussed for solving the QoS and congestion control problems such as: resource reservation [4], priority mechanisms [1] and application control [7], i.e., to instruct the applications at the end systems to adapt the bandwidth share they are utilizing to the network congestion state.

Reserving enough resources for supporting a certain QoS level in advance guarantees this quality level and would be the most straightforward approach for handling the problem. However, as it is usually impossible to know the exact characteristics of a stream in advance, one would tend to over-allocate resources to guarantee the requested QoS level, leading to network underutilization. In addition to the complexity and scalability problems of reservation based QoS control schemes, these schemes do not easily allow the use of extra bandwidth for improved quality during network underload states for example.

With priority mechanisms, different data packets or streams are labeled with different priorities and thereby treated differently at the network routers. Such an approach is simpler than the reservation approach as it requires no signaling and less complicated control mechanisms at the routers. However, the exact mechanisms for setting the priority levels, the router mechanisms for controlling these levels and the actual gain achieved with such an approach are

*This work was funded in part by the BMBF (German Ministry of Education and Rese arch) and the DFN (German Research Network).

still under discussion [1].

In this paper, we propose to use QoS control mechanisms based on adapting the sender transmission rate in accordance with the network congestion state. That is, based on feedback information from the receivers, the sender would increase its transmission rate during underload situations and reduce it otherwise. This is especially beneficial when the bandwidth availability may change during a session, particularly during long-lived sessions typical for multimedia communications. Note, that with such an approach no QoS guarantees can be made, but the user perceived quality is improved through loss reduction.

Ergonomic studies and the experience gained from the Internet demonstrate that people can use audio and video data as long as the information content is above a minimum level [28]. This level depends on media content and the task at hand. For instance, a foreign language is more difficult to understand than a native language when audio quality is reduced. So, at the expense of slight degradation in user satisfaction, it is possible to adjust the transmission rates of end systems in accordance with the network congestion state. Hence, deploying application control results in a better overall bandwidth utilization, as it avoids network congestion. A major advantage of application control schemes is that they require no support from intermediate routers and are, thus, easy to deploy.

Applications control works best for live transmission for unicast and small multicast groups; in large multicast groups in a heterogeneous environment, a "race to the bottom" can occur so that one poorly connected receiver determines the quality for the much larger number of well-connected receivers. To avoid these problems, various proposals have been made for hierarchical data distribution [27, 18, 29]. Those proposals are based on partitioning a data stream into a base layer, comprising the information needed to achieve the lowest quality representation and a number of enhancement layers. The different layers are then sent to different multicast sessions and the receivers determine how many sessions to join and thereby adjust their QoS in respect to their own requirements and capacities.

While layered data transmission solves the heterogeneity problems, it might cause additional delays at the receivers. As the different layers might use different paths and hence have different round trip times, the receivers need to resynchronize the arriving data. Additionally, data partitioning might lead to a drift problem caused by motion compensated coding if only the lower bit rate layer is received [11]. Finally, it increases the complexity of the end systems considerably.

In this paper, we propose a new end-to-end rate adaptation scheme called the loss-delay based adjustment algorithm (LDA) for adjusting the transmission rate of multimedia applications to the congestion level of the network. The LDA algorithm resembles other adaptation approaches proposed in the literature [3, 5] that increase the transmission rate during network underload periods by an additive increase rate (AIR) and reduce the rate multiplicatively during congestion periods. However, these schemes show a rather aggressive adaptation behavior, often leading to the starvation of competing TCP traffic [23]. The LDA algorithm is on the contrary designed in a TCP similar fashion that prevents the starvation of TCP connections but sill allows for a stable transmission behavior.

Another issue that none of the adaptation schemes we encountered addresses is how to choose AIR. The appropriate value of AIR depends on the bandwidth share of a connection. As this share might change dynamically during the lifetime of a connection using a constant AIR value is rather inappropriate. The LDA algorithm adjusts its adaptation parameters dynamically in response to the losses, delays and capacity observed on the path traversed by the adaptive connection. The capacity of the network is estimated by enhancing RTP with the packet pair approach presented in [13].

In Sec. 2 of this paper we present some related work on the issue of TCP-friendly adaptation. The loss-delay based adjustment algorithm (LDA) is then presented in Sec. 3. Finally, the performance of the algorithm in terms of bandwidth utilization, scalability and fairness towards competing TCP connection is investigated using simulations and measurements in Sec. 4.

2 TCP-Friendly Adaptation Algorithms

As deploying non-congestion controlled UDP traffic in the Internet might result in starving competing TCP connections [10], different approaches have been suggested that aim at adjusting the transmission behavior of UDP senders in a TCP similar fashion. In the following we present a brief summary of some of those approaches:

- Jacobs [12] presents a scheme that uses the congestion control mechanisms of TCP, however, without retransmitting lost packets. In his scheme, the sender maintains a transmission window that is advanced based on the acknowledgments of the receiver. Based on the size of the window the sender can estimate the appropriate transmission rate. Thereby, the adaptive connection is guaranteed to acquire the same bandwidth a TCP connection would be getting under the same conditions of losses and delays. However, the need to acknowledge each packet limits the benefits of such an approach to unicast communication. Also, the acknowledgment packets increase the overall network traffic significantly diminishing the benefit of adaptation, particularly for short voice packets.

- Floyd et al. [10] and Ott et al. [19] propose a model that estimates the throughput of a TCP connection (r_{TCP}) under known delay and loss conditions as

$$r_{\text{TCP}} = \frac{1.22 \times M}{\text{RTT} \times \sqrt{l}} \quad (1)$$

with M as the maximum packet length, RTT as the round trip delay of the connection and l as the average loss measured during the lifetime of the connection.

Based on this estimation, Mahdavi et al. [15] and Turletti et al. [26] propose end-to-end, TCP-friendly congestion control schemes in which the end systems measure losses and delays in the network and restrict their transmission rates to the value estimated by Eq. 1. Floyd et al. [10] describe a mechanism in which routers identify connections that use more bandwidth than allowed by Eq. 1 and then throttle these connections.

The TCP-throughput model is based on a TCP sender interpreting packet drops as an indication of congestion and responding by reducing the congestion window and thereby the effective sending rate by half. Further, this sender should only increase its transmission window by at most one packet each round trip time.

While this model is rather simple, it is only partially correct. It does not consider timeout cases or delayed acknowledgments. However, even under those restrictions different studies [10, 16] have shown this model to be accurate enough for losses of less than 16%.

Note, however, that the TCP-throughput model is based on the *average* loss and delay observed during the lifetime of a connection. However, adaptation decisions need to be taken based on the current loss and delay values. As the observed losses and round trip delays change dynamically, using the TCP-throughput model as the sole basis for adaptation results in a rather oscillatory adaptation behavior which might lead to an annoying perceived QoS by the users due to the rapid changes in the quality of the received data stream. In addition, the TCP-model does not state the rate increase method during no loss periods. Testing a simple adaptation mechanism based on setting the bandwidth to the rate estimated by the TCP model using the loss and delay values reported in the RTCP packets and increasing the rate additively during no loss periods the scheme showed a very oscillatory behavior and on the average a rather low throughput [23]. Also, the authors in [19] state that if the round trip time is caused by queuing delays, or if the bottleneck router is shared among competing connections, this model is only of limited value.

3 The Loss-Delay Based Adjustment Algorithm (LDA)

The LDA algorithm is a sender based adaptation scheme. It relies on the Real Time Transport Protocol (RTP) [20] for feedback information about the losses at the receivers and round trip time. We additionally enhanced RTP to estimate the bottleneck bandwidth of a connection. Based on these information, the sender calculates for each received feedback message from receiver (i) the appropriate bandwidth to use for the reporting receiver (r_i). During periods without losses the sender increases r_i by an additive increase rate (AIR_i) which is estimated using the loss, delay and bottleneck bandwidth values included in the feedback reports. The actual sending rate is then determined at equally spaced adaptation points as $\min(r_i)$.

We divide the algorithm description into three parts handling the feedback information, upon which the adaptation decisions are taken, the mechanisms for setting the adaptation parameters and the actual adaptation mechanisms.

3.1 Feedback Information

The loss-delay based adjustment algorithm is based upon the Real Time Transport Protocol (RTP) [20] designed within the Internet Engineering Task Force (IETF). RTP enhances other transport protocols such as UDP with features needed by continuous media applications, such as the capability of distinguishing between different media streams and keeping track of various statistics describing the quality of the transmission as seen by other members of the session.

RTP sessions consist of two lower-layer data streams, namely a data stream for, say, audio or video and a stream of control packets (using the sub-protocol called RTCP). Each session member periodically sends RTCP control reports to all other session members. Senders transmit reports describing the amount of data they have sent and a timestamp indicating the time the report was generated. For each incoming stream the receivers send a report indicating the fraction of lost data, the timestamp of the last received sender report (t_{LSR}) for that stream and the time elapsed in between receiving the last sender report and sending the receiver report (t_{DLSR}). Knowing the arrival time (t) of the RTCP packet the sender can calculate the round trip time (RTT) as follows:

$$\text{RTT} = t - t_{\text{DLSR}} - t_{\text{LSR}} \quad (2)$$

This calculation requires no synchronization between the clocks of the sender and receiver and is therefore rather accurate. With the LDA algorithm, we estimate the round trip propagation delay (τ) as the smallest measured RTT.

In addition, we enhanced RTP with the ability to estimate the bottleneck bandwidth of a connection based on the

packet pair approach described by Bolot [2]. The essential idea behind this approach is: If two packets can be caused to travel together such that they are queued as a pair at the bottleneck, with no packets intervening between them, then the inter-packet spacing will be proportional to the time required for the bottleneck router to process the second packet of the pair.

We added to the RTCP packets an application defined part including the source sequence number, the sequence number (SEQ) of a data packet that will start a stream of probe packets and the number (n) of probe packets that will be sent. Then, n data packets starting with packet numbered SEQ are sent at the access speed of the end system. At the receiver site, the bottleneck bandwidth (b) is calculated as:

$$b = \frac{\text{probe packet size}}{\text{gap between 2 probe packets}}$$

Note, that we are using data packets as probe packets and therefore the additional bandwidth required for the bottleneck probing is restricted to the increased size of the RTCP packets. Sending data packets as probe packets is appropriate when transmitting video streams as one can send a few packets belonging to the same video frame together without considerably altering the traffic characteristics. Also video packets tend to be large which increases their possibility of being buffered at the routers. For the case of audio streams with small packets and stringent timing requirements another solution might need to be considered.

To estimate the average bottleneck bandwidth we need to further filter out incorrect estimates. We rely on an approach similar to that used in the BPROBE tool [6], namely clustering similar estimates into intervals, and choosing the average of the interval with the highest number of estimates. The estimated value is then sent back to the sender with the next receiver report.

Obviously, this approach has lots of drawbacks. We do not include the time between the transmission of two probe packets. But, as we send the packets at the sender's network access speed which in our case was 10 Mb/s we can usually ignore this time. Also, we do not consider packet drops or competing traffic. However, testing this approach on different Internet connections we achieved results comparable to those estimated by a more complicated tool such as the PATHCHAR[1] tool by LBL. Still, we need to further refine the estimates filtering method and do more measurements.

3.2 Dynamical Determination of the Additive Increase Rate (AIR)

The appropriate value to use for AIR depends largely on the bandwidth share a connection could utilize. For example, consider the case of a 1 Mb/s link shared between 2 connections. An ideal sharing would be if each connection received 500 kb/s. For this case, measurements [5] have shown that an appropriate AIR value that allows small convergence periods and only small oscillations would be 50 kb/s. This value would, however, be inappropriate if the link was being shared among 100 connections for example. With the LDA algorithm the additive increase rate (AIR) is set dynamically based on the congestion state of the network. That is, AIR is set initially to a small value, and is then increased during periods without losses. If losses are reported by the receivers, the senders set the AIR back to the initial value. We have chosen an initial value of 10 kb/s, which is small enough to be used even in narrowband ISDN connections.

If the received RTCP messages from receiver (i) indicate no losses the sender calculates an AIR_i for this receiver as follows:

$$\begin{aligned} \text{AIR}_i &= \text{AIR} * B_f \\ \text{with } B_f &= 1 - \frac{r}{b} \end{aligned}$$

with AIR as the additive increase value calculated for the entire session, r the current transmission rate, b the bottleneck bandwidth and B_f as a bandwidth factor that allows connections with a low bandwidth share to use larger AIR values and thereby converge faster to their fair bandwidth share. The difference between AIR and AIR_i will be explained in Sec. 3.3.

Further, AIR_i is limited maximally to the average rate increase (r_{incr}) a TCP connection would utilize during periods without losses in a time interval equivalent to the interval between two adaptation points (T). So for a round trip propagation delay of τ and a packet size of M, a TCP connection would increase its transmission window each τ by one packet size. Translated into transmission rate increase, r_{incr} would be

$$r_{incr} = \frac{M \times (\frac{T}{\tau} + 1)}{2}.$$

3.3 Congestion Control Mechanism

With each received RTCP packet from a member i the sender calculates a transmission rate (r_i) it would use if member i was the only member of the multicast session. r_i is then saved into a data base. So, if no losses were indicated the sender can recalculate AIR_i as was described in 3.2 and r_i is then set to

$$r_i = r + \text{AIR}_i$$

with r as the transmission rate the sender is using for the entire session.

[1] A tool for estimating the loss, delay and bandwidth characteristics of an Internet path. The tool is freely available from ftp://ftp.ee.lbl.gov/pathchar/

Otherwise, the sender reduces r_i in a rather TCP similar fashion, i.e., proportional to the indicated losses (l)

$$r_i = r * (1 - (l * R_f))$$

with R_f as the reduction factor. This factor determines the degree of the reaction of the sender to losses. Higher values result in a faster reduction of the transmission rate but a more oscillatory behavior. Lower values, on the other hand, lead to more stable rate values but result in longer convergence periods. The different simulations we ran suggest that we get the best tradeoff between convergence time and stability for a reduction factor between 2 and 5. Throughout this study we use a factor of 3.

Additionally, to favor connections with a low bandwidth share, we define a lower loss threshold (l_{th}) below which a connection with lower transmission rate than an *equivalent TCP connection* can increase its transmission rate. Equivalent TCP connection indicates here a TCP connection with the same packet size, round trip delay as can be calculated using the timing information of the RTCP packets and having the same losses. In all of our simulations we set l_{th} to 1%.

Instead of reacting to each RTCP packet as some adaptation schemes suggest [5], the LDA algorithm is based on so called adaptation points. At each adaptation point, the sender searches the transmission rates data base for the minimum value (r_{min}) and sets r to r_{min}. A r_{min} smaller than the current transmission rate (r) indicates that at least one member has reported losses. In this case, AIR is reinitialized to the initial value of 10 kb/s. If r_{min} was higher than the current transmission rate (r), then AIR is set to AIR$_i$ calculated for the member for which r_{min} was determined for.

We have used a period of 5 seconds between two adaptation points which is also the average value between generating two RTCP packets at the same source. As the time between two RTCP packets might actually be larger than 5 seconds, choosing this fixed value has the disadvantage that the adaptation decision might be taken based on the reports of only a subset of the session members and not all of them. However, the other alternative would be to wait for the reports from all members before adapting. While this might work for small groups, for larger groups the interval between two sent RTCP packets increases and thereby the adaptation decision will be taken less frequently and, thus, be less effective.

4 Performance of the Loss-Delay Based Adjustment Algorithm

When designing an adaptive control scheme, following goals need to be considered:

- the scheme should result in a low packet loss ratio,
- achieve high overall bandwidth utilization,
- fairly distribute bandwidth between competing connections,
- and scale for large multicast groups.

In this section, we investigate the performance of the LDA algorithm with regard to those different requirements using both simulations as well as measurements on a real network.

4.1 Fairness and Convergence of the LDA Algorithm

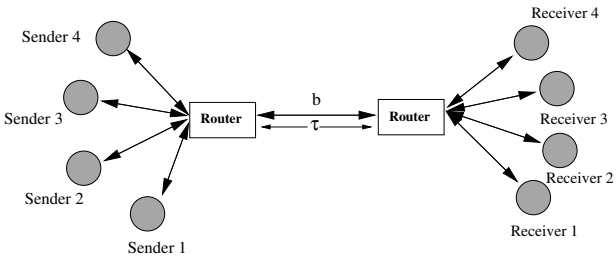
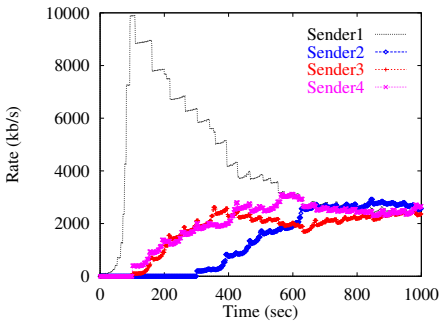
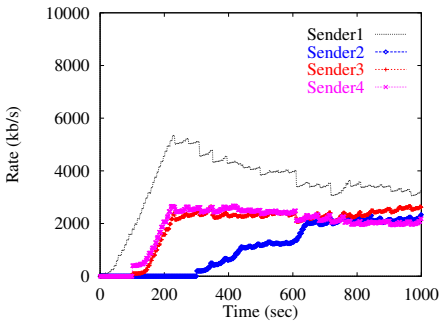


Figure 1. LDA performance testing topology

As a first performance test of the LDA algorithm, we simulated the topology depicted in Fig. 1. The model consists of 4 connections sharing a bottleneck router. All connections deploy the LDA algorithm and are persistent sources. That is, they always have data to send with the maximum allowed rate. They all have the same round trip propagation delay (τ) and are similar in their requirements and characteristics. The router is a random early drop (RED) gateway as was proposed by Floyd and Jacobson [9]. A RED gateway detects incipient congestion by computing the average queue size. When the average queue size exceeds a preset minimum threshold the router drops each incoming packet with some probability. Exceeding a second maximum threshold leads to dropping all arriving packets. This approach not only keeps the average queue length low but ensures fairness and avoids synchronization effects. Based on results achieved in [10] the minimum drop threshold was set to 0.5 of the routers buffer and the maximum one to 0.95 of the routers buffer. In our simulations we set the maximum queuing delay to 0.1 seconds and the bandwidth (b) of the bottleneck router to 1 and 10 Mb/s. The packet size is set to 1 kbyte, which is a typical video packet size. In the first set of simulations, we looked at the utilization, losses and fairness of the LDA algorithm with all connections having the same round trip time and different starting times. The bandwidth distribution and loss figures shown in Fig. 2 and 4 reveal that after a convergence period the four connections received equal bandwidth shares with average losses between

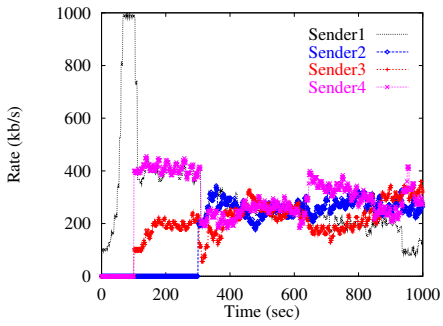


(a) τ = 1 ms, b=10 Mb/s

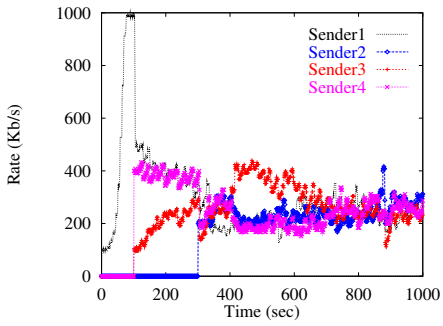


(b) τ = 100 ms, b=10 Mb/s

Figure 2. Rate of competing LDA connections with round trip propagation delay time (τ) and link bandwidth (b)



(a) τ = 1 ms, b=1 Mb/s



(b) τ = 100 ms, b=1 Mb/s

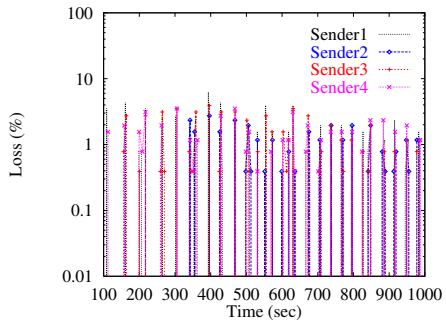
Figure 3. Rate of competing LDA connections with round trip propagation delay (τ) and link bandwidth (b)

2% and 10%. The utilization was around 95% in all measurements. Note that the convergence period for the simulations with bottleneck rate of 10 Mb/s, see Figs. 2(a) and 2(b), might extend over more than 300 seconds. This, however, depends on the value of the chosen reduction factor (R_f). We have used an R_f of 3. We could have achieved shorter convergence periods using higher values of R_f. This would, however, have lead to a more oscillatory adaptation behavior which is inappropriate for adaptive video as it would result in a lower perceived quality due to the rapidly changing video quality.

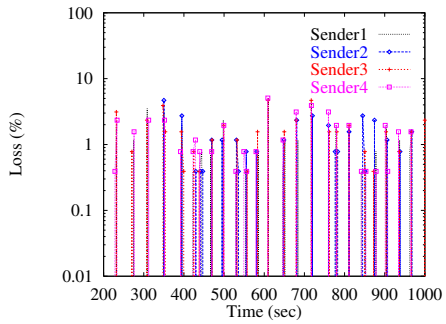
4.2 Interaction of TCP and LDA Traffic

The simulations run in Sec. 4.1 showed that the LDA algorithm achieves a rather fair bandwidth distribution among similar connections. In this section, we investigate the fairness of the LDA algorithm towards other adaptive traffic such as TCP. As a simulation topology, we use the topology depicted in Fig. 1 but with one of the senders replaced by a TCP sender. We investigate the bandwidth distribution achieved for the case of a bottleneck rate of 10 Mb/s and different round trip times.

The bandwidth distribution results depicted in Fig. 6 show that for the case of (τ = 1ms) the TCP connection receives the same bandwidth share as the LDA connections. For (τ = 100ms) the TCP connection receives nearly 1.4 Mb/s which is around half of its fair share which we would expect in this case to be 2.5 Mb/s. Based on Eq. 1 for a TCP connection with a packet size of 1 kbyte, a round trip time of 0.1 seconds to reach a bandwidth share of 2.5 Mb/s we would need to maintain an average loss rate of less than 0.16%. Actually, as we should also consider the queuing delay the loss value is even smaller. However, the RTCP packets only include a field of 8 bits for the loss value. That is, the minimum loss value that can be reported is around 0.4%. With such a loss value a TCP connection would utilize for the delay and packet size at hand a bandwidth share of around 1.6 Mb/s which is only slightly more than the value depicted in Fig. 6. To achieve optimal fair

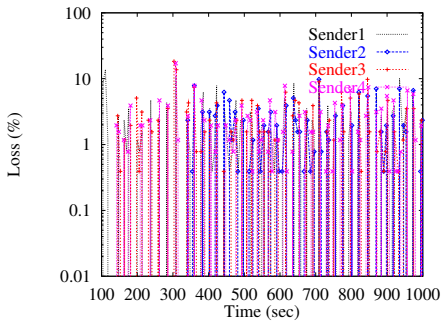


(a) τ = 1 ms, b=10 Mb/s

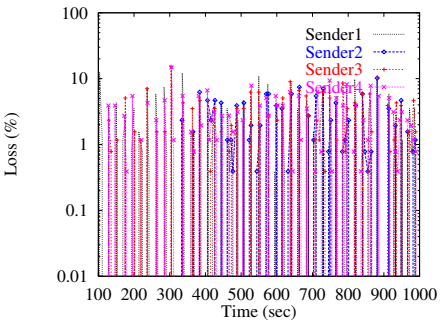


(b) τ = 100 ms, b=10 Mb/s

Figure 4. Loss of competing LDA connections with round trip propagation delay (τ) and link bandwidth (b)



(a) τ = 1 ms, b=1 Mb/s



(b) τ = 100 ms, b=1 Mb/s

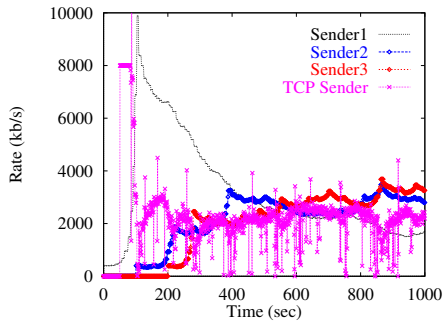
Figure 5. Loss of competing LDA connections with round trip propagation delay (τ) and link bandwidth (b)

bandwidth distribution with the TCP connection getting exactly the same share as the LDA connections we would need to enhance RTCP to carry finer granulated loss values. Another approach would be to use the value of the cumulative number of lost packets (l_{cum}) included in the RTCP packets instead of the loss fraction. This value indicates the number of packets lost since the beginning of reception. While this leads to more accurate loss indications it also increases the complexity of the scheme as the sender needs to maintain more state information to actually benefit from l_{cum}.

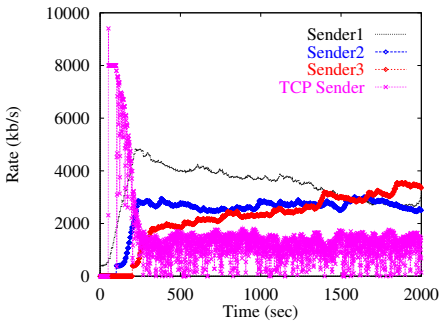
4.3 Scalability of the LDA Algorithm

To investigate the behavior of the LDA algorithm when used in multicast groups of different sizes we tested a simple case of a sender transmitting data to n receivers with n set to 1, 4, 64 and 320. As Fig. 7(a) shows the bottleneck router is shared between the sender and an on-off connection introducing bursty back ground traffic, see Fig. 7(b).

From Fig. 7(c) we can notice, that with a session of 64 or 320 members the reactions to the sudden loss peaks are larger than those for smaller groups. The reason for this is related to the RTCP scaling algorithm. To avoid the increase in control traffic with the increase in group size the RTCP traffic is restricted to 5% of the bandwidth utilized by the session. With 320 members in the session the interval between two successive RTCP messages is larger than the adaptation interval of 5 seconds that we are using. As the reported losses in an RTCP message is calculated over the entire time between the sending of two RTCP packets the effects of a loss peak will be noticed over several adaptation points even though the congestion has actually cleared. As an example, consider the case depicted in Fig. 8. Between the first two adaptation points we measure a loss rate of 10% over a time period of say 2 seconds. Assume, that the RTCP intervals of two members are 10 seconds long and are overlapping with one RTCP message arriving before the second adaptation point and the RTCP message of the other member arriving only before the third adaptation point. As the loss is averaged over the entire 10 seconds the first mem-

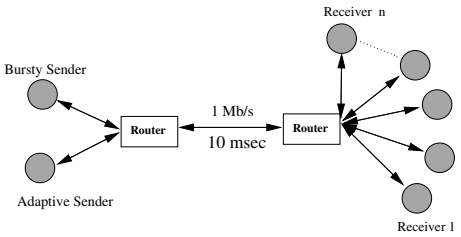


(a) τ = 1 ms, b=10 Mb/s

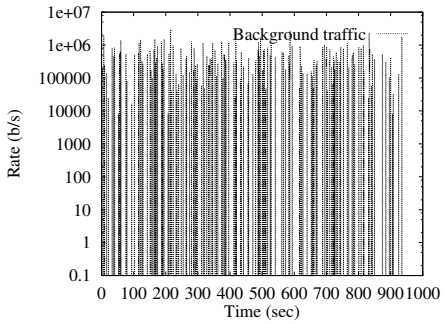


(b) τ = 100 ms, b=10 Mb/s

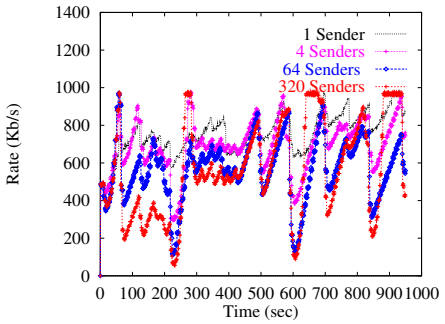
Figure 6. Bandwidth of LDA connections competing with a TCP connection with round trip propagation delay (τ) and link bandwidth (b)



(a) Multicast test topology for LDA



(b) Background traffic



(c) Sender transmission rate

Figure 7. Testing the scalability of the LDA algorithm

ber would report a loss of 2%. Thereupon, the sender would decrease its transmission rate and we would get a loss rate of 0% again. However, the second member will still report a loss of 2% even though no losses can be measured anymore. One approach that avoids this problem would be for the receivers to inform the senders about the losses seen in the last 5 seconds or so and not the average loss measured between the sending of two RTCP packets. This problem is being discussed in the IETF for a future version of RTP

4.4 Fairness of the LDA Algorithm

In this section, we investigate the fairness of the LDA algorithm for the case of:

- different connections sharing the same bottleneck but having different round trip times,
- and connections having the same round trip time but traverse a different number of congested links.

As TCP's throughput is inversely proportional to the round trip time we would expect TCP to be unfair under these conditions, that is two similar connections sharing the same bottleneck but having different round trip delays would not receive the same bandwidth share. We use the same topology depicted in Fig. 1 but with all senders starting to send data at the same time and the connections from senders 3 and 4 to the receivers 3 and 4 having a round trip time 10 times as large as the other two connections. The bandwidth distribution results depicted in Fig. 9 suggest that

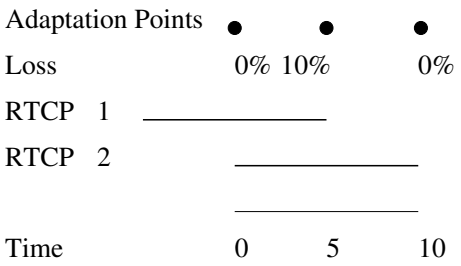
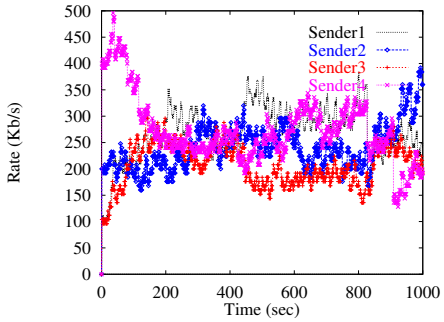
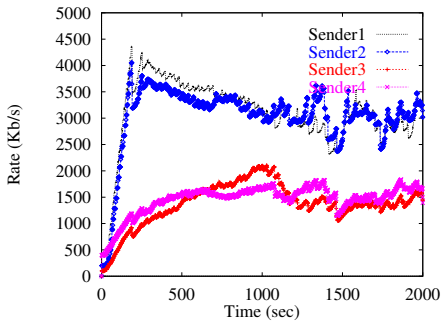


Figure 8. Timeline of adaptation points and RTCP report intervals



(a) τ= 100, 1000 msec, b=1 Mb/s



(b) τ = 100, 1000 msec, b=10 Mb/s

Figure 9. Bandwidth distribution of LDA controlled connections with different round trip times propagation delays(τ) and link bandwidth (b)

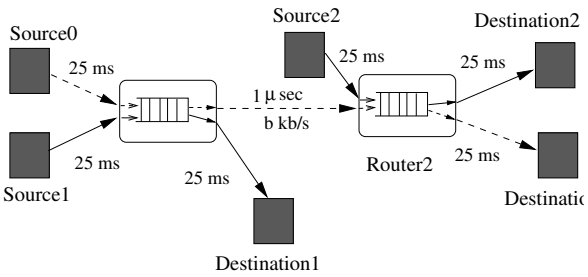


Figure 10. Test configuration for investigating the effects of the beat down problem on the LDA algorithm

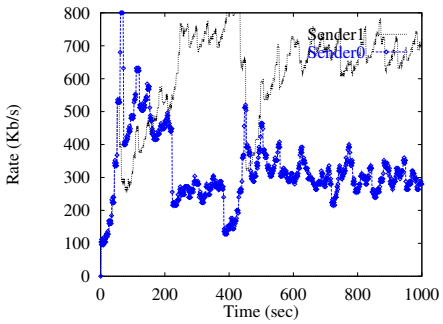
like TCP, the bandwidth distribution is unfair. This is especially evident for the case of a bottleneck bandwidth of 10 Mb/s. Due to the different round trip times the connections calculate different maximum additive increase rates. Additionally, for the case of losses below the lower threshold the transmission rate is set at the maximum to the TCP rate under comparable conditions. Thereby, connections with higher τ have a smaller maximum allowed rate and can thus only benefit less from this low loss situation.

Various studies [8, 22] have shown that the bandwidth of a TCP connection not only depends on the round trip delay but also on the number of traversed hops. This phenomena called the beat down problem arises as a result of the increase in the loss probability with each additional traversed link. In Fig. 10 we investigate the case of two connections having the same round trip delay with one connections traversing only one router and the the other one traversing two routers. Both connections use the LDA algorithm with the same initial parameters (l_{th}=1%, R_f=3 and initial AIR=10 kb/s). The results achieved for the router capacity of 1 Mb/s and 10 Mb/s, see Fig. 11, reveal that the connection traversing only one hop actually receives around twice as much bandwidth compared to the connection traversing two hops. Looking at the losses, see Fig. 12, the two connection suffer from we notice that the connection traversing two routers suffers higher losses and receives thereby a lower bandwidth share.

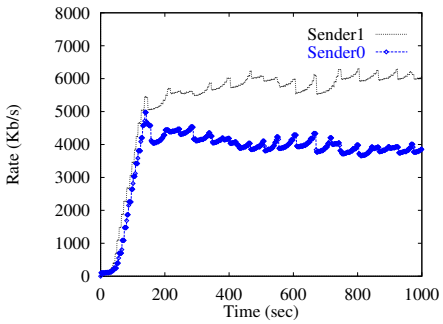
4.5 Measurement Based Testing

To verify the performance of the LDA algorithm we ran a simple experiment over our local network. We connected two workstations over a router with a limited capacity of 1 Mb/s. From the first station we sent two JPEG video streams using the NEVIT video tool [24] which we enhanced with the LDA algorithm. In addition, we started an FTP transfer from the first station to the second in parallel to the video streams.

The first 300 seconds of our measurements resemble to some extent a typical situation we might encounter in the Internet. Both video streams were sent during this period without using the adaptation scheme. This results in the starvation of competing TCP connections and the UDP connec-

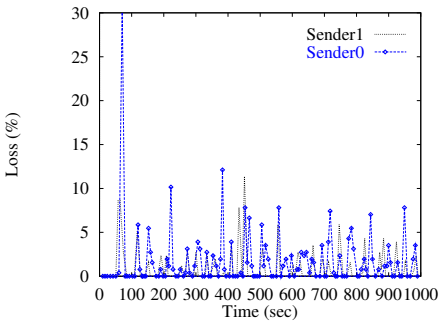


(a) τ= 100 msec, b=1 Mb/s

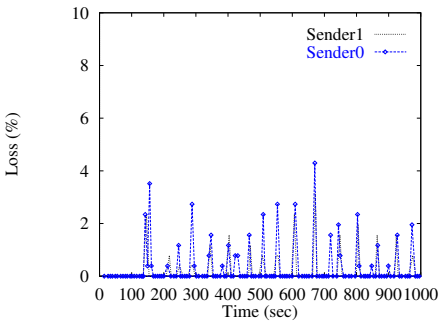


(b) τ = 100 msec, b=10 Mb/s

Figure 11. Bandwidth distribution of LDA controlled connections with different round trip propagation delays (τ) and link bandwidth (b)



(a) τ= 100 msec, b=1 Mb/s



(b) τ = 100 msec, b=10 Mb/s

Figure 12. Losses of LDA controlled connections with round trip propagation delay (τ) and link bandwidth (b) -Sender 0 traverses 2 hops and Sender 1 traverses 1

tions suffering high losses (around 50%). In the next 900 seconds both UDP connections adapt their transmission rate to the loss level. This results in a rather fair bandwidth distribution of around 300 kb/s for both the adaptive connections as well as the TCP connection. Additionally, the losses of the UDP connections drop to around 7%. In the final part of the measurement, one UDP connection stops sending data. We notice that both the other UDP connection and the TCP connection increase their bandwidth share up to around 500 kb/s.

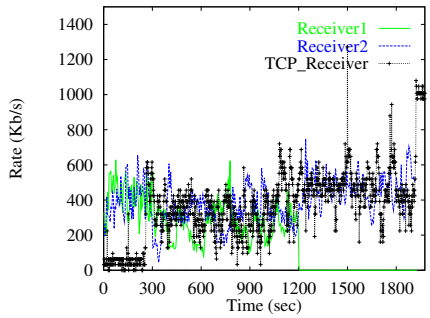
5 Summary and Future Work

In this paper, we presented a new approach (LDA) for dynamically adjusting the sending rate of applications to the congestion level observed in the network. The various simulations and measurements we made suggest the efficiency of the LDA algorithm in utilizing network resources and reducing losses. Also, the scheme is shown to be fair towards TCP traffic and flexible in its estimation of the appropriate adaptation parameters to use.

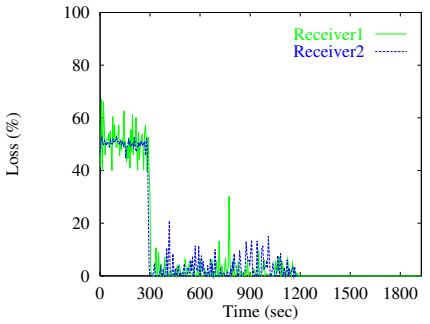
To address the problem of adaptation in heterogeneous environments, we are currently investigating a method for integrating sender based adaptation schemes such as the LDA algorithm with layered transmission methods. With such an approach, the sender determines the number of layers to use and the transmission rate to use for each layer dynamically based on the feedback information sent by the receivers [21].

Additionally, we are currently testing the efficiency of the bottleneck probing method we introduced in the LDA algorithm as well as the acceptability of adaptive video traffic from the users' point of view.

A major point that we did not address in this paper is how to enforce end systems to use adaptive QoS control schemes. In an environment in which most of the users behave in a rather social way and adapt their transmission rates to the congestion state of the network, employing non-adaptive transmission behavior would actually result in a better band-



(a) Bandwidth Distribution



(b) Losses

Figure 13. Measurement of bandwidth distribution and losses for the interaction of TCP and the LDA algorithm

width share for the offending sender.As a solution to this problem we could imagine a scenario in which the end systems deploy the LDA algorithm and the routers identify non-congestion controlled connections [10] and throttle them.

6 Acknowledgments

The RTP/RTCP simulation models were mainly implemented by Timur Friedman and improved by Frank Emanuel. The comments of Adam Wolisz are gratefully acknowledged and were the basis for various aspects of this work.

References

[1] S. Blake. Some issues and applications of packet marking for differentiated services. Internet Draft, Internet Engineering Task Force, Dec. 1997. Work in progress.

[2] J.-C. Bolot. End-to-end packet delay and loss behavior in the Internet. In D. P. Sidhu, editor, *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 289–298, San Francisco, California, Sept. 1993. ACM. also in *Computer Communication Review* 23 (4), Oct. 1992.

[3] J.-C. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the internet. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 58–67, London, England, Aug. 1994. ACM.

[4] R. Braden, L. Zhang, and S. Berson. Resource reservation protocol (RSVP) – version 1 functional specification. Internet Draft, Internet Engineering Task Force, Nov. 1995. Work in progress.

[5] I. Busse, B. Deffner, and H. Schulzrinne. Dynamic QoS control of multimedia applications based on RTP. *Computer Communications*, 19(1):49–58, Jan. 1996.

[6] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical Report BU-CS-96006, Computer Science Departement, Boston University, Mar. 1996.

[7] C. Diot, C. Huitema, and T. Turletti. Multimedia application should be adaptive. In *HPCS*, Mystic, Connecticut, aug 1995.

[8] S. Floyd. Connections with multiple congested gateways in packet-switched networks, part 1–one-way traffic. *ACM Computer Communication Review*, 21(5), Oct. 1991.

[9] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[10] S. Floyd and F. Kevin. Router mechanisms to support end-to-end congestion control. Technical report, Feb. 1997.

[11] U. Horn and B. Girod. Scalable video coding for the internet,. In *8th Joint European Networking Conference*, Edinburgh, England, May 1997.

[12] S. Jacobs and A. Eleftheriadis. Providing video services over networks without quality of service guarantees. In *RTMW'96*, Sophia Antipolis, France, Oct. 1996.

[13] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review*, 18(4):314–329, Aug. 1988. Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[14] I. Kouvelas, V. Hardman, and A. Watson. Lip synchronization for use over the internet: Analysis and implementation. In *GLOBECOM'96*, London, UK, Nov. 1996.

[15] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control, June 1997. Technical note, available from http://ftp.ee.lbl.gov/floyd/papers.html.

[16] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *IEEE Network*, 11(6), November/December 1997.

[17] S. McCanne and V. Jacobson. vic: A flexible framework for packet video. In *Proc. of ACM Multimedia '95*, Nov. 1995.

[18] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *SIGCOMM Symposium on Communications Architectures and Protocols*, Palo Alto, California, Aug. 1996.

[19] T. Ott, J. Kemperman, and M. Mathis. Window size behavior in TCP/IP with constant loss probability. In *The Fourth IEEE Workshop on the Architecture and Implementation of Hi gh Performance Communication Systems (HPCS'97)*, Chalkidiki, Greece, June 1997.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. Technical Report RFC 1889, Internet Engineering Task Force, Jan. 1996.

[21] D. Sisalem and F. Emanuel. QoS control using adaptive layered data transmission. In *IEEE Multimedia Systems Conference'98*, Austin, TX, USA, June 1998.

[22] D. Sisalem and H. Schulzrinne. Congestion control in TCP: Performance of binary congestion notification enhanced TCP compared to Reno and Tahoe TCP. In *International Conference on Network Protocols (ICNP)*, pages 268–275, Columbus, Ohio, Oct. 1996.

[23] D. Sisalem, H. Schulzrinne, and F. Emanuel. The direct adjustment algorithm: A TCP-friendly adaptation scheme. Technical report, GMD-FOKUS, Aug. 1997. Available from http://www.fokus.gmd.de/usr/sisalem.

[24] D. Sisalem, H. Schulzrinne, and C. Sieckmeyer. The network video terminal. In *HPDC Focus Workshop on Multimedia and Collaborative Environments (Fifth IEEE International Symposium on High Performance Distributed Computing)*, Syracuse, New York, Aug. 1996. IEEE Computer Society.

[25] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6):–, November/December 1997.

[26] T. Turletti, S. F. Prisis, and J.-C. Bolot. Experiments with a layered transmission scheme over the Internet. Rapport de recherche 3296, INRIA, Nov. 1997.

[27] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *INFOCOM'98*, San Francisco, USA, Mar. 1998.

[28] F. Wilson, I. Wakeman, and W. Smith. Quality of service parameters for commercial application of video telephony. In *Human Factors in Telecommunication Symposium*, Darmstadt, Germany, Mar. 1993.

[29] L. Wu, R. Sharma, and B. Smith. Thin streams: An architecture for multicating layered video. In *The 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 97)*, St. Louis, USA, May 1997.

A Toolkit for Building Migratory Continuous Media Applications

Tatsuo Nakajima and Hiroyuki Aizu
{tatsuo, aizu}@jaist.ac.jp
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292, JAPAN

Abstract

In this paper, we propose a toolkit for building environment-aware migratory continuous media applications. In future computing environments, various objects in homes and offices will embed computers, and various applications will be moved between these computers according to the location of these users. However, since the computers may have dramatically different hardware and software configurations, the application cannot be moved without taking into account the configuration of the computer that the application is migrated. Therefore, migratory applications should be aware of environments that they are executed.

1 Introduction

In future computing environments, users will expect to use their computing environments in any places even if they go out of their offices. In this case, it is convenient that their applications will be moved with them according to their location. For example, their intelligent agents that monitor their behaviors should give advice to them whenever they need helps. The users expect the agents to be executed in computers that exist near them. Traditionally, the scenario can be realized by carrying computers of respective users, and the agents can be executed on their computers.

The approach requires that the computers may execute heavy computation, and always connect to networks for acquiring information from servers. This means that users should carry powerful but heavy computers with them. The alternative approach is that the agents run on the computers near the agents' users. In future, computers will be embedded in various objects that are placed everywhere, and we want to use these computers for executing applications for helping our work. In the environments, applications will be moved among computers that are embedded in objects existing near their users according to their location. The applications are called *migratory applications*[1, 2].

However, since computers may have dramatically different hardware and software configurations, these applications cannot be moved without taking into account the configurations of the computers that the applications are migrated. Therefore, migratory applications should be aware of environments in which the applications are executed, and should be adapted to respective computing environments to which the applications are moved[4]. Also, these applications may need to process various types of media data such as video and audio for monitoring surrounding environments of the applications' users, or showing video and audio information that is interested by the users in a timely fashion. Therefore, environment-ware migratory continuous media applications are one of the most important classes of migratory applications in future computing environments.

In this paper, we present a system support for building environment-ware migratory continuous media applications. The work described in this paper is one of case studies showing what is required for supporting migratory applications. The experiences with building the system support clarify problems and requirements for realizing migratory applications in large scaled heterogeneous distributed environments.

2 Issues for Supporting Environment-Aware Migratory Continuous Media Applications

For building environment-aware migratory continuous media applications, the following three issues should be taken into account.

- How do applications recognize computing environments in which the applications run, and how do they notify their changes to the applications ?
- How do continuous media applications should be adapted to respective computing environments ?
- How are applications migrated to other computers in the middle of their executions ?

Our system support consists of three components as shown in Figure 1, which answers the above respective issues, and it makes it possible to build environment-aware migratory continuous media applications in a systematic fashion.

The first component that answers the first issue is an environment server that monitors computing environments, and it notifies the changes in the environments to applications. The environment server provides primitives for accessing information about computing environments. Traditionally, respective information about computing environments requires to be accessed by different primitives.

[1] † Currently, Tatsuo Nakajima is a research engineer at Olivetti and Oracle Research Laboratory and a visiting scholar at Department of Engineering of University of Cambridge.

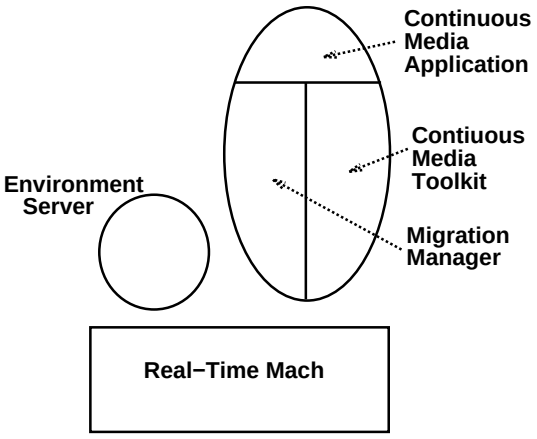


Figure 1: Structure of Environment-Aware Migratory Continuous Media Application

This makes the structures of environment-aware applications complicated. On the other hand, the uniform interface provided by the environment server enables us to build environment-aware applications in a systematic fashion.

The second component that answers the second issue is a continuous media toolkit that enables us to build continuous media applications in a highly configurable way. By using the toolkit, continuous media applications are constructed by composing several modules. The approach has two advantages. The first advantage is that the toolkit enables us to build continuous media applications with a little programming. The second approach is that the toolkit enables applications to change their configurations according to respective computing environments. The advantage makes it possible to build environment-aware continuous media applications systematically.

The last component that answers the third issue is the migration manager. The migration manager that makes continuous media application migratory has two functions. In the first function, the migration manager manages the reconstruction of an application on a computer that the application will be moved. In the second function, it saves the states of applications, and restores the states in applications that are reconstructed on a new computer. In our approach, the amount of states of an application can be changed by taking into account the tradeoff between the time for migrating the application and the bandwidth of a network connected between two computers.

The above three components are currently implemented on Real-Time Mach[8, 5]. The environment server is implemented as a server, and the continuous media toolkit and the migration manager are implemented as libraries that are linked with applications.

3 Environment Server

The *environment server*[7] is important as a basic infrastructure for building environment-aware applications. In traditional operating systems, information about computing environments is managed in a very ad-hoc way, and applications require to access the information using different primitives for respective information. The approach makes the development of applications that are aware of various differences in computing environments very complicated. The *environment server* contains a database that manages environmental information in a uniform way. The interface of the environment server is well defined, and an application can access environmental information by using its uniform interface. This makes it possible to build environment-aware applications with a systematic framework.

4 Toolkit for Building Environment Aware Continuous Media Applications

In our toolkit, continuous media applications are defined as compositions of several modules. This makes it easy to build continuous media applications significantly. Also, the approach enables applications to change their configurations in the middle of the executions. The characteristic makes us to build environment-aware continuous media applications that can adapt to respective computing environments according to the configurations of computers on which the applications run.

Also, our toolkit hides many complicated programming for building continuous media applications such as real-time programming, media synchronization, and dynamic QOS control from programmers for making it easy to build complex continuous media applications[6].

5 Migration Manager

The scripting language that is used for writing environment-aware continuous media applications is an extension of the TCL scripting language. Programmers can write continuous media applications using the script languages since the scripting language provides several primitives for calling the functions of our continuous media toolkit.

5.1 Protocols for Migrating Application

When a continuous media application detects that it should be migrated to other computers, it needs to negotiate to a computer that the application will be migrated. Let us assume that the computer executing the application currently is *computer A* and a new computer that will execute the application after the migration is *computer B*.

First, *computer A* sends a request to *computer B* for creating an application on *computer B*. As described in the previous section, our continuous media application uses the same binary. Thus, *computer B* runs the binary when the request is received. We assume that the binary is created by compiling for respective machine architectures, and it is stored in respective computers that our migratory applications may be migrated.

After creating the application on *computer B*, the newly created application sends a request for acquiring a script from the application executed on *computer A*. Then, the application sends the script to *computer B*. After receiving the script program, the application running on *computer B* configures modules and streams according to the script

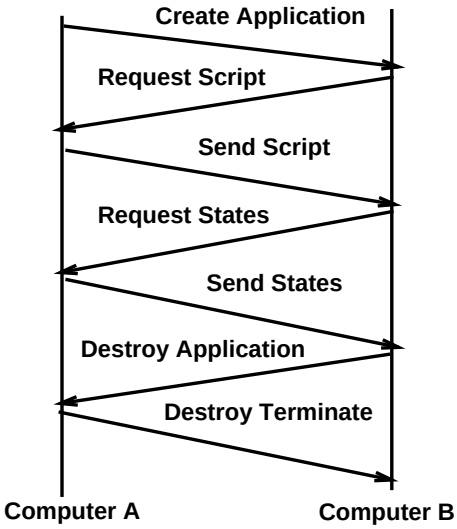


Figure 2: Application Migration Protocol

program as described in Section 5.3. Also, *Initialize* procedure in the script language is executed. The procedure includes necessary initializations for the application on *computer B* such as the registration of conditional statements to the environment server.

Next, the application running on *computer B* sends a request for saving the checkpoint of every module of the application running on *computer A* and transferring the checkpoint as described in Section 5.4. On *computer A*, the checkpoint of an application is taken, and the checkpoint is transformed to the standard format by calling the *Checkpoint* procedure in the script. Then, the checkpoint is transfered to *computer B*. The checkpoint is transformed from the standard format according to the current configuration of the application on *computer B*, and restored in the application's modules by calling the *Restore* procedure in the script. Also, the states for respective streams and GUI are transfered and restored in the application on *computer B*. Lastly, the application running on *computer B* sends a destroy request to the application running on *computer A*. Then, the application on *computer A* executes *Finalize procedure* in the script for unregistering conditional statements in the environment server, and the application is destroyed on *computer A*. Finally, the application on *computer B* starts to be executed after the execution of the destroy request is terminated.

5.2 Reconfiguring Application

In the previous section, we describe how an application is migrated to other computers during the execution. This section describes how the application is adapted according to the configuration of computers that the application is migrated.

Figure 3 shows an example of a continuous media application that captures a video stream from a video camera, converts the video stream, and displays it to a display. Let us assume that the application is migrated between two computer A and B. Computer A has a high resolution video capture, and a window system. When the application is executed on computer A, it connects the capture A module that controls a high resolution video capture board with the convert module that converts a video stream. Also, the convert module is connected with the display A module that draws a video stream to a window. Now, let us assume that the application is migrated to computer B. Since computer B has a low resolution video capture card and a small display, the capture A module and the Display A module cannot be used on the computer B. Therefore, the application changes the configuration by replacing the connections as described in Figure 3 for taking into account the hardware configuration of computer B. After the computer B receives a script program, the application is reconstruct on the computer. Then, the display A module is replaced by the display B module, and the capture A module is replaced by the capture B module. On the other hand, if the application is migrated to computer A again, the configuration of the application is recovered to the original configuration.

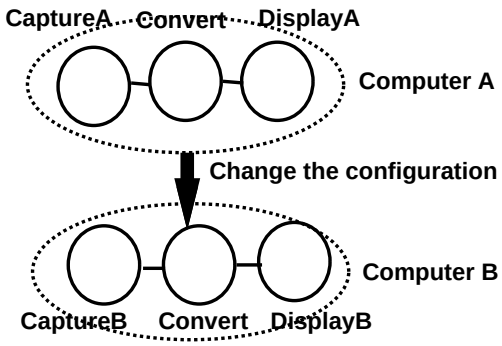


Figure 3: Reconfiguration of Application

In the migration manager, a script program manages the reconfiguration of an application when it is migrated to other computers. When the application is migrated to the computer whose configuration is drastically different from the original computer, the application receives a notification from the environment server, and the script program accesses the environment server for getting the configuration information of the new computer. In the example, the script program acquires information about a capture card and a display device of a new computer from the environment server, and the configuration of the modules is changed by executing the script program.

6 Sample Application

In the example application as described in Figure 4, a computer captures a video stream from a camera in front of a user. Another computer that receives the video stream is displayed to a window. We call the application that captures the video stream a *capture application*, and the application that displays the video stream a *display application*.

Now, let us assume that the *capture application* runs on computer 2, and the *display application* runs on computer 1. The capture application is migrated from computer 2 to computer 3 when the user is moved to near computer

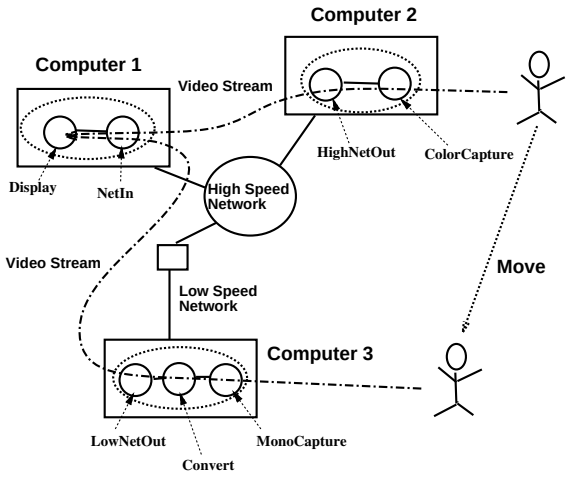


Figure 4: Application Mobility Support

3. Computer 2 is connected to a high speed network, and it has a color capture card. On the other hand, computer 3 is connected to a low speed network, and it has a monochrome capture card.

Before the application is migrated, the capture application is executed on computer 2, and it connects the HighNetOut module and the ColorCapture module. When the application is migrated to computer 3, its configuration is changed as described in Figure 4 since computer 2 and computer 3 have different types of capture cards. Also, since the connected network of computer 3 is low bandwidth, the captured video stream should be scaled down by degrading its quality. In this application, media elements buffered in computer 2 need not to be transfered to computer 3 since these media elements are not important for reconstructing the capture application after its migration.

7 Related Work

Teleporting[10] provides user interface mobility. In this system, when a user moves to other rooms, his windows that he is now working on will be moved to the nearest computer from him according to his location. The system takes into account the different characteristics of display devices that windows are migrated, but the ability is restricted by the limitation of the X window system.

Mobile agents[3, 9] is one of promising approaches for future computing environments. The agent moves among world wide computers and gathers information for helping his user's work. From a view of system research, the most important point of mobile agents is the mobility awareness. This means that the agent receives a notification when it is moved to other computers, and does a job according to the location of the agent. The approach does not take into account the adaptation of the agents according to computing environments. Therefore, traditional mobile agents degrade their performance if the agents are migrated to the computers with the machine configurations that the agents do not assume. On the other hand, our approach solves the problem by changing the configurations of applications according to computing environments by generalizing a notion of migration awareness to a notion of environment-awareness.

8 Conclusion

In this paper, we described a toolkit for supporting environment-aware migratory continuous media applications, and presented three components that we have been developed. The first component is the environment server. The second component is the continuous media toolkit. The last component is the migration manager. The components enable us to build environment-aware migratory applications in a systematic way. We implemented a prototype version of the components on Real-Time Mach, and build an example program of environment-aware continuous media applications.

References

[1] K.A. Bharat, L. Cardelli, "Migratory Applications", SRC Research 138 Report, Digital, 1996.

[2] L. Cardelli, "A Language with Distributed Scope", Computer Systems, 8(1), MIT Press, 1995.

[3] C. Harrison, D. Chess, and A. Kershenbaum, "Mobile Agent: Are they a good idea ?", Research Report RC19887, IBM Research Division, 1995.

[4] A.Hokimoto, K.Kurihara, T.Nakajima, "An Approach for Constructing Mobile Applications using Service Proxies", The 16th International Conference on Distributed Computing Systems, May, 1996.

[5] T.Nakajima,T.Kitayama,H.Arakawa,and H.Tokuda, "Integrated Management of Priority Inversion in Real-Time Mach", *In Proceedings of the Real-Time System Symposium*, 1993

[6] T. Nakajima, "A Toolkit for building Continuous Media Applications", In Proceedings of the 4th International Workshop on Real-Time Computing, Systems, and Applications, 1997.

[7] T. Nakajima, H. Aizu, M. Kobayashi, K. Shimamoto "Environment Server: A System Support for Adaptive Distributed Applications", In Proceedings of the International Conference on World Wide Computing and its Applications'98, 1998.

[8] H. Tokuda, T. Nakajima, and P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System", *In Proceeding of the USENIX 1st Mach Symposium*, October, 1990.

[9] J.E. White, "Telescript Technology: The Foundation for the Electric Marketplace", General Magic Inc., 1994.

[10] K.R. Wood, T. Richardson, F. Bennett, A. Harter, A. Hopper. "Global Teleporting with Java: Toward Ubiquitous Personalized Computing", *Computer* February 1997, Volume 30, Number 2, IEEE.

QoS Assurance vs. Dynamic Adaptability for Applications

Saleem N. Bhatti and Graham Knight
Computer Science Department, University College London, Gower St., London WC1E 6BT, UK
S.Bhatti@cs.ucl.ac.uk and G.Knight@cs.ucl.ac.uk

Abstract

Enabling adaptability to network QoS (quality of service) is seen as a key feature for future applications. One way to approach adaptability is to build it into the network and allow applications to signal their requirements to the network. This means that resource reservation mechanisms must be available end-to-end, which is not always the case, especially on the Internet. Also, user preferences affect how the application is used. So the application must be dynamically adaptive, taking into account the application's capabilities, the available network QoS and the user preferences. We suggest that it is possible to build practicable QoS summaries that capture all these inputs and allow dynamic adaptability.

1. Introduction

In today's Internet, applications face operation with a range of connectivity/access network technologies, QoS (quality of service) fluctuations throughout the network and host mobility. This means that today's applications must be adaptable enough to match their mode of operation to the resources they find available in the network. The nature of the network is such that there may be QoS variations during the lifetime of a single application flow. This is especially troublesome for multimedia applications with QoS sensitive flows. While applications show some dynamic adaptation capabilities (e.g. use of elastic buffering to counter jitter on flows), they generally require the user to select the correct configuration or user preferences for the application to function in a given situation. Typically, the user of such applications may lack the expertise or knowledge about network conditions to complete this task adequately. Even if the user has the correct application configuration, the best-effort service offered across the Internet means that applications typically experience QoS fluctuations and so the configuration may need to be changed during the lifetime of any single application instance. We would like one or all of:

- resource reservation mechanisms to ensure that applications receive the quality of service they require
- the network to provide suitable adaptation mechanisms which allow communication to continue when QoS fluctuates
- applications to be dynamically adaptable, being able to configure their flow construction appropriately to match the detected network QoS

2. QoS assurance

In [1], there is an excellent description of the elements of a general QoS architecture for assuring QoS. With QoS assurance mechanisms, applications have to specify what they want the network to do for them, and the network tries to honour this request. The network itself may provide adaptability mechanisms [2] to cope with fluctuating QoS or heterogeneity due to network architecture. [1] suggests that (among other things) the application should be in control of the following information that passes to the network: QoS service-level (e.g. guaranteed-service, controlled-load service) and QoS management policy (how adaptation should take place when there are QoS fluctuations). These affect the way the application is integrated with the network (the separation of functionality between the network and the application), and how heterogeneity is supported (at the user, application and network access level).

QoS assurance mechanisms are based around the use of resource reservation and QoS re-negotiation to achieve the desired QoS for the application. Mechanisms such as RSVP [3] are designed to provide resource reservations in the Internet. However, RSVP can fail [4], and adaptability mechanisms in the network become impracticable and increasingly difficult to support in *ad hoc* or mobile network architectures. Additionally, where resource reservation and QoS re-negotiation are used, they should be available end-to-end for maximum utility. In today's Internet there is very little resource reservation support and network components lack the programmability required to enable adaptability in the network. Furthermore, as such mechanisms do become available, they will not be deployed uniformly, and this will further compound the heterogeneity in the Internet. For many applications, such QoS assurance or adaptability mechanisms within the network may only be useful if they

are available end-to-end. Where these mechanisms do exist, there is great value in being able to use them, but in the "standard" best-effort Internet, other mechanisms are required to address adaptability.

3. Dynamic adaptation decisions

There is currently great interest in making applications adaptable to changes in network QoS. This is of particular importance for real-time media flows or flows that are sensitive to QoS fluctuations. Much of the attention for this work has focused on mechanisms that allow adaptability of the *media flow construction* (which we call the **flow-state**) by scaling (e.g. [5]), filtering (e.g. [6]) or encoding techniques (e.g. [7] [8]). An area that has received little attention, however, is how applications can automatically select the correct flow-state to match the available network QoS as detected during operation. The application must currently rely on the user to set the correct preferences to allow operation in a particular QoS regime. Although application specific mechanisms exist to allow some automatic adaptation (e.g. elastic buffering to combat jitter in audio tools), there is little support for mechanisms that allow applications to automatically and dynamically make decisions about their flow-state, based on user preferences and measured network QoS values for that flow.

There may be many instances of an application used by many different users. Each of the users will have their own preferences for the application. Any decision making process for adaptation must allow the application to apply user preferences to the QoS being experienced by the flow. For example, with an audio tool with numerous encoding schemes, one user may specify that the tool "always uses the best audio quality possible" while another user may specify that the tool aim to "retain stability" in the face of network QoS fluctuations. The application may interpret these preferences as meaning the "encoding with the highest data rate requirement whenever possible" and "do not change flow-state too often", respectively. So the adaptation decision is not only application specific but also application *instance* specific, and controlled by user preferences.

4. Dynamic adaptability in applications

For the provisioning of end-to-end QoS, the user/application is telling the network what is required and asks that the network should adapt/configure itself to comply to the application's requirements, i.e. static adaptability. In enabling dynamic adaptability, the application is constantly (i.e. within application specific time scales) receiving information about the QoS that the network can offer, and then changing its flow-state to comply to the network capabilities.

For assuring QoS, the integration principle says that we need QoS to be "configurable, predictable and maintainable over all architectural layers" [1]. With this in mind, the principle of integration does not have to apply so strongly for adaptable applications. What we require is a well-defined interface between the application and network to receive an application oriented view of the network capabilities; i.e. something that is easily usable and understandable at the application level. What is required is a report that summarises the compatibility between the current network QoS ant the flow-states possible for an application. Our weaker integration requirement is an advantage and a disadvantage. It is an advantage in that it requires looser coupling between the various architectural components giving greater freedom in system design. In fact, this looser integration can be seen as a requirement for Internet applications, as IP is not biased towards to any particular network technology. This latter reason is a disadvantage – we can not directly exploit any network specific mechanisms.

Additionally, the QoS service-level required may be important to the individual users and may determine the cost of the service, e.g. guaranteed-load is a "better" service than controlled-load so may cost more. We argue that the service-level should not be specified by the application. The application should be prepared to be more flexible in its adaptation capability leaving service-level selection to the user. Three reasons for this are:

1. the service-level may determine the cost of the service and users usually wish to control how much they pay.
2. network heterogeneity, lack of resource reservation or network element failure may mean that a particular service-level is not available at a given time at a given point in the network.
3. new, additional service-level definitions may be introduced that are more suitable (in terms functionality or cost) for use in a given situation, e.g. there are two recent descriptions of **adaptive** service-level [9] [10], and differentiated services may be set up that are domain/administration specific [11].

Additionally, if cost-based feedback is available from the network, then cost could be treated as a QoS parameter – although not related to the performance of the flow it acts as a defining constraint in the same way as, say, defining minimum data rate or maximum jitter for a flow. The value of making the cost explicit as a QoS parameter is that it highlights the importance of cost as a feedback control mechanism in future services [12].

QoS management policy will be subject to user preferences and application specific behaviour. Applications may find it useful to have a specification of the QoS management policy before the application starts operating. This

would certainly be of value to the network for controlled allocation of resources, and makes sense in the context of trying to assure end-to-end to QoS. However, in our consideration of dynamic adaptability, the use of the application typically requires interaction from the user in order to determine its adaptation requirements, and these may not be known until after the application is running. In [1], the QoS management policy:

> *captures the degree of QoS adaptation (continuous or discrete) that the flow can tolerate and the scaling actions to be taken in the event of violations to the contracted QoS [9].*

We chose to make a separation between what "*the flow can tolerate*" and the "*scaling actions to be taken*". We argue that the former is a property of the media and the latter is an application specific requirement that includes interaction with the user. Flow performance specifications can be used to indicate the flow-states that are possible for a flow and can be determined by the application designer. The action to be taken on fluctuations ("*violations*") of QoS is a dynamic adaptation decision and cannot be determined by the application designer *a priori*. It is the difference between the application designer saying "I know what is sensible for the application" and the user saying "I know what is sensible for the application to do for me". Ultimately the application's functional constraints have the final say on which flow-state(s) is (are) functionally *possible*, but this should not dictate *how* the user would like the application to behave, i.e. how adaptation should take place. As an example, consider a remote teaching scenario and the requirements of the audio and video flows. When operating in lecture mode (main part of the teaching session), the conferencing application may tolerate relatively high delay and throughput is asymmetric, but during a question and answer session (at the end of the teaching session), low delay and jitter are required with symmetric throughput for flows.

The application must be able to assess the user preferences and available network QoS in order to make automatic and dynamic adaptation decisions.

5. Application and network compatibility

The adaptation decision for the application is to select a flow-state. For example, given suitable information about loss, achievable data rate, etc. for its flow, an audio application may be able to choose an audio coding that matches the available network QoS by adapting the data rate, or by using forward error correction, etc. Real-time, QoS sensitive Internet applications may use mechanisms such as RTCP [13] (or application specific mechanisms) to distribute such QoS information. In order to select a flow-state that "matches" this information, the application must assess the compatibility between the network QoS, the flow-state requirements and the user preferences.

In [14], we use a mechanism – which we call the **QoSSpace** – that produces a **QoSReport** summarising the compatibility between the network QoS and an application's flow-states. The QoSReport contains a **state compatibility value (SCV)** for each flow-state. The SCV is a number in the range [0, 1] indicating the instantaneous assessment of compatibility between a flow-state and the network, with 0 indicating that the QoSSpace has no confidence that the flow-state cannot be supported by the network, and 1 indicating that the QoSSpace has high confidence that the flow-state can be supported. In order to produce SCVs, the QoSSpace needs QoS parameter measurements for the flow. These can come from any application specific mechanism.

The decision of which flow-state to use depends on network QoS, application capability and user preferences. This decision can be performed in an automatic (no run-time interaction with the user) or semi-automatic fashion (interaction with user before making a flow-state change). The decision mechanism is embodied in an **application adaptation function (AAF)** which takes information about the user preferences, application flow-states and network QoS (QoSReports). In the automatic case, the user indicates user preferences such as "use lowest cost service" at start-up and the AAF automatically selects the most suitable flow-state. In the semi-automatic case, the user can be given the ability of making an informed decision at run-time by the use of SCVs from the QoSReport.

Our QoSSpace model has some similarities to that of a general control system process (Figure 1). However, it has significant differences in its aims and its function.

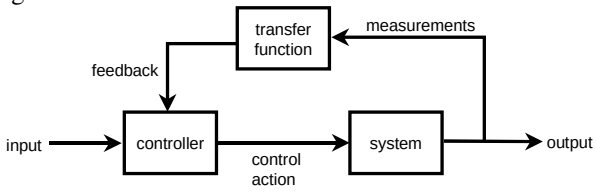


Figure 1: A general control system

Many adaptive mechanisms (e.g. congestion control) rely on the use of a control systems approach to adjust their flows. In a traditional control system (Figure 1), the aim of the *controller* is to generate *control actions* that keep the *system* at a set-point (a defined operating point for the *system*). The *transfer function* uses *measurements* of specific parameters from the *output* of the *system* to produce *feedback* for the *controller*. The *controller* uses the *feedback* information with other *inputs* to decide the *control action* it should take in order to maintain the required *system* behaviour, i.e. maintain the *set-point*.

In our model in, we can identify similarities:

- the *system* is the application and the *output* is the flow as it traverses the network
- the *measurements* from the *output* are a set of QoS parameter measurements for the flow
- the *transfer function* is analogous to the QoSSpace which generates the QoSReports as a *feedback* summary of *system* behaviour
- the *controller* is analogous to the AAF and the *control action* will be to select a suitable flow-state, which may affect the *system* (the application) and the *output* (the flow)

However, our dynamic adaptation scenario differs from the traditional control system in three important ways:

1. the QoSSpace does not model the *system* input/output but provides a summary of the relative **compatibility** of the network QoS offered to the flow and the possible flow-states
2. the contents of the QoSReports from the QoSSpace may be modified by the application and by user preferences
3. the behaviour of the AAF may be modified by user preferences or inputs from other application specific sources

So, overall, while we do have a feedback control process, the *measurements* are processed in a different manner to that of a traditional control system, and so the *feedback* is of a different nature. Essentially, the aim of our model is not that same as that for a traditional control system; instead of attempting to **maintain** a set-point, our model seeks to allow the application to **select** a flow-state. (Our work in [14] is ongoing.)

6. Conclusions

The Internet applications of the future will need to be adaptive in order to cope with network heterogeneity, differing QoS from the service provider, fluctuating QoS during operation, and differing user preferences. Cost of service will play an important part for selecting service-levels and so the QoS available to an application. Decisions regarding cost will be made by the user. While mechanisms for QoS assurance and adaptability in the network may be available, such mechanisms may not be uniformly deployed or available end-to-end. Applications must be prepared to adapt their behaviour as well as use mechanisms within the network. The adaptation capability in applications must be dynamic and automatic, and should not rely on the user to have expert knowledge. It seems possible to create QoS compatibility summaries that can take into account the applications flow-states, the network QoS and user preferences in order to enable dynamic adaptability. We find that there may be conflicts in requirements between QoS assurance mechanisms and dynamic adaptability requirements for the application. These issues are for further study.

7. References

[1] A. Campbell, C. Aurrecoechea, L. Hauw, "A Review of QoS Architectures", Proc. of 4th IFIP International Workshop on Quality of Service (IWQS'96), Paris, France, Mar 1996.

[2] A. T. Campbell, "Mobiware: QoS-Aware Middleware for Mobile Multimedia Networking", Proc. IFIP 7th International Conference on High Performance Networking, White Plains, New York, Apr 1997.

[3] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification", RFC2205, Sep 1997.

[4] L. Wolf, C. Gridwodz, R. Steinmetz, "Multimedia Communication", Proceedings of the IEEE, vol. 85 no. 12, pp-1915-1933, Dec 1997.

[5] L. Delgrossi, C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, C. Vogt, "Media scaling for Audio-visual Communication with the Heidelberg Transport System", Proc. ACM Multimedia, pp99-104, Jun 1993.

[6] N. Yeadon, F. Garcia, D. Hutchison, D. Shepherd, "Filters: QoS Support Mechanisms for Multipeer Communications", IEEE Journal of Selected Areas in Communication, vol. 14, no. 7, pp1245-1262, Sep 1996.

[7] V. Hardman, M. A. Sasse, M. Handley & A. Watson, "Reliable Audio for Use over the Internet", Proc. INET'95, pp171-178, Hawaii, USA, 27-30 Jun 1995.

[8] E. Amir, S. McCanne, M. Vetterli, "A Layered DCT Coder for Internet Video", Proc. ICIP'96, Lausanne, Switzerland, Sep 1996.

[9] A. Campbell, G. Coluson, D. Hutchison, "Supporting Adaptive Flows in Quality of Service Architecture", ACM Multimedia Systems Journal, 1996, May 1996.

[10] S. Lu, K.-W. Lee, V. Bharghavan, "Adaptive Service in Mobile Computing Environments", in Building QoS into Distributed Systems, (A. Campbell, K. Nahrstedt, Eds), pp25-36, [Chapman & Hall] 1997

[11] K. Nichols, S. Blake (Eds), "Differentiated Services Operational Model and Definitions", IETF DIFFSERV WG, work-in-progress, Feb 1998.

[12] S. Shenkar, D. Clark, D. Estrin, S. Herzog, "Pricing in Computer Network: Reshaping the Research Agenda", ACM Computer Communications Review, vol. 26. No. 2, pp19-43, Apr 1996

[13] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC1889, Jan 1996.

[14] S. N. Bhatti, G. Knight, "Notes on a QoS information model for making adaptation decisions", to appear, HIPPARCH'98, UCL, London, UK, 15-16 Jun 1998.

A Perspective on Application/Network Coupling

Thomas Gross[1,2] and Peter Steenkiste[1]

[1]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[2]Departement Informatik
ETH Zürich
CH 8092 Zürich

Abstract

Networked applications execute in a distributed environment that is shared with many users, so the level of service they receive from the network can be highly variable. Since the characteristics of the network often determine overall application performance, network-aware applications attempt to adjust to changes in the network conditions. Therefore, such applications must be able to obtain performance-related information about the network. We discuss two dimensions of the coupling of such applications to the network system. One dimension focuses on how the application learns about the state of the network: The application can use implicit information, e.g. the packet loss rate it observes, or it can receive explicit information from the network about topology and dynamic traffic conditions. The other dimension covers the model of interaction between application and network: Either the information is collected continously and the application controls when it acquires information about the network (e.g., the application consults a database maintained by the network), or the interaction is triggered by specific network events (e.g., the network notifies the application of significant changes). Network-aware applications today almost exclusively use the implicit style of information collection, in part because it does not require any support from the network. In this paper we argue that explicit information is needed as well if we want to give applications the opportunity to explore the full range of adaptation options. To demonstrate the benefits of explicit coupling, we developed Remos (Resource Monitoring System), an interface that allows applications to collect information about network conditions. We briefly describe the design and prototype implementation of Remos.

1 Introduction

Many networked applications are very sensitive to timely responses of the communication system. Integrated applications that mix traffic types (data, video, audio, control) often face difficulties to obtain adequate network services from today's networks. The response of network architects to such application demands has been the definition of different service classes. Some service classes provide hard guarantees (e.g., the IETF guaranteed service), or weaker "guarantees" (e.g., the IETF controlled load service). Researchers are also exploring the notion of service differentiation, i.e. multiple classes with different (relative) service quality are provided, but no guarantees are made about the (absolute) level of service that will be received.

This development is a mixed blessing for application developers. Guaranteed service is an attractive solution, but it is hard to use and likely to be relatively expensive, especially for bursty traffic, and it may not be deployed for a long time. Service classes such as controlled load are easier to deploy and use, but since they do not provide strict guarantees, applications will have to be able to adapt within some window of network performance. Best effort services are likely to be most widely available and most economical, but applications may have to adapt over a very wide performance window. The net result is that while some of the proposed service classes will make it easier to develop applications that transfer audio, video, and time sensitive data, most service classes still require applications to adapt to dynamic network conditions. Moreover, the application (developer) must now also select an appropriate service class.

2 Problem

It is advantageous for applications to be network aware since it allows them to run under a broader range of conditions, and it may also allow them to use less expensive service classes, e.g. best effort instead of guaranteed service. Some of the functionality (and complexity) associated with adaptation can be embedded in middleware, but we first have to understand the adaptation process before we develop widely usable libraries or middleware layers.

2.1 Space of feedback options

To adapt, applications need information on the status of the network. Traditionally, this task has been performed by communication protocols, such at TCP, so it is worthwhile to look at how these protocols collect information about network conditions. Protocols are often classified as using *implicit* or *explicit* feedback from the network. In protocols based on implicit feedback, the receiver monitors the incoming data stream and uses this stream to derive information about network conditions. TCP is good example: dropped packets are viewed as a sign of congestion, and the sender responds by reducing its rate. In contrast, with explicit feedback, some entity inside the network provides explicit information about network conditions to senders. A good example is the ATM ABR traffic class: senders receive periodic information about network congestion conditions (e.g., congestion bit in rate management cells), or even the specific maximum traffic rate they are allowed to use (e.g., EPRCA).

Implicit feedback has the advantage that it does not require support from the network so this approach to provide feedback is always feasible[1]. Implicit feedback has also disadvantages: (i) it only allows incremental adaptation (i.e., when two hosts communicate, implicit information provides updates on how the bandwidth between these hosts evolves), and (ii) it is sometimes difficult to interpret the "information". Explicit feedback is in general easier to use, but it requires network support. Protocols today primarily rely on implicit feedback, and the same is true for initial network-aware applications. The reason is simple: implicit feedback does not require networking support, which does not exist.

Explicit information can be provided to the application in two ways. First, the network can provide feedback continuously. This approach is, e.g., employed for ABR traffic: a rate management cell is exchanged with the network for every 32 data cells. Continous feedback is most often based on network properties that subsequently must be interpreted in the application space; for this reason this kind of coupling is also called *indirect*. An alternative is that applications receive a notification when specific events happen, e.g, an application receives an asynchronous notification when the network bandwidth drops below a certain threshold or when the connection switches from one type of network to another [7]. Such notifications can be in the form of callbacks, or by invoking a specific event handler. With this style of interaction, the relationship between the network event (e.g., drop in bandwidth) and the actions (by the application or protocol software) is clearly established (e.g., when registering the handler). Therefore we call this style *event-driven* or also direct coupling.

Finally, it useful to distinguish between the interface used by the application and the functionality supported by the network, since it is possible for a library or middleware layer to translate one interface into another. For example, a library could translate continuous network feedback into event-based application feedback. A more interesting approach includes activities by the middleware: middleware could use a set of benchmarks to collect information on the conditions in a network (that does not provide explicit feedback) and present this information to the application in an explicit form. We focus on the application level interface, and we will touch on the lower level interface when we discuss implementation options.

2.2 Space of application adaptation options

The above discussion indicates that network-aware applications may need to adapt to the network in two ways: (i) pick the most appropriate service class, if multiple service classes are available, and request service (if not a best effort class); (ii) adapt to the service level available in the current service class. The first step requires an interface that allows applications to discover what services are available. The second step can in principle be done using implicit network feedback, e.g., reduce the frame rate or size if the receiver observes corrupted frames, and periodically probe for more bandwidth if the connection is lossless. This model of network-awareness is appropriate for many applications and can be encapsulated in a framework[1].

However, there is more to network awareness than adjusting frame rates for video streams or generating the appropriate parameters needed to create a connection with delay guarantees. The above model focuses on the "steady state" (of adaptation, not necessarily a steady state in the network!). E.g., it leaves open the issue of how to deal with start up, and it also does not support more interesting forms of adaptation, e.g., switching to a higher or lower quality service class during execution.

If we look at the general problem of how applications can adapt to network conditions, we see a multi-dimensional space of options. The three main dimensions are time, space, and service classes.

Along the *time* dimension, applications dynamically adapt to changing network conditions. Adaptation is in general based on the assumption that recent past conditions are a good predictor of near-term future conditions, an assumption that often holds. Adaptation along the time dimension is addressed by protocols such as TCP, and has also received the most attention from researchers studying network-aware applications.

Adaptation along the *space* dimension reflects the fact that applications sometimes have a choice about what

[1] Implicit feedback typically makes some assumptions about the network, e.g., it considers packet loss to be a sign of congestion.

nodes, and thus what part of the network, to use. A simple example is a Web browser that can choose which of several replicated servers to use, or that can select a proxy cache that may have a potentially stale copy of a video clip. Another example is a distributed simulation that has a choice of how many and what nodes it should use.

The *service class* dimension reflects the fact that the application has to pick a service class, and this decision may (should) be based on dynamic conditions. E.g., when setting up a video conference over a network that supports differentiated service, the user or application will want to pick the lowest service class, if that can provide sufficient bandwidth. A higher service class will be selected if it can deliver the bandwidth while the lower service class cannot.

2.3 Network-application interactions

To be able to adapt along all three dimensions, applications will need information on network conditions, which span a matching space with the same dimensions. How applications collect network information impacts how easily applications can explore this space.

Implicit information is based on experience, which severely restricts what part of the information space is explored: the application only learns about the part of the space it currently operates in. This means that it can only collect information on the network conditions along the paths it is currently using and on the service class it is in. Implicit feedback does provide information along the time dimension, but only while the application is actively using the network. At startup or after the application has been idle for a while, no useful information is available.

We argue that given the limits on what information can be collected using implicit feedback, mechanisms must be provided so that applications can get explicit information on network conditions, e.g., by querying an standard interface. Such an interface should allow applications to collect information on network conditions in the entire network space (space, time and service class dimensions), allowing applications to make adaptation decisions in space, across service classes, and at startup.

One can argue that the restrictions on the type of information that can be collected using implicit feedback is not fundamental. Applications can use probing to explore the entire information space, e.g., they can periodically try all service classes and they can measure measure the network performance between every pair of usable hosts. While this approach may be appropriate in some cases, it is in general undesirable. First, developing effective network probing routines is difficult; it is not something that application developers should be required to do. Second, probing can be expensive, both in terms of elapsed time for the application and consumed network resources. Furthermore, excessive probing may disturb the measurements taken by this and other applications. In fact, large scale probing by applications would negate many of the advantages of implicit feedback. If probing is needed, it should be performed by a middleware layer, which would mean that the code can be developed as part of the network architecture, and the collected information can be shared by many applications.

Note that we are proposing explicit feedback as a complementary mechanism to implicit feedback, and not as a replacement. The advantages of implicit feedback, if used appropriately, remain. Implicit feedback will remain useful as an inexpensive way of getting continuous feedback, once a particular operating point along the space and service dimensions has been picked. Implicit feedback is also like to give more accurate and timely information (in a narrower part of the operating space) than explicit feedback.

3 Example: Remos

The Remos API provides a query-based interface that allows clients to obtain "best-effort" information [3] on network conditions. The applications specifies the kind of information it needs, and Remos supplies the best available information. To limit the scope of the query, the application must select network parameters and parts of a larger network that are of interest. In this section we briefly describe the main Remos features. A more detailed description can be found elsewhere [3, 6].

3.1 Level of abstraction

To accommodate the diverse application needs, the Remos API provides two levels of abstraction: high level flow-based queries and lower level topology-based queries.

Remos supports flow-based queries. A *flow* is an application-level connection between a pair of computation nodes. Queries about bandwidth and latency of sets of flows form the core of the Remos interface. Using flows instead of physical links provides a a high level of abstraction that makes the interface portable and independent of system details. Flow-based queries place the burden of translating network-specific information into application-oriented information on the implementor of the API. However, flows are an intuitive abstraction for application developers and they allow the development of adaptive network applications that are independent of the heterogeneity inherent in a network computing environment.

Remos also supports queries about the network *topology*. The reason we expose a network-level view of connectivity is that certain types of questions are more easily or more efficiently answered based on topology information. For example, finding the pair of nodes with the highest bandwidth connectivity is expensive using only flow-based queries.

The topology information provided by Remos consists of a graph with compute nodes, network nodes, and links, each annotated with their physical characteristics, such as latency and available bandwidth. Topology queries return a *logical* interconnection topology. This means that the graph represents the network behavior as seen by the application, and does not necessarily reflect the physical topology. Using a logical topology gives Remos the option of hiding network features that do not affect the application. For example, subnets can be replaced by (logical) links if their internal structure does not affect applications. Topology information is in general harder to use than flow-based information, since the complexity of translating network-level data into application-level information is mostly left to the user.

3.2 Dynamic resource sharing

Since networks are a shared resources, it is important to account for how resources are shared by multiple flows. Since multi-party applications will use multiple flows, it is not only necessary to account for sharing across applications, but also across flows belonging to the same application. In order to be able to consider the effects of "internal" sharing, Remos supports multi-flow queries in which the application lists all its flows simultaneously. Applications can generate flows with very diverse sharing characteristics, ranging from constrained low-bandwidth audio to bursty high-bandwidth data flows. Remos collapses this broad spectrum into three types of flows. *Fixed flows* have a specific bandwidth requirement. *Variable flows* have related requirements and demand the maximum available bandwidth that can be provided to all such flows in a given ratio. (e.g., all flows in a typical all-to-all communication operation have the same requirements) Finally, *independent flows* simply want maximum available bandwidth. These flow types also reflect priorities when sufficient resources are not available to satisfy all the flows. *Fixed flows* are considered first, followed by *variable flows*, then *independent flows*.

Determining how the throughput of a flow is affected by other messages in transit is very complicated and network specific. Remos approximates this complex behavior by assuming that, all else being equal, the bottleneck link bandwidth is shared equally by all flows (that are not bottlenecked elsewhere). If other information is available, Remos can use different sharing policies when estimating flow bandwidths. The basic sharing policy assumed by Remos corresponds to the max-min fair share policy [4]. use topology-based queries are themselves responsible for taking the effects of both internal and external sharing into account..

3.3 Accuracy

Applications ideally want information about the level of service they can expect to receive in the future, but most users today must use past performance as a predictor of the future. Different applications are also interested in activities on different timescales. A synchronous parallel application expects to transfer bursts of data in short periods of time, while a long running data intensive application may be interested in throughput over an extended period of time. For this reason, relevant queries in the Remos interface accept a timeframe parameter that allows the user to request data collected and averaged for a specific time window.

Network information such as available bandwidth changes continuously due to sharing and as a result, characterizing these metrics by a single number can be misleading. For example, knowing that bandwidth availability has been very stable represents a different scenario from it being an average of rapidly changing instantaneous bandwidths. To address these aspects, the Remos interface adds statistical variability and estimation accuracy parameters to all dynamic quantitative information. Since the actual distributions for the measured quantities are generally not known, we present the variability of network parameters using quartiles [5].

3.4 Implementation

An initial version of Remos API has been implemented. It has two components, a Collector and Modeler, that are responsible for network-oriented and application-oriented functionality, respectively. The Collector is responsible for collecting low-level network information. Such data can be collected in many ways, e.g. one can periodically run benchmarks that probe the network for available bandwidth or rely on information collected by applications [8]. Our current implementation uses a third method: the Collector explicitly queries routers using SNMP [2] for both topology and dynamic bandwidth information. This is a standard way of monitoring networks and it should allow us to collect detailed information in a relatively non-intrusive way on a broad set of networks. The Modeler is a library that is linked with the application. It translates the information provided by the Collector into a logical topology graph or per-flow data in response to application requests. The Modeler-Collector architecture is in part driven by the need to address scalability issues and network heterogeneity. In large networked environments, multiple Collectors may have to be deployed and each Collector can collect information in a way that is most appropriate for the network it is responsible for. Work is in progress on implementing Collectors that use sources of network information other than SNMP.

	Explicit	Implicit
Indirect (continuous)	Queres to network management database Rate management cells	Counting lost packets
Event-driven (direct)	Handlers to react to changes in the network	Reacting to lost retransmissions

Figure 1: Examples for two dimensions of application/network coupling.

Some initial experiments using the Remos interface on a dedicated IP-based testbed at Carnegie Mellon University are described in [6].

4 Concluding remarks

Figure 1 gives examples of the 4 different kinds of coupling between applications and networks that we discussed. Network-ware applications today focus overwhelmingly on implicit interaction. We argue that this state of affairs is due to the current (lack of) support for other interactions models by network architectures. As network architectures begin to provide information that is more accurate, more timely, and more detailed, network-aware applications will be motivated to also explore explicit interaction. E.g., an application may at startup use explicit queries to obtain information about available nodes and links, and then use implicit feedback after the startup phase.

Acknowledgments

We appreciate contributions and comments by the Remulac and Darwin groups.

Effort sponsored by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

References

[1] J. Bolliger and T. Gross. A framework-based approach to the development of network-aware applications. *IEEE Trans. Softw. Eng.*, 24(9):(to appear), Sept 1998.

[2] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol operations for version 2 of the simple network management protocol (snmpv2), January 1999. RFC 1905.

[3] T. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland. Remos: A resource monitoring system for network-aware applications. Technical Report CMU-CS-97-194, Carnegie Mellon School of Computer Science, Dec 1997.

[4] J. M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, July 1981.

[5] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[6] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. In *Proc. 7th IEEE Symp. High-Performance Distr. Comp.*, page (to appear), July 1998.

[7] P. Steenkiste. Middleware adaptation models for network-aware application. Technical report, Department of Computer Science, Carnegie Mellon University, June 1998.

[8] M. Stemm, S. Seshan, and R. Katz. Spand: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, June 1997.

Network Adaptive Continuous-Media Applications Through Self Organised Transcoding

Isidor Kouvelas, Vicky Hardman and Jon Crowcroft
Department of Computer Science
University College London
{*I.Kouvelas,V.Hardman,J.Crowcroft*}*@cs.ucl.ac.uk*

Abstract

With the deployment of the Mbone, multimedia conferencing is becoming a common practice on the Internet. In order to coexist with traditional services like email and file transfer, mechanisms to fairly share available bandwidth have to be developed for real-time audio and video media streams. The network scale and heterogeneity in available bandwidth complicate the design of network adaptive multicast applications. This paper presents a new scalable architecture for congestion controlled multicast real-time communication. The proposed scheme uses self-organisation to form groups out of co-located receivers with bad reception and provides local repair through the use of transcoders. The receiver driven nature of the protocol ensures high scalability and applicability to large Mbone sessions. The viability of the proposed protocol is demonstrated through simulation.

1 Introduction

With the deployment of the Mbone, multipoint multimedia conferencing has become a common practice on the Internet. The network scale and heterogeneity in available bandwidth complicate the design of network adaptive real-time multipoint applications.

Audio and video conferencing applications can tolerate a certain amount of packet loss and delay jitter from the network. They have been designed to adapt to network conditions and minimise the perceived signal degradation by trading off reliability for interactivity [1, 2].

The UCL Robust-Audio Tool (RAT) [3] and Freephone developed at INRIA use forward error correction (FEC) techniques [4] and successfully address the loss problem with minimal increase in stream delay. FEC used is in the form of highly compressed low quality audio that is piggybacked on normal audio packets. The decision on the level of FEC to use is made per source based on receiver loss reports and is tailored to cover for the average or highest requirements of the receiver group. This strategy is only good for a group observing similar loss rates. In a diverse group receivers observing low loss are forced to receive useless redundant information whereas receivers with very bad loss may not be covered.

The variable network loss rates and perceived quality in different areas of a multicast distribution tree are a result of different link bandwidth availability and link load. The extent of this problem is best illustrated by the work of Handley [5] in figure 1. The graph represents the packet loss rates experienced by different receivers of a popular Mbone session over the period of the session. It is clear that although most receivers are seeing low to moderate loss, there are a small number of sites suffering.

This indicates that a single stream addressed to the whole group can not possibly cover the needs of all receivers. Instead the data rate and amount of redundancy has to be customised and separately distributed to problematic areas. It has been shown that sender driven schemes that try to address the receiver heterogeneity problem do not scale to large groups. Any scheme that attempts to separately cover for the different needs of problematic receiver subgroups has to be receiver driven to scale [6]. Subgroups of co-located receivers in a multicast delivery tree suffering from similar problems should co-ordinate their efforts in improving reception quality. Furthermore their attempts should not affect reception for the remaining participants in the multicast session.

Strict low delay requirements of real-time data distribution preclude solutions using retransmissions to achieve required reliability. The dynamic nature of the Mbone delivery and membership model does not allow for manually configured static schemes that work

around congested links.

The solution we present in this paper uses a self-organisation scheme to form groups out of co-located receivers with bad reception. A representative of the group is responsible for locating a suitably positioned receiver with better reception that is willing to provide a customised transcoded version of the session stream. The transcoding site thus provides local repair to the congestion problem of the group with minimal increase in stream delay. The data rate and level redundancy of the transcoded stream are continuously modified to adapt to the bottleneck link characteristics using reception quality feedback from a member of the formed loss group. Network friendly congestion control of the real time multicast stream can thus be achieved.

The rest of this paper is structured as follows. Section 2 describes related work on congestion control for multicast distribution and attempts to solve the group reception diversity problem. In section 3 we describe our self-organised transcoding solution to the problem. The proposed solution has been implemented and evaluated through simulation using the VINT network simulator [7]. The simulation results can be found in section 4.

2 Background and Related Work

The current multimedia conferencing architecture over the Mbone / Internet has the following characteristics:

- Conferencing applications use the Real-time Transport Protocol (RTP) [8, 9] to transmit information over an unreliable best-effort multipoint network.
- Receivers express interest in receiving traffic by tuning into a multicast address and the network forwards traffic only along links with downstream recipients.
- No knowledge of group membership or routing topology is available at the source or receivers.

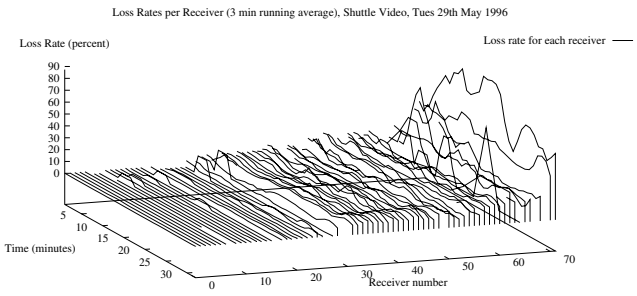


Figure 1: Loss rate against time for different receivers

The rest of this section discusses existing work and alternatives addressing the reliability issues for continuous media streams in a heterogeneous multicast environment.

2.1 Layered Encoding

McCanne et al [6] combine a layered compression scheme with a layered transmission scheme to address the network heterogeneity issue. The media stream is encoded into a number of layers that can be incrementally combined to provide refined versions of varying quality of the encoded signal. The individual layers are then transmitted on separate multicast addresses. Receivers adapt to network conditions by adjusting the number of levels they subscribe to and thus improving perceived quality by trading off average signal quality for packet loss. The application they propose for this scheme is multicast video. For this purpose they have developed a video codec that can compress a video frame providing very fine granularity layers.

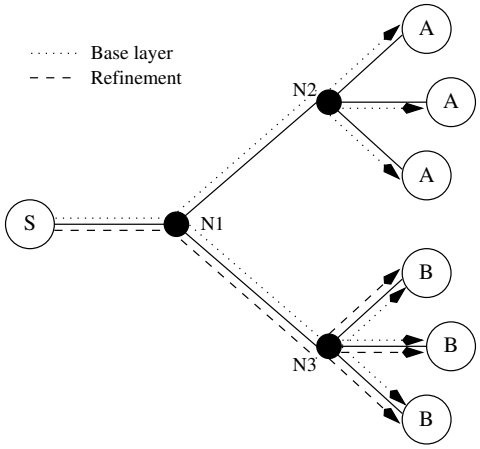


Figure 2: Example of layered encoding and transmission

Although layered encoding is possible with audio, the transmission bandwidth range is significantly more restricted in comparison to that available to video applications. This is definitely the case for sampled speech. There is none or very little improvement in intelligibility to be gained by sampling speech at full CD quality (44.1KHz stereo sampling) rather than a single channel sampled at 16KHz [10]. With music a wider range is available, from CD quality to a highly compressed low quality format.

Currently available speech codecs do not render themselves naturally to layering. It is possible to modify a scheme to split up the resulting compressed block

into a number of sub-blocks that can be separately decoded to provide increasing levels of quality. However, to achieve the same quality that the original not split up version of the codec provides, a larger number of bits per codeword is required [11].

An additional requirement from a layered encoding scheme, in order for it to be suitable for use with a network adaptation algorithm, is that there must be an exponential relationship between the bandwidth of different layers. Such an arrangement maximises support for network bandwidth adaptation while keeping the routing overheads due to the number of multicast groups used low. This requirement in combination with the problems listed above makes layered transmission unsuitable for real-time audio steams.

Even with a video stream depending on the application there is a target frame rate from which you cannot deviate. In video conferencing the entire range can be used, from very slow scan video to the full potential of the camera, but when watching a film full frame rate is required.

2.2 Simulcasting

With simulcasting a group of receivers can adapt to network conditions by having the sender transmit a new parallel stream and customise it to match their requirements. The new stream can use a different compression scheme to reduce the bandwidth required and employ some form of FEC to counter packet loss. This approach is likely to create congestion on links that are close to the sender, as all simulcasted streams will have to traverse them.

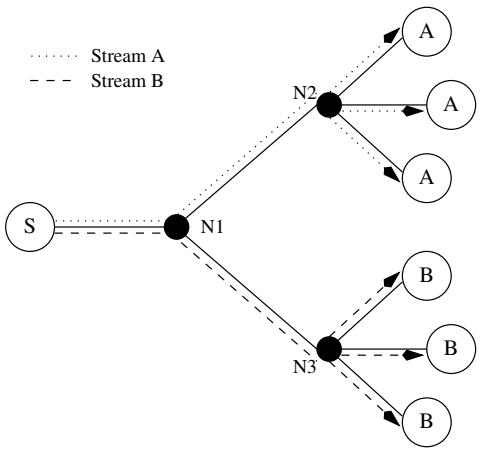


Figure 3: Example of simulcasting

The bandwidth utilisation advantage of layered encoding over simulcasting is illustrated in figures 2 and 3. Due to different bandwidth availability on links N1-N2 and N1-N3 groups of receivers A and B require different streams. With simulcasting the link between the source and N1 has to carry both the full bandwidth stream for receiver group B and a lower bandwidth stream for group A. Using layered encoding and transmission the link between the source and N1 does not carry duplicate redundant information. Ideally the sum in bandwidth of both layers will exactly cover the requirements of group B whereas the base layer will be customised for group A.

Simulcasting suffers from scalability problems because the sender is involved in the adaptation.

2.3 Retransmission Based Reliability

Proposals exist for integrating reliable multicast schemes into audio and video applications so that missing packets can be recovered from neighbours with better reception [12, 13, 14]. This is achieved by trading off quality for delay as any reliable multicast protocol has to request retransmission and wait for the repair. Although this may be acceptable in a real-time lecturing scenario it becomes less useful with interactive communication. An additional undesirable side effect is that the operation of the reliable protocol creates extra control traffic.

Maxemchuk et al [13] propose a hierarchy of retransmission servers positioned around expensive or over utilised links. The servers operate a NACK based reliable protocol between them and receivers use a similar scheme for requesting lost packets. Their proposal significantly improves reception quality but requires manual configuration of the retransmission servers.

In [14] Xu et al describe the STORM protocol that develops parent child relationships between participants of a multicast using an expanding ring search technique. Parents are chosen according to loss statistics so that they have a good chance of receiving packets their children are likely to request.

Streaming of stored data makes little sense unless browsing and selective playback is a requirement. For totally non real-time scenarios, a normal transport protocol and pre-fetch can be used to achieve perfect audio quality. TCP can be used in a single user scenario or a multicast congestion control protocol like RLC [15] for multiple recipients.

2.4 Statically Configured Transcoders

In many situations where a group of people with limited network resources want to participate in a high bandwidth multicast conference, the use of transcoders

is employed. A transcoder is an application that is placed at the far end of the low bandwidth link to down-convert a high bit rate stream so that it can fit through the link. Transcoders for video can reduce the frame rate and image quality and audio transcoders can re-encode the audio signal using a higher compression scheme. A feature of audio transcoding is that it adds minimal delay to the signal when relaying it as it can be done on a per packet bases. Apart from changing the bandwidth requirements of a stream a transcoder can also introduce or remove forward error correction information to counter packet loss.

In a multicast scenario a transcoder can be positioned on the sender's end of a problematic link to re-encode the stream to use lower bandwidth and add FEC information. The resulting stream can be re-multicast to a new address. If all receivers beyond that link tune to receive the new customised stream then there will be no bandwidth wasted as the original stream will no longer traverse the problematic link. This solution is static and has problems with the dynamic nature of Mbone multicast routing.

In [16] Pasquale et al propose the use of self-propagating filters over a dissemination tree. Leaf nodes specify to the node above them filters that can convert an incoming stream to match their requirements. When a non-leaf node has multiple output links with similar filters, the filter is propagated to a node higher up the tree. This scheme can achieve optimal network utilisation with minimal processing but requires full knowledge of the distribution tree topology and processing capabilities at each node.

3 Self Organised Transcoding

By combining the simulcasting, local repair and transcoding schemes we have developed a solution that does not suffer from the above problems. What is needed is a control scheme that automatically configures transcoders within the multicast tree to support branches with bad reception.

When a group of receivers detect loss caused by a congested link, an upstream receiver with better reception at the far end of the bottleneck link affecting the group needs to act as a transcoder and provide a customised version of the stream. This new stream will be multicast to a different address to which receivers affected by the bottleneck need to switch. To achieve this in a scalable way, the suffering receivers need to elect a representative which will attempt to locate an upstream receiver willing to serve them and co-ordinate the transcoding process (figure 4).

Ideally there should be no links carrying more than

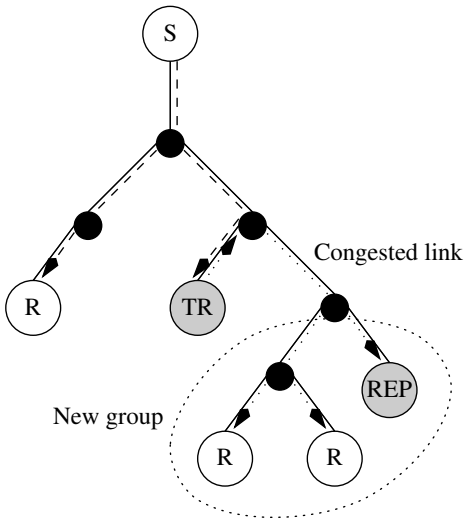


Figure 4: Transcoder requester and provider configuration around a congested link

one of the transcoded streams (including the original encoded stream). To achieve this the following conditions must hold:

- The transcoder servicing a sub-tree should be as closely located to the sub-tree as possible. The preferable location is at the end of the bottleneck link closer to the source.
- A group of receivers behind a bottleneck link has to be co-ordinated in its actions. All the receivers responding to congestion have to switch to a new transcoded stream at the same time.

To avoid the need of processing capabilities at nodes of the tree that do not have any receivers we co-locate the transcoders with active receivers and allow the users media tool to execute our protocol. Although this makes our proposal applicable on the currently available network infrastructure, it results in sub-optimal transcoder configurations. In the example of figure 4 the ideal placement for the transcoder would be on the network node above the requesting site. The current placement of the transcoder results in wasted resources for the transcoded stream out of the transcoding site. If the transcoding site was connected to the network through a shared medium link, like an ethernet, than both the incoming and outgoing stream would have to traverse the same link possibly causing congestion.

3.1 Transcoding Provider Selection

When a receiver detects packet loss, it schedules a transcoder request message. To avoid multiple re-

ceivers that see the same loss simultaneously sending a request and overloading the network, requests are multicast and sending them is delayed by a time proportional to the distance of the requester from the stream source plus an additional small random interval. If a receiver sees a request while it has one scheduled then the request is cancelled.

The request includes a description of the loss patterns observed. Receivers of the request that have better reception from the requester can offer to provide a transcoded stream. This is achieved in a similar manner to the request. The response message is scheduled to be multicast after an amount of time proportional to their distance from the requester plus a small random interval. The response message contains a description of the loss experienced by the offering receiver and its distance from the requester. On reception of the offer other receivers that have offers scheduled suppress their messages unless they can provide a better service.

The quality of service that a site offering a transcoded stream can provide is calculated as a function of the difference in loss rates observed by the requester and the offering site and the distance between the two. The quality is better for larger loss rate differences and smaller distances.

After a short timeout period, long enough for the offers from potential transcoders to arrive, the requesting receiver multicasts a transcoding initiation message. This message serves two purposes. It notifies the offering receivers of who is going to provide the transcoded stream and instructs all other members in the loss group to switch to the new stream.

As control messages are sent to the whole receiver group, while a transcoding negotiation is in progress, control messages from other receivers are suppressed.

3.1.1 Receiver Distance Calculation

The timer based-scheme described above is similar to that used in the SRM [17] reliable multicast protocol for retransmission requests and repairs. In SRM round-trip times (RTT) are used as distances between receivers and are calculated from timestamps in session messages. Reporting receivers include timestamps received from other receivers plus the amount of time elapsed between receiving the stamp and sending the report. Round-trip time estimates can thus be obtained. RTP uses the same scheme with timestamps in RTCP messages just for sources so that they can calculate the RTT to receivers. The SRM extension to obtain distance estimates between all the receivers does not scale for large sessions since every pair of receivers has to exchange at least three messages. Puneet et al [18] have developed a hierarchical self-organising scheme that elects top level receivers for different regions of the distribution tree that represent their region in distance calculation estimates. This scheme significantly improves the scalability of RTT calculation in SRM.

The requirement for background session messages to build distance information is removed if receivers use NTP [19] and have synchronised clocks. Although the level of deployment of NTP on current Mbone hosts is not very good, there is no reason why it should not be in use. With synchronised clocks a distance estimate can be obtained from a single timestamped message. A receiver of a request or an offer can calculate the distance from the sender by subtracting the timestamp in the message from the current time. This estimate may not be very accurate for the reverse distance from the receiver to the sender, as paths in the Internet are not always symmetric, but for our purposes it is good enough.

3.1.2 Multiple Offer Resolution

Depending on how the delay timers are set it is likely that the requester will receive more than one offer. Some of these may even not have originated from another receiver further up the delivery tree from the source but by a receiver on a side branch with a better link to the requester. A control protocol can be used in such cases to measure the performance of different links and decide on which one to use.

3.2 Receiver Group Control

In order to reduce the amount of traffic flowing through bottleneck links when a transcoded stream is initiated, all receivers behind that link should stop receiving the original stream. The transcoding initiation message provides the synchronisation needed to co-ordinate the switching action. Receivers of the initiation message decide individually whether they belong to the group and accordingly switch to receiving the new stream or continue without change.

The decision on whether a receiver belongs to the same loss group as the requester is based on correlated loss information between the two. The main principle behind this is that receivers behind the same congested link will miss the same packets and see similar loss patterns.

The current Mbone media tools implementing the RTP protocol provide periodic loss measurements in RTCP receiver reports. These reports are multicast so that all receivers see reports from other receivers. By correlating the variations in loss between what is

reported and what is observed locally, some grouping information can be derived. The problem with RTCP reports is that they become very infrequent as the size of the receiver group grows to maintain the amount of bandwidth used by control traffic small. Furthermore the period of time over which the loss is reported is not obvious. A guess at the reporting interval can be made as the last correctly received packet is given in the report. By monitoring the frequency of reports from each receiver we can figure out the period over which the report is referring to. The estimation process is complicated by lost report messages. To perform the correlation the loss observed locally over the same period has to be calculated. To achieve this, a history of arrival timestamps and sequence numbers of correctly received packets has to be maintained. The history has to be long enough to cover the maximum possible reporting interval. Apart from the excessive amount of storage this method requires, the results produced cannot be very accurate.

The need for background control messages to exchange locality information can be removed by including a description of the loss pattern observed by the requester in the transcoding initiation message. The loss description can be in the form of a bitmask representing which of the last transmitted packets were received and which not. The sequence number of the last packet in the bitmask can also be included. Other receivers can use the information in the bitmap to correlate the loss and decide if they belong to the group or not. This alleviates the problem of infrequent RTCP reports in large sessions, as even receivers that have not had a chance to send a report will know if they belong to the group.

To perform the correlation each receiver must maintain a history of received packets. The size of the state can be as little as one bit per packet as all we are interested in is whether a packet was received or not. The length of the history does not have to be much longer than the length of the loss bitmaps as the loss data in a received initiation message will always be recent.

A received bitmap can be compared with the local log to provide a loss proximity measure. Packets that are lost in both sites increase the likelihood that the receivers are located close to each other. Packets lost at one receiver but not at the other reduce the likelihood. The accuracy of the result can be improved by increasing the size of the bitmap at the expense of larger control messages. An optimisation would be to use a Huffman encoding for the bitmap. This way information about more packets can be packed in the same space in the message. The encoding method used can vary and be optimised for different loss rates.

The bitmap control scheme requires less state and processing and provides much more accurate results to the RTCP loss variation correlation.

3.2.1 Loss Bitmap Comparison

As the result of the loss bitmap comparison determines the stability of Self Organised Transcoding (SOT) it is crucial to minimise decision errors. Errors can have two outcomes:

- A receiver can decide to join a group it doesn't belong in, thus pulling the transcoded stream to some remote network location.
- A receiver that should join the transcoded group does not do so and the original data stream continues to flow down the bottleneck link.

Thus the decision process has to be as precise as possible and cannot be either over optimistic or under optimistic.

SOT uses three summary measures when comparing the loss bitmaps from two different receivers. These measures are the number of packets *lost in common*, the number of packets *lost at one receiver* but not the other and the overall *loss rate*. The overall loss rate is calculated as the average between the rates at the two sites as it is assumed that in order for two bitmaps to match the two rates have to be similar.

In order for the comparison to be useful, some information has to exist in the bitmaps. A pattern full of losses is not useful, as it will be the result of a broken link. Two such patterns although identical could be the results of two different broken links. The same holds for a pattern full of received packets. In contrast we can be almost certain that two identical bitmaps with a number of transitions between lost and received packets are the result of the same problematic link.

In some cases it may be desirable to accept a small number of not common losses in order to form groups of receivers that are affected by a common major bottleneck but also have smaller problems of their own. An example can be seen in figure 5. Receiver X can be grouped together with Y and have a single stream provided by Z. The stream can be customised to cover for the additional low loss on the link to X at the expense of having the link to Y slightly under-utilised. The alternative configuration would be to have Z transcode a stream for Y and then Y transcode a stream for X. There is a trade-off between the amount of separate losses that we should allow in the comparison to achieve such configurations and the probability of error in the decision.

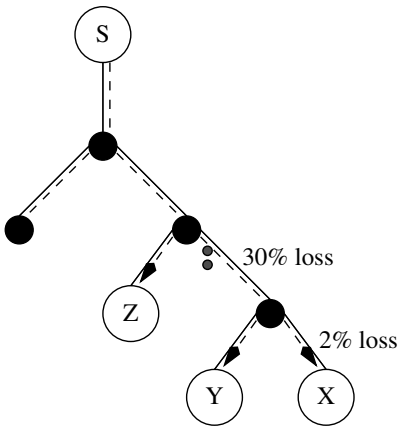


Figure 5: Receivers with slightly different loss affected by the same main bottleneck

3.3 Congestion Control

The transcoder initiation protocol results in a natural pairing between requester and transcoding provider on either side of a bottleneck link (figure 4). This provides a solution to the scaling problem of multicast congestion control. The original requester can represent the receiver group and provide the feedback needed to the transcoder provider to adapt to the available bandwidth of the bottleneck. This can be achieved on similar time-scales to TCP congestion control thus resulting in fair sharing of the network with non-multicast traffic. A possible design of a congestion control algorithm for real-time streams and results from simulation are presented in sections 4.2 and 4.5 respectively.

3.4 Membership Changes

The above discussion does not address start and end time issues. Receivers may join and leave at different times during a conference. When new receivers join they have to find out if their branch of the network is being serviced by a transcoder or the real source, and which address they should join to receive the traffic.

To achieve this a new receiver has to be able to probe existing receivers in its network neighbourhood in a scalable manner. With currently deployed multicast distribution protocols this can be achieved through the use of a time to live (TTL) based expanding ring search (ERS) algorithm. The idea is that the TTL field of the IP header can be used to limit the lifetime of multicast packets and restrict their distribution to a local part of the network. Using larger TTL values allows packets to live longer and reach further. A new receiver can send query messages to a control group starting with a low TTL and increasing until it receives a response.

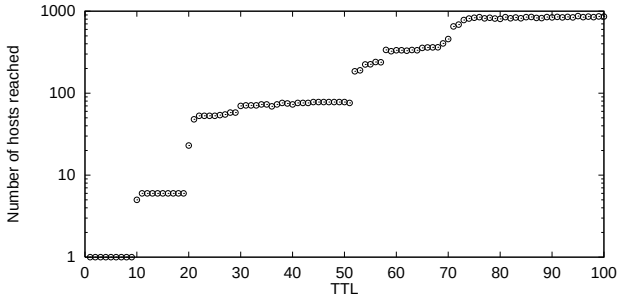


Figure 6: Number of hosts running sdr reachable for different TTL

Unfortunately increasing the TTL over the threshold necessary for the query packets to live beyond some router means that a whole new part of the network is reached and not a single potential responder. Figure 6 shows the number of hosts listening to the SDR [20] session announcement multicast address reachable from UCL for increasing TTL values. The measurements were collected in September 1997 using the multicast ping mechanism. Because the use of ping on multicast addresses causes an implosion of responses resulting in lost messages, the values in the graph are lower than the real ones especially for larger TTL values. The graph clearly shows that as the TTL exceeds various thresholds, large groups of receivers become reachable.

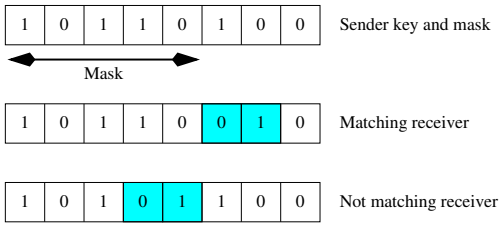


Figure 7: Using a sliding key to probe receiver group

To avoid an implosion of responses to a query message that suddenly reaches a large number of receivers, an additional mechanism is required to restrict the number of potential responders. A sliding key probing mechanism (introduced in [21]) can be employed. This operates by having the query sender and receivers each choose a random key. The sender's key is included in the request and only receivers with a matching key are allowed to respond. The number of matching keys can be controlled through the use of a mask that specifies the number of significant bits in the key that have to match (figure 7). Thus the number of responses to a request can be controlled by selecting an initial mask

length and then re-sending the request with a reduced mask size until a response is received.

An additional problem with the locality achieved through TTL scoped ring searches is that they cannot be constrained to work along the distribution tree from a given source. That would be desirable behaviour as we would be able to constrain configuration of transcoding groups along the original source distribution tree and have more predictable behaviour from our protocol. The use of subcasting, which has recently been proposed as a modification to multicast to support reliable multicast applications, can provide this functionality [22].

Receivers quitting the session are not a problem except in the cases of the requester or the last member of the group leaving. By having the requester periodically multicast alive messages after group formation to the formed group other members can detect when the requester has left. The messages are sent on a separate control address to prevent distribution to other session participants. On detection of departure, the remaining group members schedule a message to take over the role of the requester. The transmission is delayed proportionally to the distance from the transcoder so as to achieve election of the member closest to the bottleneck link. A transcoder can detect the departure of the last member of the group and stop transmission by the cease of congestion feedback information.

3.5 Topology Changes

Link and router outages although not very frequent are quite common in the Internet / Mbone [23]. As a result of an outage the multicast routing to some members of a session being serviced by a transcoder may change resulting in a non-optimal or even problematic configuration (figure 8). To recover from such situations SOT needs to periodically repeat the initiation protocol. The task of doing this can be left up to the original requester.

The resolution of congestion problems that initially caused group formation or the appearance of a new bottleneck splitting an existing group in half also affect group dynamics. The first effect can be addressed by having the transcoder dissolve a group that has been following the transmission rate of the original sender for some period of time. The introduction of a new bottleneck will cause some members of the loss group to observe different loss patterns to those of the requester. By introducing a loss bitmap in the alive messages sent by the requester, this can be detected and the affected members can rerun the initiation protocol.

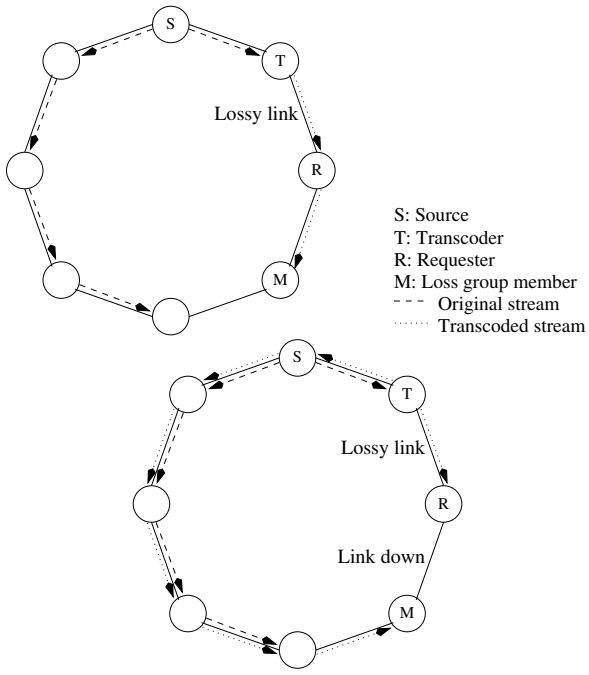


Figure 8: Possible effect of a topology change on a transcoder setup (before and after)

3.6 Multicast Address Allocation

Multicast distribution trees vary for different sources in a session so SOT adaptation has to be per source. With currently deployed multicast protocols there is no way a receiver can express interest in a particular source. Instead subscription is per group and every sender sending to this group is received. For SOT to work in this environment each sending participant has to use a separate multicast address to transmit data. In addition each new transcoder instantiation has to transmit on a new unused address. In sessions with a large number of participants this can be a problem. Typically very large sessions are lecture based where only a few participants transmit data and the majority are only spectators which somewhat alleviates the problem.

The real solution to this problem is the deployment of the Internet Group Management Protocol version 3 (IGMPv3) [24] which is currently under development by the IETF IDMR working group. IGMPv3 supports expression of interest in particular sources for each multicast address joined. This reduces the number of multicast addresses required by SOT to the maximum number of transcoded streams forwarded by any node.

4 Simulation

As part of the development of some of the ideas in SOT and in order to evaluate its performance, we implemented the protocol in version 2.1 of the VINT network simulator ns [7]. Ns is an event driven packet-level simulator. Within ns there are several multicast protocol implementations. We chose to use the dense mode (DM) version as it behaves similarly to what is currently available on the Mbone. We extended the implementation by adding source-specific group membership control, which is expected to be available on the Mbone with the deployment of IGMPv3 [24]. The Self Organised Transcoding protocol was implemented as an extension to ns using C++ and otcl. The protocol implementation and simulation scripts are available upon request from the authors.

4.1 SOT Implementation

This section describes the design of the SOT transcoder initiation implementation in ns. In the initiation stage SOT uses the following three messages:

request: Sent by the loss group representative to locate possible transcoders. This message identifies the requester and includes the observed loss rate so that sites willing to offer a transcoded stream can compare their reception and respond only if it is better.

response: Sent by receivers that have received a request, have better reception to the requester and are willing to provide a stream. The locally observed loss rate is included so that the requester can select the best transcoder if multiple offers are received.

command: After responses have been collected by a requester, a transcoder is selected and this message sent to instruct the transcoder to initiate the new stream and other loss group members to switch streams. The message includes a loss bitmap for other receivers to compare against and decide if they belong to the group or not.

In addition to receiving the above messages there are two more events that can occur during protocol operation. The first is a *loss report* that is triggered by the reception of a data packet. The second is a *timeout* from the internal protocol timer.

These events cause each member of a SOT session to be in one of the following states:

Idle: This is the initial state.

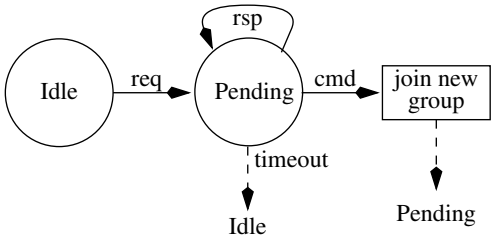


Figure 11: Transcoding initiation state transition diagram (loss group member)

Schedule request: In order to locate a transcoder after loss has been reported a request message has to be sent. To avoid message implosion, these messages are randomly delayed as described earlier in section 3. A timeout event is scheduled and the receiver waits in this state before sending the request.

Wait for response: After a request has been sent, a requester waits in this state for offers from transcoders. A timer is set to retransmit the request if no responses are received for a predefined period of time. If at the expiration of the timer an offer has been received a command message is sent out to initiate the transcoded stream and instruct other receivers in the loss group to receive it.

Schedule response: A site willing to offer its services as a transcoder waits in this state for a timeout before sending a response. Reception of a better offer from another transcoder before the timeout cancels the scheduled response.

Pending: As SOT messages are multicast to the entire receiver group, in order to reduce the amount of bandwidth consumed at any instant the messages have to be spread out in time. To this end an attempt to have only one request in progress is made. This is achieved by having all other traffic cancelled and the senders back off when a setup with a worse loss problem is seen to be in progress. Backing off is implemented by waiting in this state for a timeout that returns the receiver to the idle state.

The transition diagrams for the states and events listed above are shown in figures 9, 10 and 11 for the requester, the site offering to transcode and a loss group member respectively.

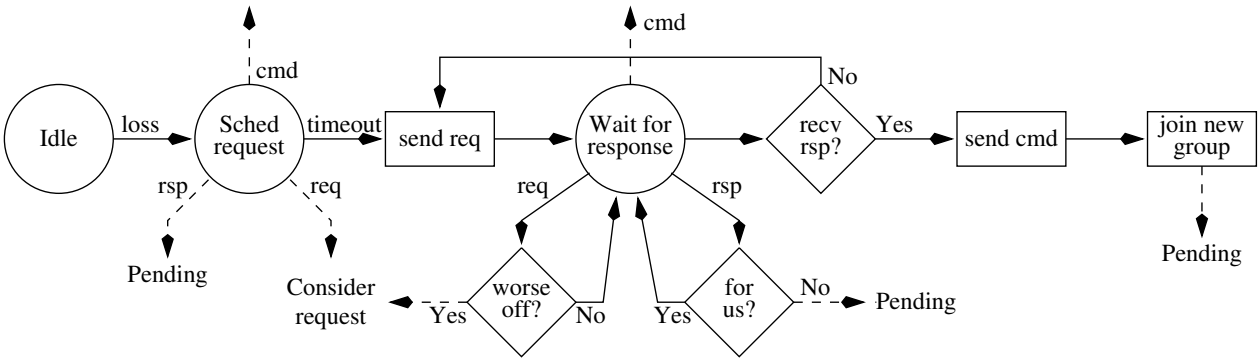


Figure 9: Transcoding initiation state transition diagram (requester)

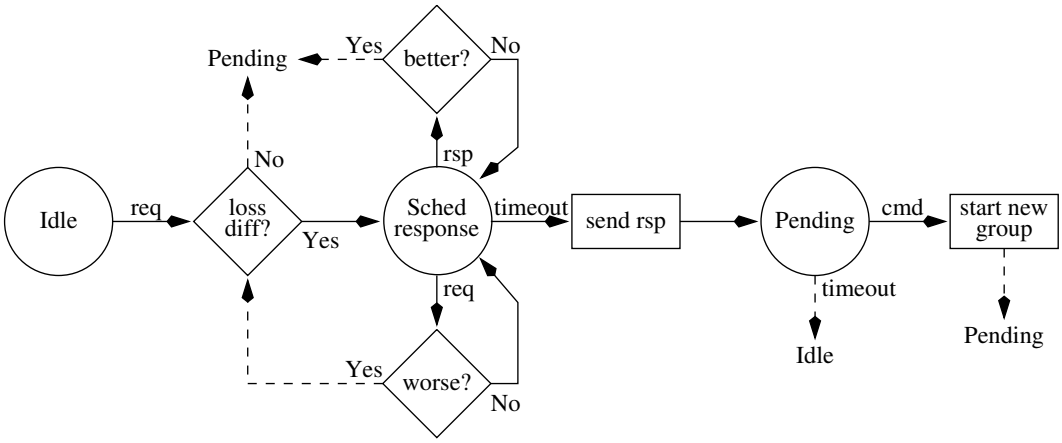


Figure 10: Transcoding initiation state transition diagram (transcoding offer)

4.2 Congestion Control Implementation

After the initiation stage is complete, the requester provides feedback to the transcoder concerning the bottleneck behaviour. The information consists of loss / no loss signals. The transcoder uses this information to adapt the bandwidth of the transcoded stream. Although the adaptation algorithm is independent from the operation of SOT, we implemented a simple version for the purposes of our simulations. The implemented algorithm tries to behave in a manner similar to the congestion control algorithm in TCP [25] by halving the stream bandwidth when loss is detected and linearly increasing the bandwidth when no loss is signalled. When transcoders start the initial bandwidth of the transcoded stream is set to a very low rate thus performing the equivalent of a slow start.

Bandwidth selection is achieved by varying the transcoded packet size. This is the behaviour that would be expected by an audio transcoder when selecting a different encoding scheme for the outgoing stream. The number and frequency of outgoing packets is the same as that of incoming ones. This is true of most transcoding techniques as each packet corresponds to a specific time interval. In video transcoding there are two different options to control the transmission rate. The simplest solution is to reduce the frame rate which will result in a smaller number of packets. A better approach is to reduce the quality of the encoded image resulting in the same number of smaller packets. Codecs available in current Mbone tools support image quality selection.

When stream bandwidth is increased in response to a period with no loss we are performing an experiment to see if the bottleneck can accommodate some additional traffic. If the link is full this will result in congestion and some packets will be dropped. To avoid degradation in perceived quality due to the loss, the additional bandwidth can be used to carry FEC redundant information [4] for a short period after the bandwidth

increase. The switch to a higher quality encoding without redundancy can be postponed until we feel that the link can take the new traffic.

4.3 Simulation Metrics and Parameters

SOT was simulated on a number of simple network topologies, which were designed to include specific problematic configurations (figure 12), and on larger random topologies that were created with the assistance of topology generators (figure 13).

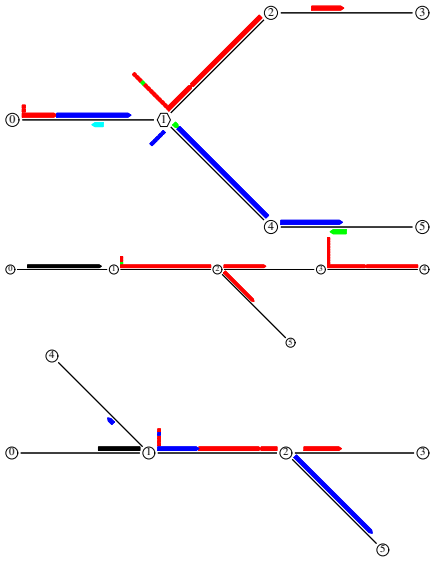


Figure 12: SOT simulation on specific problem topologies

Tests included sparse and dense topologies, bottlenecks in series and introduction of additional non-real-time TCP flows during SOT sessions. In all situations the simulations performed as expected forming groups and setting up transcoders around the problem links.

In the rest of this section simulation results evaluating the performance of SOT and the congestion control algorithm are presented. The experiments performed were designed to measure the time-scales over which SOT reacts to a congestion problem, the amount of extra bandwidth used by SOT control messages and the level of network friendliness achieved by the congestion control algorithm.

In all the simulations packets are sent with a frequency of 50 pps (20 ms duration) simulating an audio source[1]. Transcoded packet sizes start from 32 bytes

[1] The simulation in this paper is tailored for multicast audio however the proposed scheme can be applied equally well to other types of real-time multicast streams including video.

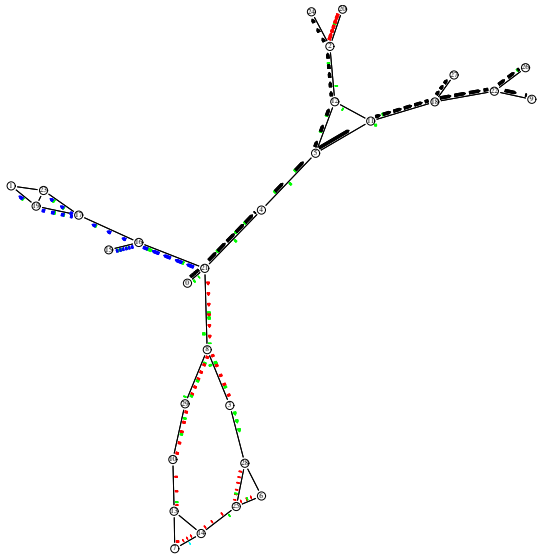


Figure 13: SOT simulation on randomly created topologies

and are increased in steps of 32 bytes up to the incoming stream bandwidth. Although the number of available coding algorithms is limited and hence the number of possible packet sizes, with the combination of redundant information all the sizes used in the simulation should be possible in a real audio tool. For video transcoding things are simpler as image quality selection provides a fuller transmission range.

4.4 Transcoder Initiation Evaluation

The main goal of the transcoding initiation algorithm is to quickly respond to a congestion problem by setting up a transcoder. The experiments performed aim at measuring the amount of control traffic introduced to the network during initiation and the delay between detecting a problem and completing initiation.

As transcoder request messages are also used to suppress further requests from other members of a loss group, only one request can be in progress at any one time. In the simulations we resolve conflicts by giving priority to the request reporting the highest loss. Back-off of competing request groups is achieved by introducing a random delay in the message that instructs other receivers how long they should wait before retrying. This resolves a request synchronisation problem and reduces convergence time.

To measure the back-off algorithm performance we used sessions with varying number of bottlenecks. The designed network topology is shown in figure 14. The session source is positioned in the centre of a star topol-

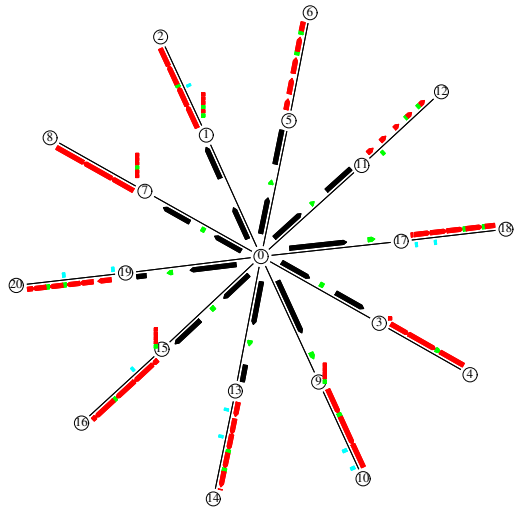


Figure 14: SOT simulation on session with 10 bottlenecks

ogy. Each branch contains two receivers connected in series. The connection from the sender to the first receiver is a high bandwidth link (0.5 Mb/s) with delay varying between 20ms and 40ms. The connection between the first and second receiver is a lower bandwidth link (150Kb/s to 200Kb/s) with longer delay varying between 50ms and 100ms. The session bandwidth is set at 256Kb/s and as a result transcoders need to be set up on all branches between the first and second receivers.

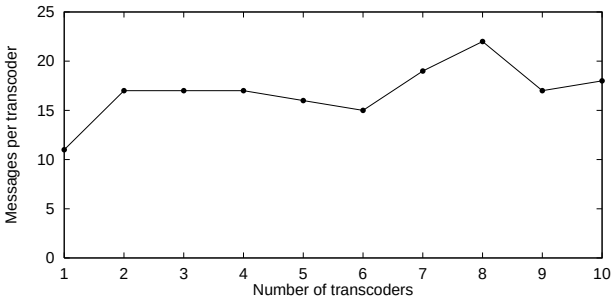


Figure 15: Number of SOT control messages sent during transcoder initiation for different scale simulations

The experiment was repeated with scale varying from one to ten branches. The number of messages that were sent during initiation over the number of transcoders that were set up is shown in figure 15 for different session sizes. The messages per transcoder is roughly constant showing that the back-off mechanism does not get stuck in loops as a result of request conflicts.

In all the simulations the source starts sending RTP

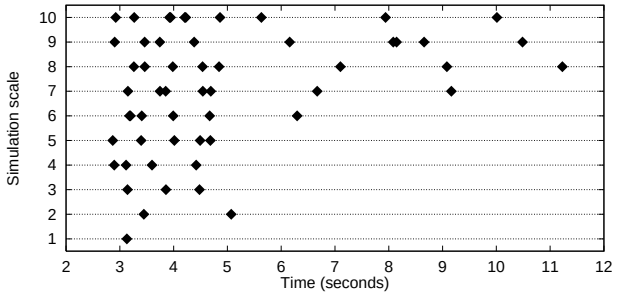


Figure 16: Transcoder initiation times for different scale simulations

packets at time 1.5 seconds and the first packets are dropped when the queues fill up somewhere around time 2 seconds. From then on the transcoder initiation process begins. Figure 16 shows the completion times for the individual transcoder initiations for each simulation. It can be seen that transcoders are set up more densely at the beginning of the simulation and then initiations reach a steadier rate. The initial concentration can be attributed to control packet loss. The links are very congested before the transcoders start and as a result request messages may not reach other requesters behind different bottleneck links thus allowing more than one initiation to progress in parallel. As the simulation progresses and congestion relaxes, the back-off algorithm works better and reduces conflicts by spreading out initiation times.

A reduction of initiation times can be achieved by removing the back-off algorithm and allowing multiple initiations to take place simultaneously. To ensure that other members of the same loss group are still suppressed when one member sends a request, a loss bitmap has to be included in the message. Receivers of the request can then decide if it refers to their loss problem or not. The problem with allowing simultaneous initiations is use of excessive bandwidth with control messages at any time. This is alleviated by the following factors:

- Very few messages need to be exchanged during initiation
- It is not important if control packets get lost and do not reach the rest of the net, as initiations should be localised close to the problem
- The number of initiations is proportional to the number of bottlenecks and not session size, although they may have a close relation depending on geographical coverage

Persistent Responses: Transcoder response messages serve two purposes. They suppress additional offers and double as a response to the requester. In order to reach the requester the response has to pass through a congested link. To improve the chances of reaching the requester the offering site can follow the multicast response with a small number of unicast copies of the message addressed to the requester. These should be spaced in time by a small random interval.

Initiation Congestion Reduction: When a new transcoder starts two new congestion problems can arise. Prunes from the receivers of the new transcoded group for the original data take time to be forwarded to the transcoding site and hence the original group will still be forwarded for some time causing more congestion on the problematic link. This can be partially avoided by starting the transcoder at very low bandwidth (like a slow-start). The second problem is that receivers of the original group that are not interested in the new multicast traffic need time to prune it. In the meantime the additional traffic may cause problems in previously non-congested links. The slow-start will help here as well. An additional measure can be to have transcoding offers followed by a single packet in the intended new group. In this way receivers that are not interested in the new traffic can have a head start with pruning.

4.5 Congestion Control Algorithm Performance

In order to measure the performance of the congestion control algorithm two different experiments were performed. In the first experiment ten SOT sessions are created that share the same bottleneck link. This arrangement is shown in the topology of figure 17. Each session contains two members, the sender and one receiver. All senders are positioned on nodes on one side of the bottleneck link and all receivers on the other. The bandwidth of all sessions is set at 256kb/s. The available bandwidth on the bottleneck link is set at 1Mb. The aim of the experiment is to show that transcoders are set up and that they fairly share the bandwidth of the bottleneck link.

Figure 18 shows the packet size variation for each of the transcoders during the simulation. There is considerable variation due to the adaptation process of slowly increasing and then halving the bandwidth but all the sessions oscillate around roughly the same packet size. This is better illustrated in figure 19, which shows the average packet size and the standard variation during adaptation for each of the transcoders.

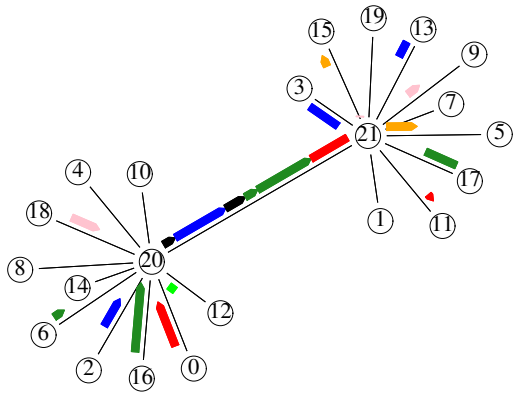


Figure 17: Multiple SOT sessions sharing a bottleneck

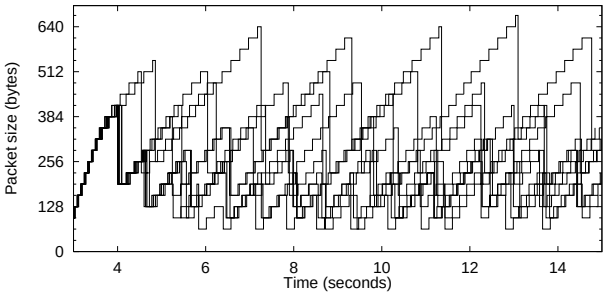


Figure 18: Packet size variation for each transcoder during adaptation

The percentage of simulation time that the transcoders spend sending each packet size can be seen in figure 20. This graph is cumulative for all ten sessions. Although during the adaptation some extreme small and large packet sizes are reached, for most of the simulation a small number of packet sizes is used that is close to the optimum.

The cycle of variation between packet sizes will require switching between different codecs and levels of redundancy. Care needs to be taken in order to limit the impact of this variation on the user. With audio the changes will be very hard to perceive except for the cases where really low bandwidth codecs (like LPC) are used. However, the amount of time spent by the adaptation algorithm in packet sizes requiring such codecs will be very small (a few packets). Perception experiments carried out at UCL have evaluated the impact of mixing small intervals of LPC synthetic speech with toll-quality speech for the purposes of audio redundancy [26]. Results show that for small intervals (around 40ms) intelligibility of speech does not deteriorate whereas for intervals larger than 80ms there is a slight deterioration.

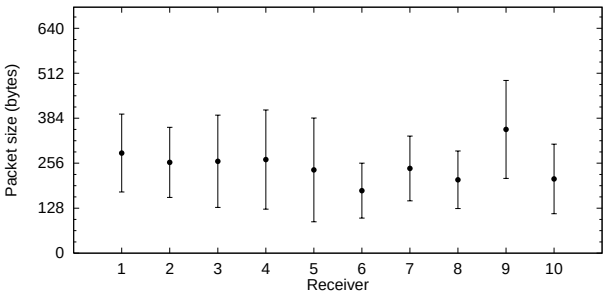


Figure 19: Average and standard deviation of packet size for each transcoder

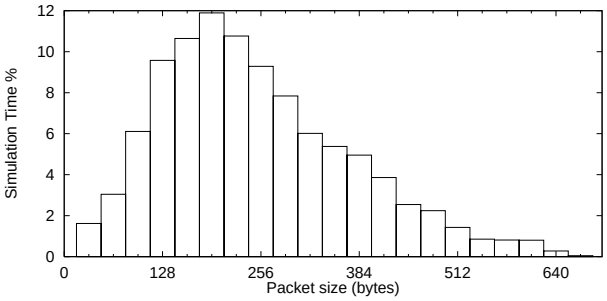


Figure 20: Percentage of simulation time that each packet size was used by all the transcoders

The second experiment shows fair sharing of the bottleneck bandwidth with TCP. The same topology was used and the bottleneck link shared between four SOT and four TCP sessions. The default ns drop-tail queueing strategy was used on the bottleneck link. Using this strategy, the queue capacity is measured in number of packets and packet size makes no difference. Figure 21 shows the total bottleneck bandwidth used by SOT and TCP for each second of the simulation after the SOT transcoders have started. The two curves are very evenly matched. Jain's index [27] with each individual flow as a user gives 99.5% fairness.

By modifying the simulation to use drop-tail queues that take into account packet size and have limited buffer space, the fairness drops to 96.8% The reason for the change is that SOT packets are smaller than TCP packets and hence have a better chance of fitting in a full queue.

5 Conclusions

This paper has presented a solution to the problem of multicast congestion control for real-time streams. The proposed scheme uses self-organisation to form groups

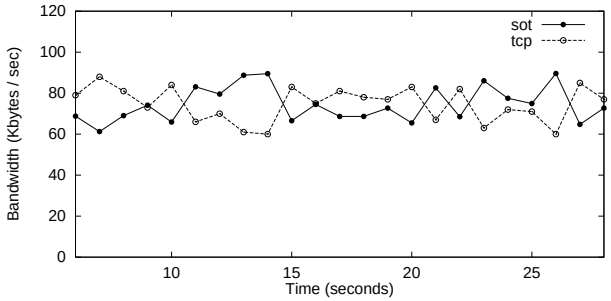


Figure 21: Bandwidth use of 4 SOT and 4 TCP sessions sharing a bottleneck link

out of co-located receivers with bad reception and provides local repair through the use of transcoders. The receiver driven nature of the protocol ensures high scalability and applicability to large Mbone sessions. An evaluation of the proposed scheme has been conducted through simulation. The results are encouraging as they show that the protocol is viable.

The simple congestion control implementation indicates that fair sharing of bottleneck links between real-time multicast traffic and traditional unicast traffic is possible. The simulation shows that adaptation is possible even with the limited bandwidth of audio communication. The significantly broader range available with video can provide additional freedom to the adaptation algorithm.

6 Acknowledgements

We would like to thank Mark Handley, Orion Hodson and Lorenzo Vicisano for their contribution through many useful discussions at UCL and Luigi Rizzo for his comments on drafts of the paper.

References

[1] Ramachandran Ramjee, Jim Kurose, Don Towsley, and Henning Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *Conference on Computer Communications (IEEE Infocom)*, Toronto, Canada, June 1994.

[2] Isidor Kouvelas, Vicky Hardman, and Anna Watson. Lip synchronisation for use over the internet: Analysis and implementation. In *IEEE Globecom '96*, London, UK, November 1996.

[3] Colin Perkins, Vicky Hardman, Isidor Kouvelas, and Angela Sasse. Multicast audio: The next generation. In *International Networking Conference (INET)*, Kuala Lumpur, Malaysia, June 1997.

[4] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, Mark Handley, Vicky Hardman, Orion Hodson, Isidor Kouvelas, Colin Perkins, and Andres Vega-Garcia. RTP payload for redundant audio data. Request for comments (Proposed Standard) RFC 2198, Internet Engineering Task Force, Audio / Video Transport Working Group, September 1997.

[5] Mark Handley. An examination of Mbone performance. Research Report ISI-RR-97-450, USC/ISI, April 1997.

[6] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *SIGCOMM*, pages 117–130, Stanfort, CA, August 1996. ACM.

[7] Steve McCanne et al. UCB/LBNL/VINT network simulator - ns (version 2). Software and documentation available from http://www-mash.cs.berkeley.edu/ns/, November 1997.

[8] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. RTP: a transport protocol for real-time applications. Request for comments (Proposed Standard) RFC 1889, Internet Engineering Task Force, January 1996.

[9] Henning Schulzrinne. RTP profile for audio and video conferences with minimal control. Request for comments (Proposed Standard) RFC 1890, Internet Engineering Task Force, January 1996.

[10] L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice Hall, 1978.

[11] Mostafa Hashem Sherif, Duane O. Bowker, Guido Bertocci, Bruce A. Orford, and Gonzalo A. Mariano. Overview and performance of CCITT/ANSI embedded ADPCM algorithms. *IEEE Transactions on Communications*, 41(2):391–398, February 1993.

[12] S. Pejhan, M. Schwartz, and D. Anastassiou. Error control using retransmission schemes in multicast transport protocols for real-time media. *IEEE/ACM Transcations on Networking*, 4(3):413–327, June 1996.

[13] N.F. Maxemchuk, K. Padmanabhan, and S. Lo. A cooperative packet recovery protocol for multicast video. In *International Conference on Network Protocols*, Atlanta, Georgia, October 1997.

[14] X. Rex Xu, Andrew C. Myers, Hui Zhang, and Raj Yavatkar. Resilient multicast support for continuous-media applications. In *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, St. Louis, Missouri, May 1997.

[15] Lorenzo Vicisano, Luigi Rizzo, and Jon Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Conference on Computer Communications (IEEE Infocom)*, San Francisco, California, March 1998.

[16] Joseph C. Pasquale, George C. Polyzos, Eric W. Anderson, and Vachaspathi P. Kompella. Filter propagation in dissemination trees: Trading off bandwidth and processing in continuous media networks. In *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 259–268, Lancaster, U.K, November 1993.

[17] S. Floyd, V. Jacobson, S. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transcations on Networking*, 5(6):784–803, December 1997. An earlier version of this paper appeared in *ACM SIGCOMM 95*, August 1995, pp. 342-356.

[18] Puneet Sharma, Deborah Estrin, Sally Floyd, and Lixia Zhang. Scalable session messages in SRM using self-configuration. submitted to Sigcomm '98, February 1998.

[19] D. Mills. Simple Network Time Protocol (SNTP) version 4 for IPv4, IPv6 and OSI. Request for comments (Proposed Standard) RFC 2030, Internet Engineering Task Force, October 1996. Obsoletes RFC 1769.

[20] Mark Handley. SAP: Session Announcement Protocol. Internet Draft, Internet Engineering Task Force, November 1996. Work in progress.

[21] Jean-Chrysostome Bolot, Thierry Turletti, and Ian Wakeman. Scalable feedback control for multicast video distribution in the internet. In *SIGCOMM*, London, UK, August 1994. ACM.

[22] Christos Papadopoulos, Guru Parulkar, and George Varghese. An error control scheme for large-scale multicast applications. In *Conference on Computer Communications (IEEE Infocom)*, San Francisco, California, March 1998.

[23] Vern Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transcations on Networking*, 5(5):601–615, October 1997.

[24] Brad Cain, Steve Deering, and Ajit Thyagarajan. Internet Group Management Protocol, version 3. Technical report, Internet Engineering Task Force, November 1997. Internet Draft (work in progress).

[25] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Request for comments (Proposed Standard) RFC 2001, Internet Engineering Task Force, January 1997.

[26] Vicky Hardman, Martina Angela Sasse, Mark Handley, and Anna Watson. Reliable audio for use over the Internet. In *International Networking Conference (INET)*, September 1995.

[27] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report DEC Research Report TR-301, DEC, September 1984.

MPEG-2 Video Services over Packet Networks: Joint Effect of Encoding Rate and Data Loss on User-Oriented QoS

Olivier Verscheure, Pascal Frossard and Maher Hamdi
Institute for computer Communications and Applications
Swiss Federal Institute of Technology

Abstract

We address the problem of video quality prediction and control for high resolution video transmitted over lossy packet networks. We analyze how the user-perceived quality is related to the average encoding bitrate for VBR MPEG-2 video. We then show why simple distortion metrics (e.g., PSNR) may lead to inconsistent interpretations. Furthermore, for a given coder setup, we analyze the effect of packet loss on the user-level quality. We then demonstrate that, when jointly studying the impact of coding bit rate and packet loss, the reachable quality is upperbound and exhibits one optimal coding rate for a given packet loss ratio.

1 Introduction

The choice of the compression algorithm depends on the available bandwidth or storage capacity and the features required by the application. The MPEG-2 [1] standard [1], a truly integrated audio-visual standard developed by the International Organization for Standards (ISO), is capable of compressing NTSC or PAL video into an average bit rate of 3 to 6 Mbits/s with a quality comparable to analog CATV [2]. However, much work remains to be done to optimize these audiovisual applications as the users expect an adequate audiovisual quality at the lowest possible cost. In the case of video transmission over packet networks, the User-oriented Quality of Service (U-QoS) results both from the video encoding quality and the degradations due to packet loss, delay and delay jitter during the transmission. The most economic offering can thus only be found by considering the entire system and not by optimization of individual system components in isolation [6].

Our work focuses on video quality prediction and control for high resolution packet video transmitted over lossy networks.

[1] MPEG stands for Moving Picture Experts Group

This paper is organized as follows: In Sec. 2, we introduce the MPEG-2 video and system standards. We briefly describe the impact of data loss on the reconstructed video sequence. Finally, useful video quality metrics are described, among them is the MPQM which is based on a vision model. The study of the impact of MPEG-2 rate and data loss on quality is the subject of Sec. 3. Section 4 deals with the joint impact analysis of both MPEG-2 rate and data loss on video quality. Concluding remarks are given in Sec. 5.

2 MPEG-2 over Packet Networks

2.1 MPEG-2 Backgrounder

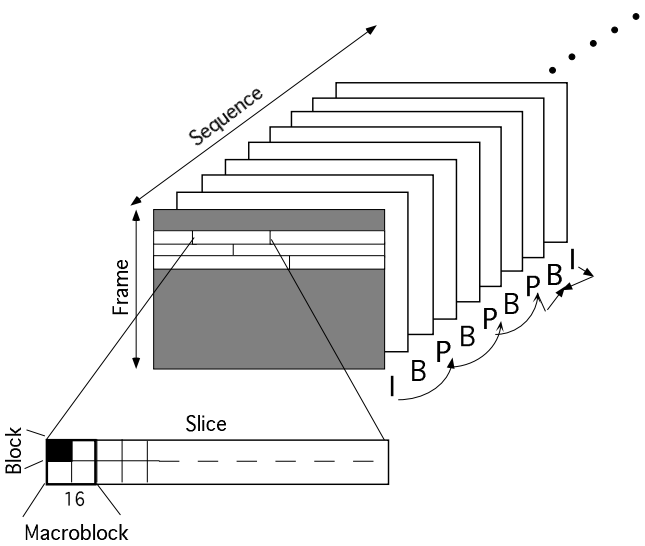


Figure 1. MPEG-2 video structure.

An MPEG-2 video stream is hierarchically structured as illustrated in Fig. 1. The stream consists of a sequence composed of several pictures. The MPEG-2 video standard defines three different types of pictures : intra-coded (I-), pre-

dicted (P-) and bidirectional (B-) pictures. The use of these three picture types allows MPEG-2 to be robust (I-pictures provide error propagation reset points) and efficient (B- and P-pictures allow a good overall compression ratio). Each picture is composed of slices which are, by definition, a series of macroblocks. Each macroblock (16×16 pixels) contains 4 blocks (8×8 pixels) of luminance and 2, 4 or 8 blocks of chrominance depending on the chroma format. Motion estimation is performed on macroblocks while the DCT [2] is calculated on blocks. The resulting DCT coefficients are quantized and variable length coded. The quantizer comes from the multiplication of a Quantizer Scale, MQUANT, and the corresponding element of a Quantizer Matrix. In general, the higher the MQUANT value, the lower the bit rate but also the lower the quality (well-known from the rate-distorsion theory).

Before being transmitted, a video stream goes through the MPEG-2 Transport Stream (TS) layer. Basically, the stream is first segmented into variable-length Packetized Elementary Stream packets and then subdivided into fixed-length TS packets. It is worth noting that a non-encoded header (i.e., syntactic information) is inserted before each of the following information elements: sequence, Group of Pictures (GoP), picture, slice, TS and PES. In general, when a header is damaged, the underlying information is lost.

2.2 MPEG-2 Sensitivity to Data Loss

In an MPEG-2 video stream, data loss reduces quality depending strongly on the type of the lost information. Losses of syntactic data, such as headers and system information, affect the quality differently than losses of semantic data such as pure video information (e.g., motion vectors, DCT coefficients, etc.). Furthermore, the quality reduction depends on the location of the lost semantic data due, not only to the predictive structure of MPEG-2 video coded streams, but also to the visual relevance of the data.

Figure 2 illustrates how network losses map onto visual information losses in different types of pictures. Data loss spreads within a single picture up to the next resynchronization point (e.g., picture or slice headers) mainly due to the use of differential coding, run-length coding and variable length coding. This is referred to as spatial propagation and may damage any type of picture. When loss occurs in a reference picture (intra-coded or predictive frame), the damaged macroblocks will affect the non intra-coded macroblocks in subsequent frame(s), which reference the errored macroblocks. This is known as temporal propagation and is due to inter-frame predictions.

However, the error visibility may be dramatically reduced by means of error concealment techniques. These error concealment algorithms include, for example, spatial

[2] DCT stands for Discrete Cosine Transform

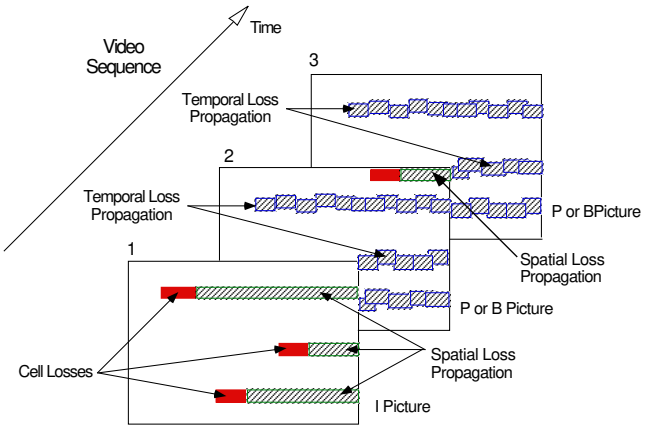


Figure 2. Data loss propagation in MPEG-2 video streams.

interpolation, temporal interpolation and early resynchronization techniques. The MPEG-2 standard proposes an elementary error concealment algorithm based on motion compensated techniques. Mainly, it estimates the motion vectors for the lost macroblock by using the motion vectors of neighbouring macroblocks in the affected picture (provided these have not also been lost). This improves the concealment in moving picture areas. However, there is an obvious problem with errors in macroblocks whose neighbouring macroblocks are intra-coded, because there are ordinarily no motion vectors associated with them. To circumvent this problem, the encoding process can be extended to include motion vectors for intra macroblocks [3].

Error concealment techniques may, in general, efficiently decrease the sensitivity to data loss. However, none of these techniques is perfect. Data loss may still involve annoying degradation in the decoded video.

2.3 Video Quality Metrics

Traditionally, the quality metric used for audiovisual signals is the Peak Signal to Noise Ratio (PSNR). However, many works have shown that such a metric was poorly correlated with human perception. Indeed, the PSNR metric does not take the visual masking phenomenon into consideration. In other words, every single errored pixel contributes to the decrease of the PSNR, even if this error is not perceived. Hence, recent research has addressed the issue of video quality assessment by means of human correlated metrics.

Several studies have shown that a correct estimation of subjective quality has to incorporate some modeling of the

[3] Some MPEG-2 encoder chips automatically produce concealment motion vectors for all intra-coded macroblocks.

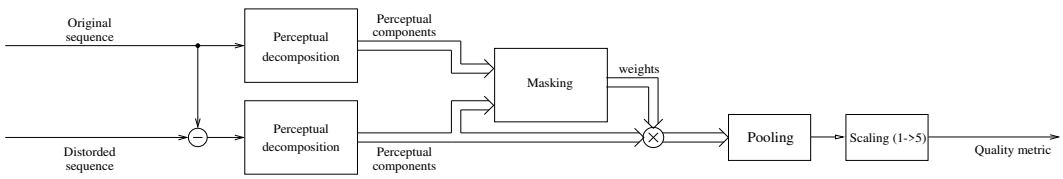


Figure 3. Moving Pictures Quality Metric (MPQM) block diagram

Human Visual System [4]. A spatio-temporal model of human vision has been developed for the assessment of video coding quality [10, 11]. It was then used to build a computational quality metric for moving pictures [11] which proved to behave consistently with human judgments. Basically, the metric, termed Moving Pictures Quality Metric (MPQM), first decomposes an original sequence and a distorted version of it into perceptual channels. A channel-based distortion measure is then computed, accounting for contrast sensitivity and masking. Finally, the data is pooled over all the channels to compute the quality rating which is then scaled from 1 to 5 [8] (see Fig. 3). This quality scale is used for subjective testing in the engineering community (see Table 1).

Rating	Impairment	Quality
5	Imperceptible	Excellent
4	Perceptible, not annoying	Good
3	Slightly annoying	Fair
2	Annoying	Poor
1	Very annoying	Bad

Table 1. Quality scale that is generally used for subjective testing in the engineering community

3 Impact of MPEG-2 Rate and Data Loss on Quality

In this section, we first describe the experimental setup used throughout this work. We then study how the video quality behaves according to the quantizer scale factor (MQUANT) in an MPEG-2 OL-VBR [4] encoding scheme. We also analyze how the average encoding bit rate is affected by this MQUANT. We then derive a mathematical relation modeling the impact of the average variable rate on the video encoding quality. Finally, we study how the video quality decreases when the data loss ratio is increased, for a fixed average encoding bit rate.

3.1 Experimental setup

The experimental testbed is composed of four parts (see Fig. 4):

- An MPEG-2 software encoder, which is composed of an open-loop VBR TM5 video encoder [3] and a transport stream encoder. Four 100 frame-long sequences conforming to the ITU-T 601 format were used (i.e., Football, News, Ski and Barcelona). All these sequences are very different in terms of spatial and temporal complexities. They were encoded, as interlaced video, with a structure of 12 images per GOP and 2 B-pictures between every reference picture in an OL-VBR mode. The following MQUANTs were used: 6, 10, 16, 20, 28, 32, 36, 40 and 48. Motion vectors were generated for all intra-coded macroblocks. It is to be noted that the OL-VBR encoding quality is not affected at all when introducing these extra motion vectors. Before being transmitted, each MPEG-2 video bitstream was encapsulated into 18800-bytes length Packetized Elementary Stream (PES) packets and divided into fixed length Transport Stream (TS) packets by the MPEG-2 system encoder.

- A model-based data loss generator was used to simulate packet network losses. For this purpose, we used a two-state Markovian model (Gilbert model [5], see Fig. 5).

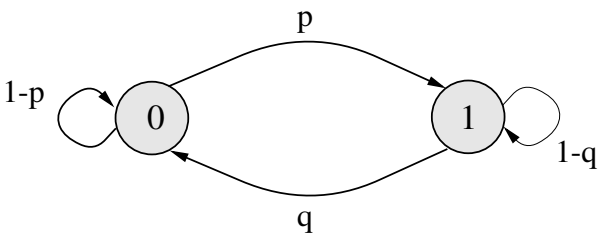


Figure 5. Two-State Makov Chain: Gilbert Model.

[4] OL-VBR stands for Open-Loop Variable Bit Rate (constant quantizer scale over the whole sequence).

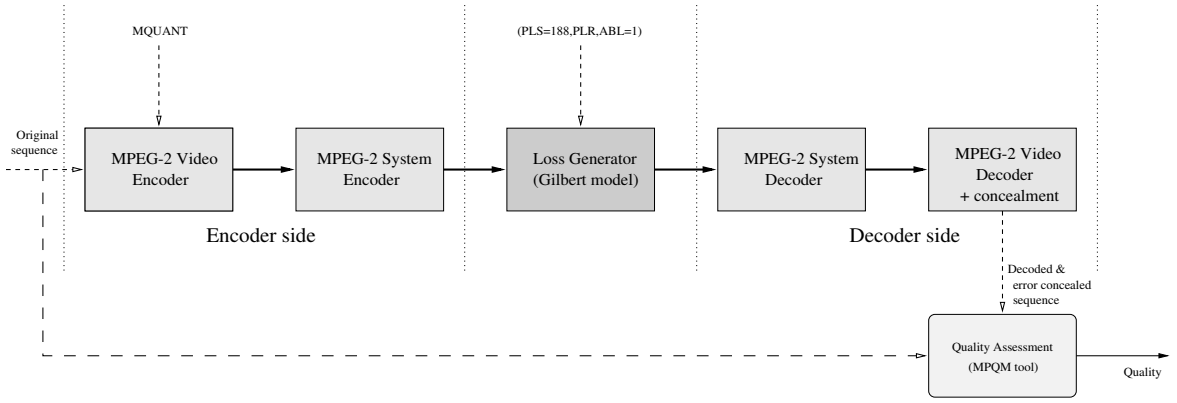


Figure 4. Experimental testbed.

States 0 and 1 respectively correspond to a correct and an incorrect packet reception. The transition rates between the states control the lengths of the bursts of errors. Hence, there are three parameters to be controlled: the packet loss size (PLS), the packet loss ratio ($PLR = \frac{p}{p+q}$) and the average length of a burst of errors ($ABL = \frac{1}{q}$). In our simulations, we imposed a non-bursty ($ABL = 1$) TS packets ($PLS = 188$ bytes) loss process and made the packet loss ratio vary between 10^{-2} and 10^{-7}.

- Video quality was evaluated by the MPQM tool (see Sec. 2.3).

- The last part is an MPEG-2 software decoder constituted by both a TS decoder and a video decoder. The video decoder provides the motion compensated concealment technique briefly explained in Sec 2.2.. This technique was chosen for different reasons. The first is to be consistent with real implementations. The second is to be able to perform the perceptual measurements. Indeed, the vision model currently developed and the derived metrics have been tested for errors below what is called the *suprathreshold* [5]. The problem is that, in general, the degradations due to data losses generate highly visible artefacts (i.e., holes) in the sequence and these errors are all above this suprathreshold. By using concealment techniques, most of the artifacts may be considered as being below the suprathreshold of vision, making the perceptual measure accurate.

[5] Two to three times above the threshold of vision which corresponds to the threshold of visibility of the noise

3.2 MPEG-2 VBR Encoding Impact on Video Quality

First, we study how the OL-VBR encoding process influences video quality. Figures 6 and 7 show how the quality is affected by the MQUANT parameter using, respectively, the PSNR metric and MPQM tool to measure it. While the PSNR versus MQUANT curve may be represented by a decreasing exponential [9], it is to be noted that the MPQM metric exhibits a linear relationship with MQUANT. We verified this important behaviour for the four sequences constituting our testbed. The same characteristic has recently been observed through user's subjective evaluation [7]. Computer simulation results as well as the corresponding fits are represented on Fig. 7 for both the "Barcelona" and "News" sequences. The encoding quality is approximated by a function of the form:

$$Q_E = \chi_Q \times MQUANT + Q_0 \quad (1)$$

where parameters χ_Q and Q_0 have been obtained by minimizing the mean square error (see Fig. 7). The slope χ_Q is directly related to the complexity of the sequence: the higher the encoding complexity, the higher the absolute value of χ_Q. This remark may be verified on the graph. The video sequence "News" is a *Head and Shoulder* type of sequence and does not contain any high spatio-temporal complexities. The absolute value of the slope is thus smaller than for the "Barcelona" video sequence. The value of Q_0 will always be close to 5 (highest quality).

This linear relation between the video quality and the quantizer scale factor may have several impacts on the design of, for instance, perceptual rate controllers or consistent quality regulators.

Now, we have an idea of how the encoding quality behaves according to the MQUANT. We need then to study how the average output bit rate if affected by this

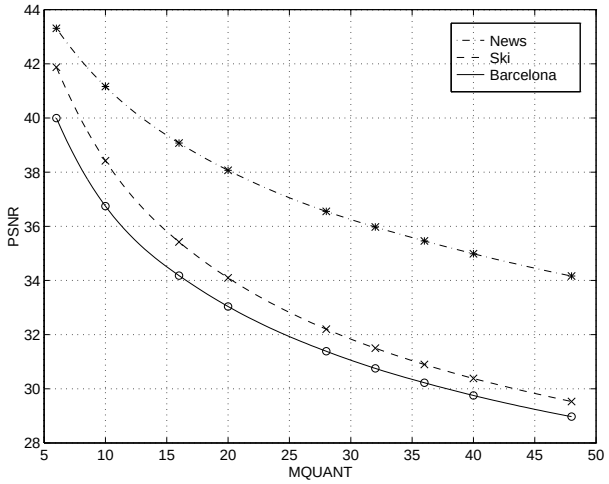


Figure 6. PSNR versus quantizer scale factor for 3 different scenes.

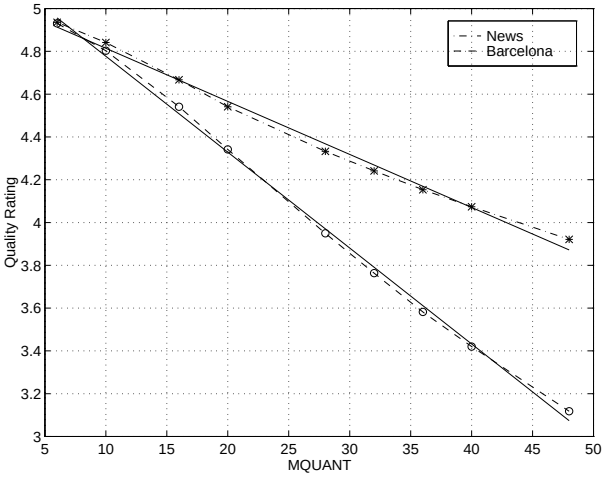


Figure 7. MPQM versus quantizer scale factor for 2 different scenes. Fitting parameters for Eq.1: News: (χ_Q=-0.025, Q_0=5.062), Barcelona: (χ_Q=-0.045, Q_0=5.226)

MQUANT. In [9], it was demonstrated that a power function curve was a good approximation to represent the relation between the quantizer scale factor and the average bit rate:

$$\bar{R} = \chi_R \times MQUANT^{-\xi_R} \tag{2}$$

in which $\bar{R}$ represents the average output bit rate and the parameters χ_R and ξ_R are related to the encoding complexity of the scene.

Figure 8 illustrates this behaviour very well. The parameters χ_R and ξ_R have been obtained by minimizing the mean square error.

Finally, by combining equations (1) and (2), we derive a model for describing how the video quality behaves according to the average encoding bit rate:

$$Q_E = \chi_Q \times \left(\frac{\bar{R}}{\chi_R}\right)^{-\frac{1}{\xi_R}} + Q_0 \tag{3}$$

As stated before, the three main parameters χ_Q, χ_R and ξ_R are somehow related to the spatio-temporal complexity of the sequence (Q_0 will always be around 5.0). However, in this work, we did not investigate this relation any further. This work would not be trivial, as it involves not only encoding complexities but also visual masking phenomenons. We are currently investigating such an extension.

Computer simulation results and the corresponding fitting curve using the equation herebefore are represented in Fig. 9.

An important result that can be extracted from the graph is that the perceptual quality saturates at high bit rates. Increasing the bit rate may thus result, at some point, in a waste of bandwidth since the end-user does not perceive an improvement in quality after a certain bit rate. However, such saturation of quality is not well captured by the PSNR.

3.3 Data Loss Impact on Video Quality

Up to this point, we did not consider any data loss in the video stream. Figure 10 illustrates how the video quality is affected by uniformly distributing TS packet losses over MPEG-2 transport streams. It is shown that, on a semi-logarithmic scale and for a given MQUANT (average bit rate), first the video quality remains constant with the PLR. This constant value corresponds to the encoding quality. Then, beyond a certain PLR, the perceptual quality quickly drops.

The higher the MQUANT value, the higher the PLR after which the quality drops. Indeed, since the PLR is equal to the ratio between the packet loss rate (number of lost packets per second) and the packet rate, then the lower the bit rate (the higher the MQUANT), the lower the packet loss rate for a fixed PLR. Also, the lower the packet loss rate, the lower the number of lost packets per frame on average. Therefore, the higher the MQUANT, the higher the PLR for an equivalent perceived degradation (see Fig. 10).

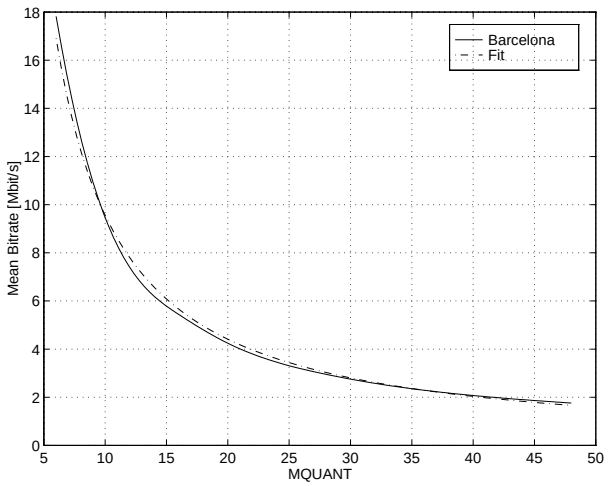


Figure 8. Average output encoding bit rate versus quantizer scale factor (MQUANT) for Barcelona. Fitting parameters for Eq.2: (χ_{R}=124.762, ξ_{R}=1.116)

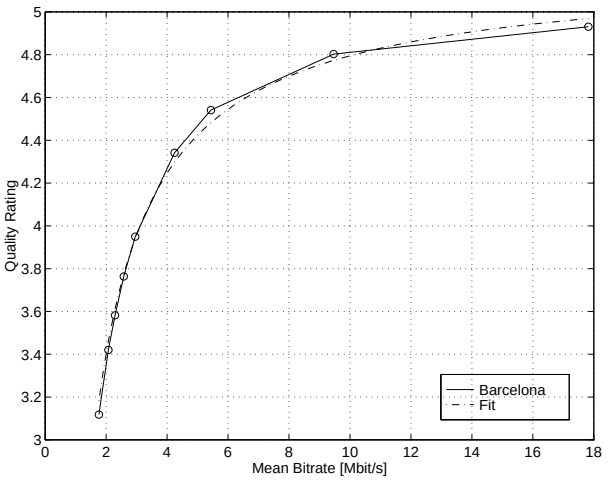


Figure 9. MPQM video quality versus average output encoding bit rate for the Barcelona sequence. Fitting parameters for Eq. 3: (χ_{Q}=-0.045, Q_0=5.225, χ_{R}=124.761, ξ_{R}=1.116)

Hence, the relation between video quality and PLR may be represented by a straight line on a linear scale:

$$Q = Q_E + \chi_L^* \times PLR, \tag{4}$$

where Q_E corresponds to the encoding quality (given by Eq. 3) and χ_L^* depends on both the complexity of the sequence and the average bit rate. In other words, for a given sequence and a fixed MQUANT, the video quality, averaged over the whole sequence, linearly decreases with the PLR.

This relation still holds if we multiply the PLR by a constant. We observed that, for a given $MQUANT$, the relation between the end-to-end video quality, Q, and the product $\bar{R} \times PLR$ could be well approximated by a straight line of slope χ_L. Therefore, Eq. 4 becomes:

$$Q = Q_E + \chi_L \times (\bar{R} \times PLR), \tag{5}$$

where χ_L is almost independent of the $MQUANT$ especially for low to medium bit rates.

4 Joint Impact of MPEG-2 Rate and Data Loss on Quality

In this section, we demonstrate why a joint analysis of the impact of both the MPEG-2 encoding bit rate and the data loss ratio on the video quality is the only way to get meaningful conclusions. We explain, for example, why the video quality may decrease when the encoding bit rate is increased in an error-prone environement.

4.1 Joint Impact Analysis

As stated at the end of the previous section, the PLR and the encoding bit rate (packet rate) are intimately related to each other in regards to their impact on video quality. For example, the higher the bit rate, the higher the encoding quality (up to saturation) but the lower the PLR after which the video quality quickly drops, and conversely. Therefore, the relation between quality and encoding bit rate for a non-zero PLR should somehow exhibit an optimum value. This behaviour is illustrated on Fig. 11. We indeed see that the video quality first increases (encoding quality) with the average bit rate and then decreases after around 4 MBits/s for the "Barcelona sequence" (data loss). This optimal average bit rate directly depends on the sequence type. We observed that it was fairly independent of the PLR though.

Such a result is crucial for the design of network-aware rate controllers, efficient error concealment algorithms, etc.

4.2 Tri-Dimensional Representation

The purpose of this subsection is to put all the results together and represent them by a single graph. Thus, by putting together Eq. 3 and Eq. 5, we obtain a model of the end-to-end video quality Q:

$$Q = \chi_Q \times \left(\frac{\bar{R}}{\chi_R}\right)^{-\frac{1}{\xi_R}} + Q_0 + \chi_L \times (\bar{R} \times PLR), \tag{6}$$

in which the two first terms of the sum represent the encoding quality (see Eq. 3).

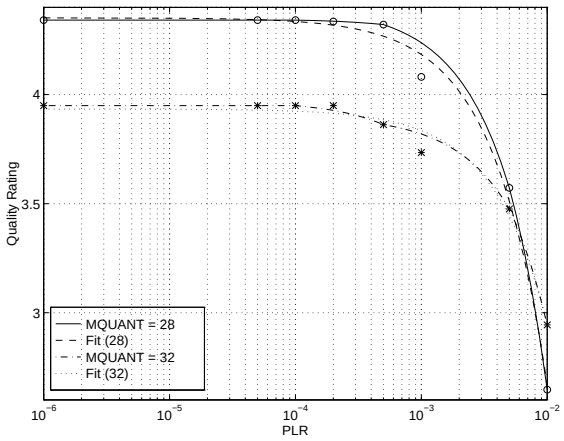


Figure 10. MPQM versus PLR (ALB=1, PLS=188) for MQUANTs=$\{28, 32\}$ using the Barcelona sequence. Fitting parameters for Eq.4: MQUANT=28, ($\mathrm{Q_E}$=4.352, χ^*_{L}=-168.162) and for MQUANT=32, ($\mathrm{Q_E}$=3.934, χ^*_{L}=-98.351). Sequence: Barcelona

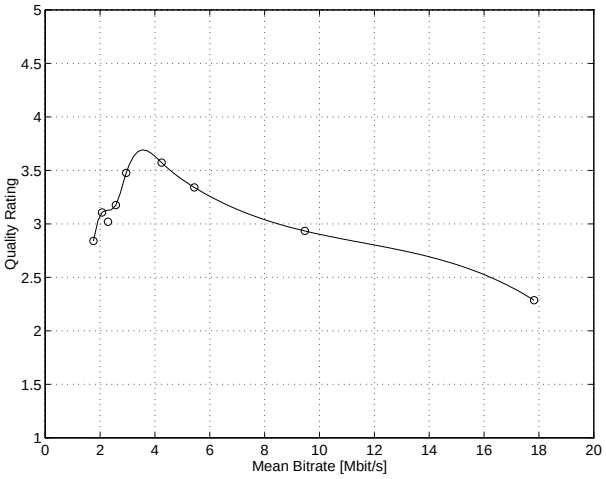


Figure 11. MPQM versus average encoding bit rate for $\mathrm{PLR} = 5 \times 10^{-3}$ for the Barcelona sequence.

We then performed a complete set of measurements in order to verify this relation. The same simulation setup as the one presented in the previous section has been used. Figure 12 presents the resulting surface for the "Barcelona" sequence while Fig. 13 shows its corresponding fit using relation (6).

Several results may be extracted from these graphs. Most of these results have already been discussed throughout this paper. In general, when considering video transmission over lossy networks, not only it is bandwidth consuming to increase the encoding bit rate above a certain threshold due to saturation of quality (which varies according to the scene complexity), it may also be quality consuming. In other words, when the user-oriented QoS is not high enough, an increase of the encoding bit rate at a fixed PLR may even degrade the quality, depending on the position of the working point on the 3D graph presented herebefore. There is an optimal bit rate to be determined that maximizes the end-user perception of the service under certain given network conditions (i.e., network impairments).

Such a conclusion is general enough to be applied to a different encoding system.

5 Conclusion and Future Works

The combined effect of the coding bitrate and the network impairements on the user-perceived quality is still not well understood. However these results are needed for the design and deployment of packet video services. One of the common misleading intuitions is that increasing the coder bit rate enhances image quality. In this paper we have shown that this intuition is proper to a lossless communication channel and that the quality-rate function is no longer a strictly increasing function when video packets are subject to loss.

The major conclusion is that image quality cannot be improved by acting on the coding bit rate only: increasing the bit rate above a certain threshold results in quality degradations. For a given packet loss ratio, there is a quality-optimal coding rate that has to be found. Although the relationship between coding bit rate, packet loss ratio and user-level quality is intrinsically complex, it can be characterized by a simple expression and parameters set. These parameters seem to depend on global properties of the video sequence (e.g., spatio-temporal complexities). Such parameters have to be predicted when video is coded and transmitted in real-time over lossy networks. Therefore, this work is being extended to on-line prediction of the 3D quality graph in the context of MPEG-2, as well as other emerging encoding standards.

References

[1] I. J. 1. *Information Technology - Generic Coding of Moving Pictures and Associated Audio Information - Part 1, 2 and 3*. ISO/IEC JTC 1, 1996.

[2] Barry G. Haskell, Atul Puri and Arun N. Netravali. *Digital Video: an Introduction to MPEG-2*. Digital Multimedia Standards Series. Chapman and Hall, 1997.

[3] Chadd Fogg. mpeg2encode/mpeg2decode. In *MPEG Software Simulation Group*, 1996.

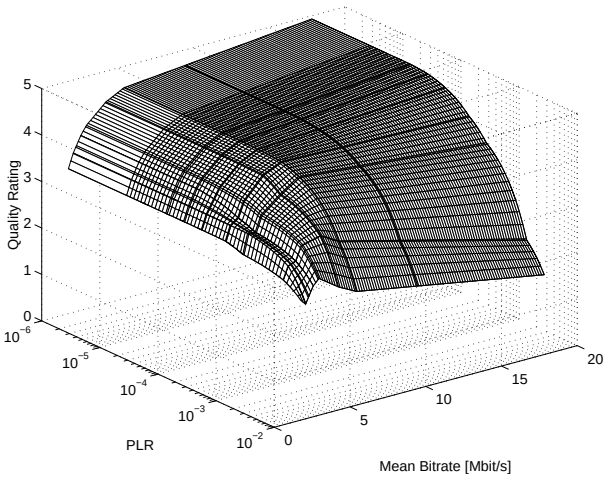


Figure 12. Q versus average bit rate and PLR: Simulations on the Barcelona sequence.

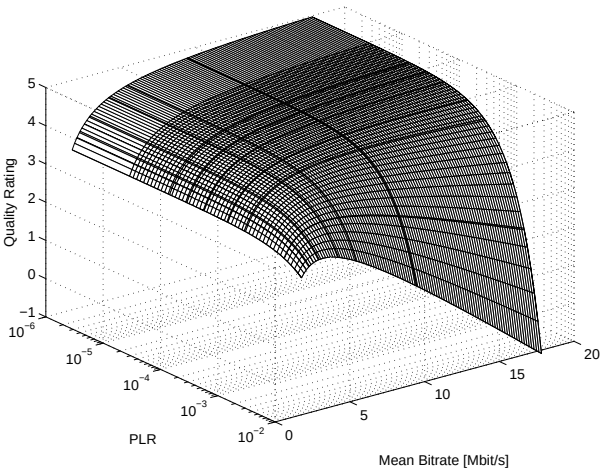


Figure 13. Q versus average bit rate and PLR: Fitting function with parameters: ($\chi_{\mathbf{Q}}$=-0.045, Q_0=5.22, $\chi_{\mathbf{R}}$=124.76, $\xi_{\mathbf{R}}$=1.12 and $\chi_{\mathbf{L}}$=-33.9).

[4] S. Comes. *Les traitements perceptifs d'images numerisées*. PhD thesis, Université Catholique de Louvain, 1995.

[5] J. W. Roberts, J. Guibert and A. Simonian. Network Performance Considerations in the Design of a VBR Codec. In *Queuing Performance and Control in ATM*, pages 77–82. J. W. Cohen and C. D. Pack, June 1991.

[6] G. Karlsson. Asynchronous Transfer of Video. *IEEE Communications Magazine*, 34(8):118–126, August 1996.

[7] Kentarou Fukuda, Naoki Wakamiya, Masayuki Murata and Hideo Miyahara. On Flow Aggregation for Multicast Video Transport. In *6th IFIP International Workshop on Quality of Service (IWQoS)*, May 1998.

[8] M. Ardito and M. Barbero and M. Stroppiana and M. Visca. Compression and Quality. In *Proceedings of the International Workshop on HDTV 94*, Brisbane, Australia, October 1994.

[9] Shigeyuki Sakazawa, Yasuhiro Takishima, Masahiro Wada and Yoshinori Hatori. Coding Control Scheme for a Multi-Encoder System. In *7th International Workshop on Packet Video*, pages 83–88, March 1996.

[10] C. J. van den Branden Lambrecht. A Working Spatio-Temporal Model of the Human Visual System for Image Restoration and Quality Assessment Applications. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 2293–2296, Atlanta, GA, May 7-10 1996.

[11] C. J. van den Branden Lambrecht and O. Verscheure. Perceptual Quality Measure using a Spatio-Temporal Model of the Human Visual System. In *Proceedings of the SPIE*, volume 2668, pages 450–461, San Jose, CA, January 28 - February 2 1996.

Issues of Quality and Multiplexing when Smoothing Rate Adaptive Video

N.G. Duffield*, K. K. Ramakrishnan†, Amy R. Reibman‡
AT&T Labs–Research

Abstract

We have been examining the transport of compressed video over rate-controlled networks, and have developed a smoothing and rate adaptation algorithm, called SAVE (Smoothed Adaptive Video over Explicit rate networks). SAVE attempts to preserve quality as much as possible, by exercising control over the source rate only when essential, when the delay at the source may exceed a bound. Understanding the impact on the quality of the video using rate adaptation is important and we begin to examine some quantitative measures of video quality in this context. For the form of source rate adaptation we advocate, it appears necessary to evolve the quality metrics typically used to evaluate the efficacy of mechanisms to transport video. We also attempt to understand the dynamic nature of the reduction in the encoded rate, since any prolonged impairment is likely to be noticeable. We study the sensitivity of SAVE to its parameters and network characteristics. Finally, the utility of a proposed scheme is measured by its ability to multiplex a large number of streams effectively. We demonstrate that by smoothing the demand from the source, SAVE helps achieve good multiplexing gains. We examine the effectiveness of SAVE across 20 different traces of entertainment videos using different compression algorithms.

1 Introduction

The desired quality of compressed video to be delivered to the receiver varies widely, depending on the application, the potential cost to the user, and the network infrastructure that is available for transporting the video. We have been studying the transport of adaptive compressed video over ATM (Asynchronous Transfer Mode) networks using the ABR (Available Bit Rate) service [2] and [7]. In [2] we proposed an on-line source smoothing algorithm, called SAVE (Smoothed Adaptive Video over Explicit rate networks), that enables us to obtain better multiplexing gain without degrading the quality of the delivered video stream significantly. SAVE exploits the source buffer to reduce the burstiness of the video over short time scales, and uses the inherent negotiation of the rate based feedback control schemes to manage burstiness over the intermediate time scales. Only when the delay contributed by buffering at the source exceeds a reasonable bound is the quantization value increased.

We believe that using feedback based mechanisms for modifying the quantization parameter infrequently to accommodate both the fluctuations in the load (due to burstiness in the video) as well as the rate allocated by the network (possibly due to congestion) is better than carrying the video using other possible QoS classes that do not use feedback. For example, in unrestricted (or open-loop) VBR (Variable Bit Rate) transport, when buffering in the network is unable to overcome burstiness in the aggregate offered load, frames are lost. For a given transmitted rate, better video quality can be obtained by having the encoder produce coarser quality video by adjusting its quantizer, rather than having the network discard packets arbitrarily. In [6] this was demonstrated in the context of rate-adaptation by the encoder.

In this paper, we build on the work described in [2]. We explore more closely the quality achieved using SAVE. In particular, we study the sensitivity of video quality to the various parameters of SAVE and study the impact of network congestion and feedback delay. Another important issue we address here is the gain from having several sources multiplexed together. The multiplexing gain is achieved by overlapping the "peaks" and the "valleys" of the different sources of video on a link at a given time. The resulting aggregate flow tends to be less bursty than the individual flows. The larger the number of simultaneously active flows, the higher the potential for multiplexing gain. We show that there are significant multiplexing gains to be achieved using SAVE.

Re-negotiated CBR (RCBR) [5] is a mechanism proposed to overcome the medium-term variations in compressed video. It uses re-negotiation of the traffic parameters of a Constant Bit Rate connection using ATM signaling messages. Signaling is often performed by software in the end-systems, and typically involves (although not necessarily) considerable processing including the invocation of

*AT&T Labs–Research, Rm. B139, 180 Park Avenue, Florham Park, New Jersey 07932; duffield@research.att.com
†AT&T Labs–Research, Rm. A155, 180 Park Avenue, Florham Park, New Jersey 07932; kkrama@research.att.com
‡AT&T Labs–Research, Rm. 3-233, 100 Schultz Drive, Red Bank, NJ 07701; amy@research.att.com

Connection Admission Control functions in switches. On the other hand, the explicit rate based feedback mechanisms use RM (Resource Management) cells that are generated in hardware in the ATM adapter (network interface) and almost all the negotiation is performed in hardware both in the end-systems and in the switches, without requiring any CAC (Connection Admission Control) functions. Thus, the negotiation is likely to be more efficient. RCBR could use a source rate adaptation mechanism as we have proposed here with SAVE. However, in the work reported so far with RCBR, the use of such an adaptation has not been studied, and instead the primary emphasis has been to examine the probability of re-negotiation failure. We feel that it is better, even with a scheme like RCBR, that re-negotiation failures are handled gracefully by adapting the quantization parameter of the encoder. Moreover, we are able to address in detail the issue of buffer delay management in the face of delayed network responses to requests from the source for increased bandwidth.

There has been considerable work examining the effectiveness of smoothing of stored video; see e.g. [8, 9, 10, 11, 14]. These require advance knowledge of at least a part of the future sequence of frame sizes. This results in a corresponding playback delay of a few seconds. On the other hand, our approach to smoothing is to make it suitable for a wide range of video applications including interactive ones. Because of this desire to transport interactive, high quality video, we attempt to meet tight delay and quality constraints.

The paper is organized as follows. In Section 2 we briefly describe the SAVE algorithm. In Section 3 we explain the various metrics we use to evaluate the quality of the delivered video. Section 4 summarizes the traces used in the experiments. In Section 5 we investigate the robustness of SAVE with respect to variations in the control timescale of the ABR network. In Section 6 we determine the robustness of quality with respect to network contention, and in so doing establish the degree of multiplexing gain available. In Section 7 we examine the effect of tunable parameters within SAVE on the quality. We then conclude.

2 Description of SAVE Algorithm

The operational setting for SAVE is shown in Figure 1. We briefly describe SAVE here. More detail is provided in [2]. There are four data rates which describe the operation of SAVE. First, uncompressed video is fed into an encoder. The IDEAL RATE is that required by the encoder to encode the frame at ideal quality. The ENCODED RATE will be the rate at which the frame is actually encoded. We assume that the encoder will precisely meet a given target. The adapter is able to make rate requests to the network at least once per frame time. Averaged over one frame time, this is the

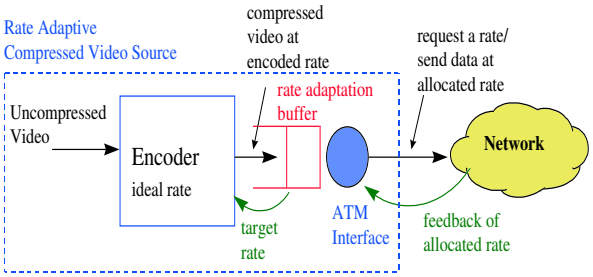


Figure 1. Framework for Rate Adaptive Video in an Explicit Rate Environment

REQUESTED RATE. Finally, the network returns the ALLOCATED RATE to the adapter after some delay primarily involving the propagation delays. Since the network returns a rate for the source to transmit at with every RM cell, the Allocated rate we refer to here is based on the average of the rate returned by the network over a time window of k frames (the results reported here are with $k = 1$). Because the explicit rate-control algorithm used in the network keeps the queueing delays low, the feedback delay is primarily the propagation delay. We consider networks with one-way propagation delays of the order of 50ms to 100ms.

Our aim is to smooth the compressed video using the source buffer while keeping source delays small enough for real-time video: in the region of 100ms or so. Under this constraint, we want the requested rate to be smooth and not unnecessarily large, while at the same time we want the encoded rate to be close to or equal to the ideal rate. This also helps the rate allocation algorithms to stabilize [1]. When a given ideal frame-size cannot meet the delay target given current buffer occupancy and the allocated rate, we encode at a lower rate so as to meet the constraint. We aim to request and be allocated sufficient bandwidth such that this happens rarely. The SAVE algorithm achieves this aim using two disjoint parts. The RATE REQUEST ALGORITHM specifies how the adapter requests bandwidth from the system. The FRAME QUANTIZATION ALGORITHM specifies the rate at which frames are to be encoded.

The Rate Request Algorithm. The requested rate is (essentially) the maximum of two rates: r_{sm} average rate per frame (over some short smoothing window of w_{sm}), and $r_{\max}$, the maximum rate (over a medium term window $w_{\max}$) required to drain a frame from an empty buffer within the delay constraint. Let $f(n)$ be the size of frame n, τ the interframe time and $\tau_{\max}$ the buffer delay constraint. Then for the n^{th} frame

$$r_{\mathrm{sm}}(n) \quad = \quad (\tau w_{\mathrm{sm}})^{-1} \sum_{i=0}^{w_{\mathrm{sm}}-1} f(n-i), \qquad (1)$$

$$r_{\max}(n) \quad = \quad (\tau_{\max})^{-1} \max_{i=0,1,w_{\max}-1} f(n-i). \quad (2)$$

The heuristic for this choice is as follows. We clearly want to request a rate that is commensurate with at least the average rate of the source. By choosing $w_{\rm sm}$ quite small we will make this responsive to rate changes within the medium term. (But if there is a Group of Pictures (GOP) structure of period p present in the encoding, we want $w_{\rm sm} \geq p$ to avoid systematic variations in the requested rate). However, if the allocated rate were just $r_{\rm sm}$, when the short term peak to mean ratio exceeds $\tau_{\max}/\tau$ (typically only 2 to 3) large frames will suffer delay beyond τ. So by allocating the maximum of $r_{\rm sm}$ and $r_{\max}$, the large frames typically find the buffer empty, and so drain within time $\tau_{\max}$. By taking $w_{\max}$ sufficiently large we aim to anticipate the large frame typical of a scene. (Actually there is a little more detail from [2]: we take the maximum also with an autoregressive estimate $r_{\rm ar}$ of the maximum frame size divided by $\tau_{\max}$). Finally, we systematically over-request by a factor $\beta > 1$: the requested rate at frame n is then

$$r_{\rm req}(n) = \beta \max\{r_{\rm sm}(n), r_{\max}(n), r_{\rm ar}(n)\}. \quad (3)$$

Frame Quantization Algorithm. Given a buffer occupancy $b(n)$ and allocated rate $r_{\rm all}(n)$ at frame n, the estimated size of a frame which will empty within the delay bound is $f_{\rm avail}(n) = \tau_{\max} r_{\rm all}(n-1) - \max\{0, b(n) - \tau r_{\rm all}(n-1)\}$. We stipulate that no frame can be encoded in a size less than some proportion γ of its ideal size. The encoded size of frame n will be

$$f_{\rm enc}(n) = \min\{f(n), \max\{f_{\rm avail}(n-1), \gamma f(n)\}\}. \quad (4)$$

Choosing $\gamma > 0$ risks that a frame will be delayed more than τ, the inter-frame time. However, since the network component of the delay is a variable quantity, it may be better to risk a frame to be delayed slightly more (but within the maximum buffer delay constraint $\tau_{\max}$) rather than encode only very coarsely.

3 Evaluation Criteria

In this section, we describe our criteria to evaluate the performance of the overall system: delay, quality, and networking performance. While the criteria for video quality are somewhat heuristic, they are based on known characteristics of how humans subjectively rate the quality of video. [4]

Source Delay. The source buffer should not introduce delay so large as to eat into the delay budget of the network; this would make the network less attractive for real-time services. We assume that there is a sufficiently large playout buffer at the receiver to overcome delay jitter. Hence the primary concern for our work is the aggregate delay introduced in the source buffer and the network. We assume that an overall (one-way) delay budget around 200 milliseconds to 300 milliseconds is acceptable, and that, of this, a delay target of about 100 milliseconds for the source buffer is reasonable for interactive applications. We assume the source buffer is large enough to accommodate any backlog arising from a shortfall of the allocated rate from the encoded rate; at the operating point such differences will only last a short time if our delay target is reached.

Quality and Adaptation. We assume that it is desirable to keep the quality of the video transmitted by the source as close to the ideal quality as possible. Our premise is that sources are adaptive enough that even if the encoded rate falls below the ideal rate, the video quality at the receiver will not suffer significant perceptual impairment. We think this is true provided the shortfalls are sufficiently small, rare and short-lived. We shall use the term *cropping* to describe the reduction of the ideal rate to the encoded rate. In particular, cropping entails a reduction in encoding detail rather than a truncation of the size of the image. Cropping can occur either because of network delay in responding to changes in the short-term average rate, or because congestion forces the network to allocate less than the requested rate. In the latter case, the network is unable to know either the actual video quality aimed for, or the effect the reduction of allocated rate will have on quality; rate allocation amongst sources is done entirely on the basis of the requested rates. The rate allocated by the network itself may be based on a weighted max-min fair allocation [7], where the weights are proportional to the requested rate. This enables the network to favor a video source that has a higher requested rate (possibly because of a higher ideal rate) than another, even though they share the same bottleneck.

In evaluating the operation of SAVE with a given video source, we look at the pattern of cropping over the entire sequence of frames. We strive to keep the proportion of cropping below 20%, only exceeding this level for 0.1% of all the frames. At the rates in question, this amount of reduction of the encoded rate will degrade video quality as measured by the Peak Signal to Noise Ratio (PSNR) by about 1-2 dB. We believe that greater than this amount of degradation is generally perceivable by moderately experienced viewers.

We also look at the dynamics of cropping. In addition to meeting the above criteria for cropping amount and frequency over the whole trace, we also want to avoid long sequences of consecutively cropped frames. So we looked at the distribution of the length of bursts of successive frames cropped above (or below) the 20% threshold (other thresholds could also be considered.) We expect that the quality of video is perceived to be equivalent to the quality of the worst segment, provided this segment is long enough

[4]. Thus we will be particularly interested in the maximum burst length of cropping greater than the threshold of 20%. Conversely, during periods when cropping is below the threshold (including when there is no cropping) we expect there to be little impact on perceivable quality, even for experienced viewers.

We aim to keep the maximum burst length of cropping not much greater than any GOP period present in the encoding. Otherwise cropping of the large frame in successive GOPs could lead to noticeable quality reduction over timescales up to 1 second. An example of this would be the cropping of consecutive I-frames in a 12 frame GOP encoded at 24 frames per second. Cropping of an I-frame can impair quality for subsequent frames. In addition, the periodic nature of the impairments will make them more noticeable [4]. For this reason, when there is a GOP structure present, if two frames cropped more than 20% are separated by less than the GOP length, then for statistical purposes we treat all the intermediate frames as though they have been cropped more than 20%.

Robustness to Network Feedback Delay. The rate allocation mechanism of the explicit rate network is not expected to instantaneously allocate a rate in response to requests. To include the time needed for the stabilization of the rate allocation algorithms [1], we need to verify that quality measures are preserved even for a relatively large feedback delay (considered in number of frame times).

Even in the absence of network congestion, frame cropping is likely occur when the short term average demand suddenly changes. Therefore, some frames are likely to suffer cropping until the network allocates an increased rate.

Channel Capacity and Multiplexing Gain. We look at the number of sources that may be multiplexed within a link of a given capacity when our delay constraints and cropping criteria are met. This is eventually the criterion that will guide us to choose one algorithm over another. For statistical multiplexing gain we want to be able to assign capacity to an aggregate of sources at less than the peak of their aggregate requested rate. During transient periods in which the aggregate requested rate exceeds the capacity, the explicit rate mechanism of the network will proportionately reduce the allocation to each source so as to avoid congestion. We use the term *rate-reduction* when the allocated rate is less than the requested rate. We determine the sufficiency of such an allocation by establishing the extent to which such rate-reduction is compatible with the delay and quality targets described above.

Sensitivity to Algorithm Parameters. Finally, SAVE has a number of tunable parameters: $w_{\mathrm{sm}}, w_{\max}, \beta$ and γ. We shall investigate the sensitivity of the quality metrics to variations in these parameters.

4 The Experimental Traces

Here we summarize the properties of the 2 sets of traces used in the paper (maintaining conformity with the labeling of [2]).

A. An MPEG-2 encoding of a 40680 frame portion of "The Blues Brothers", with M=1. There is no periodic structure. The frame rate was 24 frames per second.

E. 19 MPEG-1 traces, each with 40,000 frames, compiled by Rose. They originate from cable transmissions of films and television; see [12] for further details. The GOP is 12 frames with an IBBPBBPBBPBB pattern. For our experiments we assumed a uniform rate of 24 frames per second.

Many of the experimental results were carried out using 1- to 38-fold aggregations of the 19 traces from set E. Each trace was used twice, all traces having randomized phase and ordering within the aggregation. Experiments were repeated over different randomizations.

5 Sensitivity to Network Feedback Delay

We examine the sensitivity of SAVE to the feedback delay from the network in the context of Trace A. In the experiments we assumed the feedback delay for the network to allocate requests was fixed:

$$r_{\mathrm{all}}(n) = \begin{cases} r_{\mathrm{req}}(n-\delta) & n > \delta \\ r_0 & n \leq \delta \end{cases} \tag{5}$$

Here r_0 is an initial rate given to the source until the network responds to the first rate request after the feedback delay of δ. We used the mean ideal rate as r_0.

We display the extent of degradation measured by our quality metrics in Table 1. The target for the proportion of frames not suffering more than 20% cropping of the frame size is clearly met. In fact, only 0.3% to 0.4% of the frames suffer any cropping at all. As seen in the table, the reduction in quality shows a slight sensitivity to the feedback delay. We feel that this reduction is well within reasonable levels for our target for quality degradation.

Even with a feedback delay of 4 frame times (about 166 milliseconds) the maximum delay contributed by the source buffer was 105ms, the next largest delay being 98ms, again deemed acceptable based on our target of 100 ms for the source buffer delay.

Also shown in Table 1 are the dynamic measures of quality. We look at the statistics of the length of bursts of consecutive frames that have more than 20% cropping, and those of the complementary bursts of frames for which

F.back	Proportion of Cropping			Mean Burst Crop	
Delay	$> 0\%$	$> 20\%$	$> 50\%$	$\leq 20\%$	$> 20\%$
1 fr.	0.0032	0.0011	0	2257	3.28
2 fr.	0.0036	0.0012	0	2031	3.25
3 fr.	0.0037	0.0014	0	2030	3.85
4 fr.	0.0042	0.0016	0	1933	4.00

Table 1. Proportion of Frames experiencing a degradation in quality for trace A. Also shown are mean number of successive frames having less than 20% cropping (if any), and mean burst length of successive frames failing the 20% cropping criterion.

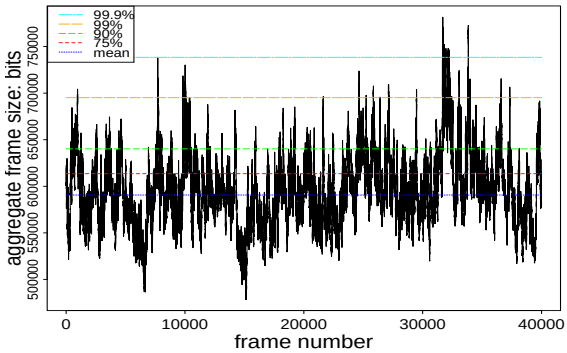


Figure 2. SMOOTHING BY AGGREGATION. Aggregated requested rate of 19 40000-frame MPEG-1 video traces in set E. The vertical range is about 1/2 of the lowest value

cropping, if present, is no more than 20%. In the table we see that the mean burst length of cropping $> 20\%$ ranges from 3.25 to 4, for varying feedback delays. In contrast, the mean length of the complementary bursts is around 2000. When the feedback delay is 4 frames, the longest burst of cropping $> 20\%$ was 8 frames, or about 1/3 of a second. Furthermore, there were only 26 such bursts, involving only 67 frames in total. This was for the trace of 40680 frames. This will produce a very slight visual impairment, probably visible only to the most experienced and critical viewers.

6 Multiplexing Gain and Sensitivity to Network Congestion

In this section we investigate the bandwidth requirements of the aggregate request rate from a number of sources. We want to determine the extent to which statistical multiplexing across sources is feasible: we aim to be *able* to allocate less than the aggregated peak requested rate of sources. The extent to which this is possible depends

Capacity	delay		fraction cropped	
(% of peak)	mean	max	$> 0\%$	$> 20\%$
100%	20.4	99.0	0.003	0.0006
90%	20.5	99.9	0.004	0.0006
80%	20.8	102.2	0.006	0.0008
70%	21.1	103.8	0.008	0.0010
60%	21.5	104.9	0.011	0.0017
50%	21.8	106.2	0.013	0.0022

Capacity	burst crop $> 20\%$		burst crop $\leq 20\%$	
(% of peak)	mean	max	mean	min
100%	2.2	7.3	2900.	108.
90%	2.2	8.9	2600.	106.
80%	2.7	13.8	2297.	79.
70%	3.3	20.2	1869.	28.
60%	7.2	75.6	1405.	25.
50%	9.0	96.9	1235.	20.

Table 2. IMPACT OF STATISTICAL REDUCTIONS IN ALLOCATED RATE. Averaged impact on single traces within 5-fold aggregation in set E. Capacity expressed as quantile of peak aggregated requested rate. As capacity decreases, observe insensitivity of delay and mean burst length of cropping over 20

on the impact to quality experienced by individual sources caused by transient contention. We are concerned with establishing the minimum capacity required; we intend to deal elsewhere with the problem of characterizing the statistical behavior of requests for the purposes of admission control.

Rate-Reduction, Cropping, and Statistical Multiplexing. A time-series of the aggregate *requested rate* from 19 traces of set E is shown in Figure 2. Consider a channel carrying the aggregated traffic from these sources. If the capacity C of a channel is less than the maximum aggregate requested rate, then periods of contention will occur from frames n (of the aggregate stream) for which the aggregate requested rate $R(n)$ exceeds the capacity C. The ABR rate-allocation algorithm responds to the demand exceeding the available bandwidth by allocating bandwidth to individual sources in proportion to their requests, the proportion being such that the total allocation equals the available bandwidth. We achieve this by using a weighted max-min fair allocation algorithm in the network [7]. Thus when $R(n) > C$ the rate allocated to each source will be a proportion $C/R(n)$ of its request.

Simulating Rate-Reductions in Aggregates. We can gauge the effect of attempting to use less capacity than the peak aggregate rate as follows. For a given set of traces, we run the SAVE algorithm to construct the requested rate for each trace as before, then sum to yield the aggregate requested rate $R(n)$. For a capacity C, we construct the

proportional rate-reduction process

$$p(n) = \max\{1, C/R(n)\}. \quad (6)$$

To asses the impact of contention on an individual source we rerun the SAVE algorithm, but now proportionately reducing the allocated rate, subject to the network roundtrip delay δ. Hence (5) is replaced by

$$r_{\text{all}}(n) = \begin{cases} p(n-\delta) r_{\text{req}}(n-\delta) & n > \delta \\ r_0 & n \le \delta \end{cases} \quad (7)$$

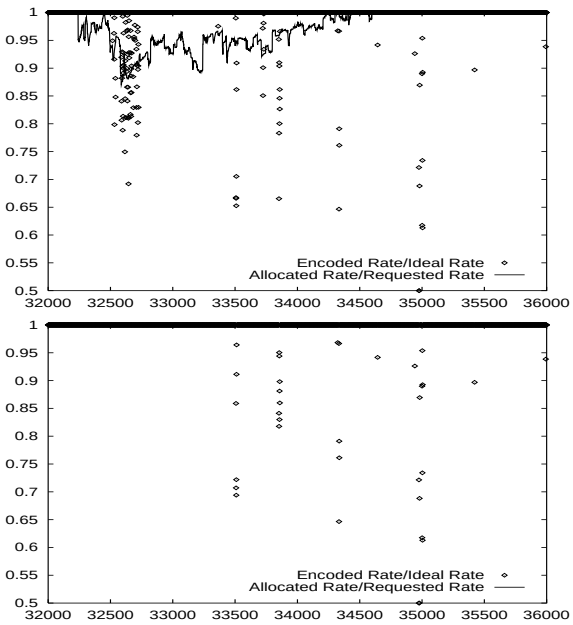


Figure 3. PATHWISE COMPARISON OF CROPPING AND RATE-REDUCTION. TOP: Segment of trace of Set A, impacted by contention with 10 traces from Set E, the total allocated at the 90$^{\text{th}}$ %-ile of the peak requested rate. Encoded/Ideal rate ($=$ 1$-$ cropping proportion) drops below 1, partially in response to contention. BOTTOM: Encoded/Ideal rate for same trace without contention. The horizontal axis is the frame number.

Note that this ignores some second order effects: reduction of the allocated rate for a given frame may increase buffer occupancy at the end of that frame time, and hence the rate request on the next frame may be increased to meet the delay target. We overcome this approximation when we report results using a network cell-level simulation, where we obtain an accurate characterization of the effects of a reduced rate being returned by the network. We show a sample of the effects of contention in Figure 3. The points show the ratio of encoded to ideal rate (i.e., 1$-$ cropping proportion). The base behavior without contention is shown at the bottom. At the top is shown the modified ratio in the presence of contention, as reflected by the ratio of allocated to requested rate. Note additional cropping that now occurs between frames 32,500 and 33,000. Most of the cropping is less than 20% (i.e., encoded/ideal rate greater than 0.8) even though allocated rate is about 90% of the requested rate.

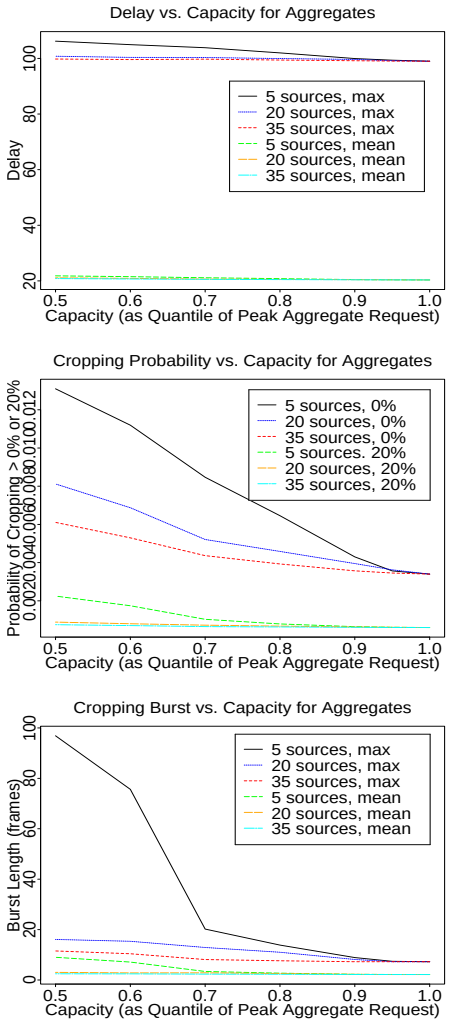


Figure 4. VARIATION WITH AGGREGATION SIZE OF IMPACT OF RATE-REDUCTION THROUGH REDUCED CAPACITY. For aggregations of 5, 20, 35 sources, averaged impact on single traces within aggregate; capacity expressed as quantile of peak aggregate requested rate. TOP: mean and max. delay. MIDDLE: proportion of cropping $>$ 0% and $>$ 20%. BOTTOM: mean and max. burst length of cropping $>$ 20%.

Impact on Quality of Statistical Multiplexing. We assess the impact of statistical multiplexing on quality of single sources by subjecting SAVE to the simulated rate-reduction process $p(n)$. We explore the variation of quality metrics with aggregation size and allocation. Table 2 summarizes experiments with 5 fold aggregates of traces from set E.

The statistics are averaged over individual traces. Figure 4 shows the variation in delay, cropping and burst length of cropping as a function of capacity, over a range of aggregations from 5 to 20 to 35 sources. Both show that mean and maximum of the delay are quite insensitive to both aggregation size and the degree to which capacity is set below the peak aggregated requested rate. This shows that one of the design criteria of SAVE (cropping to avoid delay) operates well over a wide range of conditions. The price to be paid for this is the frequency and duration of cropping. Both increase as the allocated rate is reduced. Our criterion that the *maximum* burst of cropping $> 20\%$ should not exceed a GOP (12 for the traces of Set E) is met when capacity is (not much) less than the 90^{th} percentile of the peak aggregate requested rate. The overall cropping criteria (no more than 1 in 1000 frames to be cropped more than 20%) is already satisfied at this capacity.

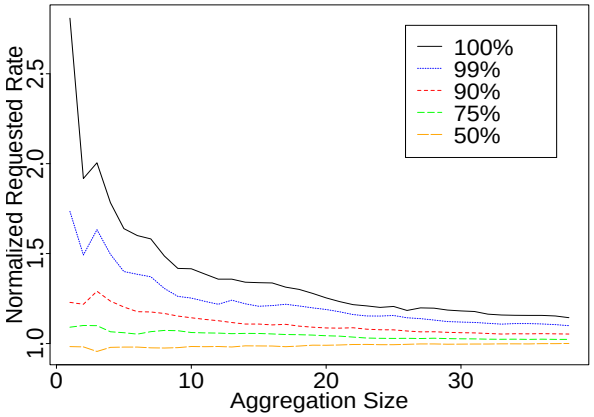


Figure 5. SMOOTHING THE REQUESTED RATE BY AGGREGATION. Quantiles of the aggregated rate for 1- to 38-fold random phase aggregations of traces from Set E. Quantiles normalized as a multiple of mean aggregate requested rate.

As the size of the aggregate increases, performance measures improve for capacity at a given quantile of the peak aggregate rate. Thus, averaging across sources means that it will be possible to obtain the desired performance with capacity equal to smaller quantiles of the peak rate for large aggregations. However, smoothing means that the quantiles of the aggregate requested rate (expressed as a proportion of the mean) become closer as the number of sources in the aggregate increases; see Figure 5. Thus the quality obtained becomes relatively insensitive to which quantile is chosen. In the example in Figure 5 the peak of the aggregate requested rate is only about 20% higher than the mean for aggregations of 20 or more sources.

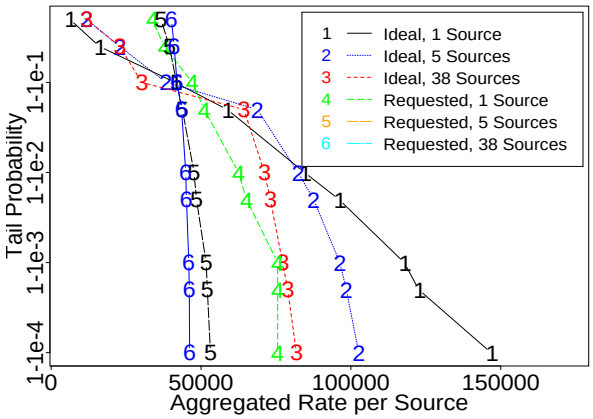


Figure 6. RATE SMOOTHING AND AGGREGATION Tail distribution, for ideal and requested rates per source, for aggregations of 1,5 and 38 sources of Set E. High quantiles of requested rate are lower for requested rate.

The smoothing by SAVE of individual sources reduced the variability of the requested rate compared with the ideal rate. This smoothing within sources in turn makes smoothing across sources in an aggregate more effective. We demonstrate this by displaying in Figure 6 the tail distribution of the aggregate rate per source for ideal and requested rate, for aggregations of 1,5 and 28 sources from Set E. The high quantiles of the requested rate (above the 90%-ile) take much lower values. The variability of the ideal rate does not approach that of the requested rate, even for very large aggregations. The price of this reduction in variability that quantile below about the 90%-ile, and indeed the mean, are higher for the requested rate than the ideal rate. An important potential benefit of the reduction of the high quantiles is for measurement based admission control. We think of an effective bandwidth for a source as a rate required to accommodate its extremes of variability. The narrowness of the distribution of the requested rate means that measurements of the effective bandwidth will be less influenced by statistical errors than for the ideal rate.

Single source behavior and comparative multiplexing gain. When SAVE is employed for transport of a single source's video stream, then we shift the focus away from the statistical properties of aggregations and instead ask what is the *constant rate* required for a given source so that SAVE achieves the quality targets. In our experiments on this topic, it was necessary to set the parameter γ (the minimum proportion of the frame to be encoded) to zero. Since the rate given to the source is constant, when $\gamma > 0$ we are unable to adaptively increase the allocated rate. When this

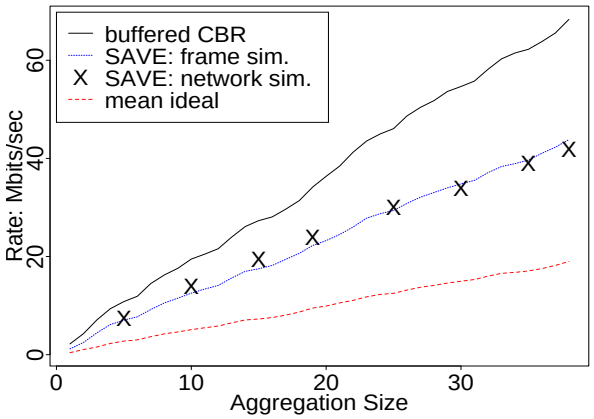


Figure 7. UTILITY OF EXPLICIT RATE FOR MULTIPLEXING GAIN. Bandwidth requirements, normalized by mean ideal rate, for 1- to 38-fold aggregates of Set E. LOWER CURVE: SAVE, 90th percentile of aggregate requested rate. UPPER CURVE, CBR rate for 100ms source buffer. Also shown: mean ideal rate, and capacity estimate based on full network simulation described below.

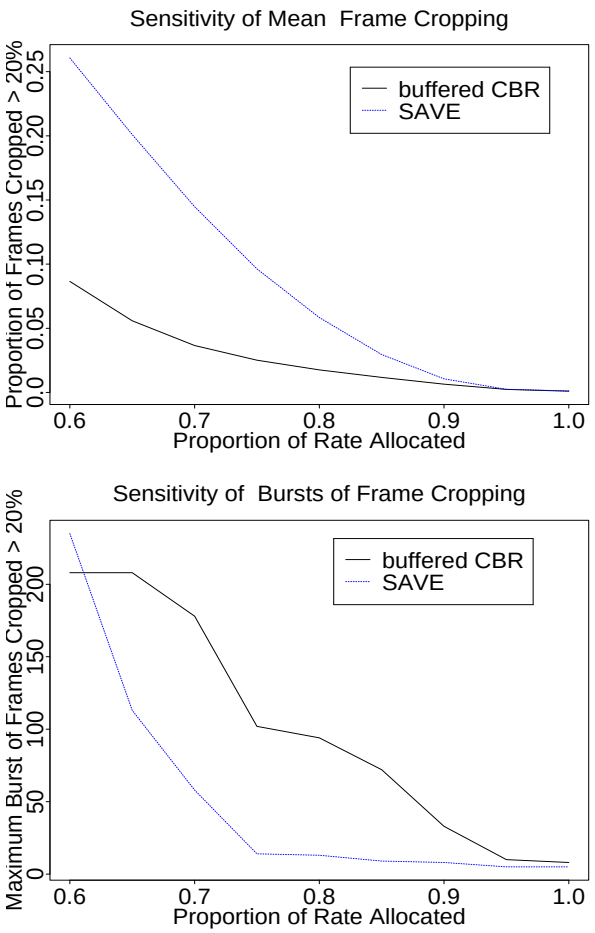


Figure 8. COMPARATIVE SENSITIVITY OF FRAME CROPPING IN SAVE AND BUFFERED CBR TO SYSTEMATIC RATE-REDUCTION OR UNDERALLOCATION OF BANDWIDTH. TOP: average cropping more sensitive for SAVE. BOTTOM: burst cropping more sensitive for CBR.

occurs over lengthy periods of high activity it leads to either excessive delays or buffer overflow. We refer to this mode, in which SAVE fits frames to a buffer that is drained at constant rate (CBR), as "buffered CBR".

Over the 19 traces of Set E, the required CBR rate was found to exceed the mean ideal rate by a factor of between about 2.4 and 6.0, with a mean of about 3.6. For the trace of set A it is about 2.2 times the mean ideal rate. To examine the multiplexing performance of the SAVE algorithm working in tandem with an explicit rate network, we compare these CBR bandwidth requirements to the bandwidth required when using SAVE in the network it was designed for, a network with explicit rate feedback. For CBR, we use the sum of the CBR rates of the constituents of the aggregate. For SAVE, we use the 90th percentile of the peak aggregated requested rate. These are displayed for 1- to 38-fold aggregations from set E in Figure 7. The CBR, SAVE and mean ideal rates (for data alone, neglecting any protocol overhead) are in approximate proportions $3.6 : 2.3 : 1$.

The higher rate allocation for the buffered CBR reflects the necessity to allocate a sufficient rate to accommodate long-lived trends found in the video sources. If bandwidth allocated by the network is slightly lower than needed, degradation of quality will occur, often over a long burst. Since this is an inherent property of the video source, we expect that similar allocations will be required for other algorithms which adapt and transport real-time video at constant rate and similarly high quality. We compared the buffered CBR rates with the peak smoothed rate of the algorithm in [10] under the same delay constraint; the rates were similar.

We quantified the relative sensitivities of SAVE and buffered CBR to systematic rate-reduction (for SAVE) or underallocation (for buffered CBR). One motivation for this is trying to understand the effect of errors in traffic characterization at admission control time. We compared the sensitivity of the quality metrics (frequency of cropping $>$ 20%, and maximum burst length of such cropping). For SAVE we reduced the time-varying allocated rate by a fixed proportion; for buffered CBR we reduced the constant rate by the same proportion. The results for trace A are summarized in Figure 8 for proportions from 0.6 to 1. Burstiness of cropping is not very sensitive for SAVE for rate-reductions down to at least 0.75, but then increases rapidly. Buffered

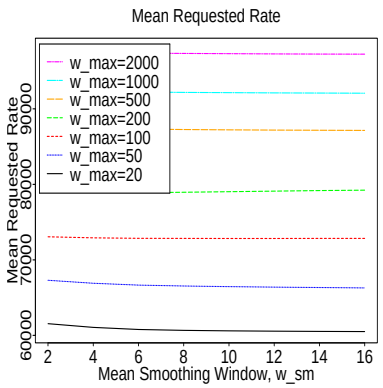


Figure 9. MEAN REQUESTED RATE VS. w_{sm}, trace set A, for varying $w_{\max}$

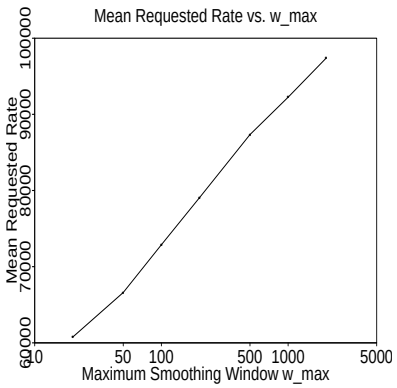


Figure 10. MEAN REQUESTED RATE VS. $w_{\max}$, averaged over w_{sm}

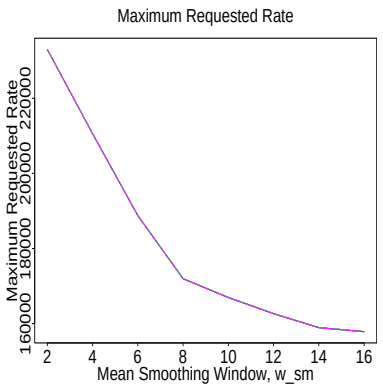


Figure 11. sc Maximum Requested Rate vs. w_{sm}, trace set A. Curves coincide for $w_{\max}$ from 20 to 2000.

CBR on the other hand is quite sensitive to underallocation, for reasons outlined above. Cropping frequency for SAVE is more sensitive, in this case degrading to 1 in 1000 for a systematic rate-reduction of about 0.9.

The ordering of the sensitivities to rate-reduction by the network has the consequence that if we relax the quality target on maximum burst length of cropping more than 20%, and replace it by one on, say, the mean burst length, the performance gap between SAVE and buffered CBR narrows somewhat. In this case the ratios of required capacities for aggregates of buffer CBR, SAVE, and the ideal rate were found to be in ratios $3.2 : 2.1 : 1$.

Validation through a Network Level Simulation. The foregoing analysis was based on the requested rate generated by SAVE operating on individual traces. The aggregated requested rate was in turn used to estimate the impact of contention on individual sources. While having the simplicity that only individual traces need be analyzed, this approach has the potential drawback that the impact of rate-reduction on buffer occupancy is ignored. Rate-reduction can cause increased buffer occupancy, which can in turn cause cropping to be increased. In addition, the feedback delay has the potential to vary because of queueing at the switches, although we try to keep this small. This motivates the evaluation of SAVE in a full network simulation at the cell level. The rate allocated by the network, including the dynamics of contention are accurately reflected in the network level simulation.

We simulated the operation of SAVE in an ATM network using the ABR service [13]. The explicit rate option is used, with the switches implementing a weighted max-min fair rate allocation algorithm [7]. The simulation was performed at the granularity of ATM cells, rather than video frames. We used a relatively simple configuration of all the sources "fanning in" to a network of 3 switches in tandem. The bottleneck is the link between the second switch and the destination host, the sink for all the video sources. The capacity of the bottleneck link was tuned in order to find the minimum capacity required for acceptable quality in the individual sources. The initial one way delay is 50 milliseconds. The simulation was run for approximately 20,000 frames from each source.

The results from the network level simulation are displayed as points in Figure 7. The other traces in the figure have been adjusted for protocol overhead. All include a factor for the ATM cell header (about 10%); the SAVE frame simulation also contains a factor for Resource Management cells (about another 3%). We see that the full network simulation and the frame level simulation of individual traces are in close agreement. The difference between them was at most 10%.

7 Parameter Sensitivity of the SAVE Algorithm

Sensitivity to Smoothing Parameters. The SAVE smoothing algorithm has two components of smoothing, as described in Section 2: the short-term smoothing window w_{sm} and the medium term maximum window $w_{\max}$. We nominally choose w_{sm} to be of the order of 12 frames, the GOP length of the traces in set E. The maximum window $w_{\max}$, which aims to capture the characteristic of the large frames of a scene, was nominally set to 1000 frames. In this section, we study the sensitivity of SAVE, specifically its quality and bandwidth usage, to these two smoothing parameters.

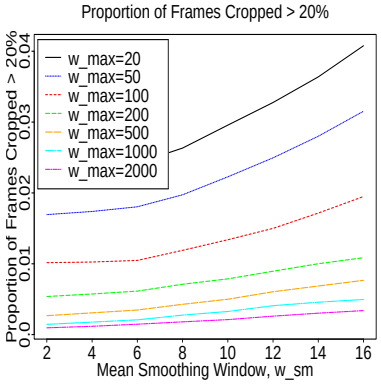


Figure 12. PROPORTION OF FRAMES CROPPED $> 20\%$ VS. w_{sm}, trace set A for varying $w_{\max}$

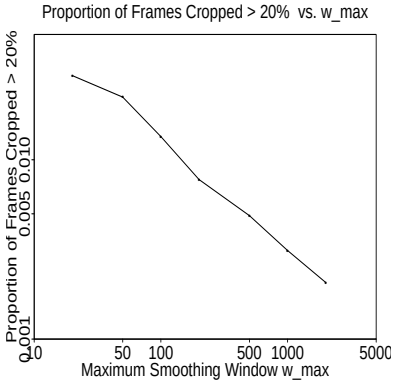


Figure 13. PROPORTION OF FRAMES CROPPED $> 20\%$ VS. $w_{\max}$, trace set A, averaged over w_{sm}.

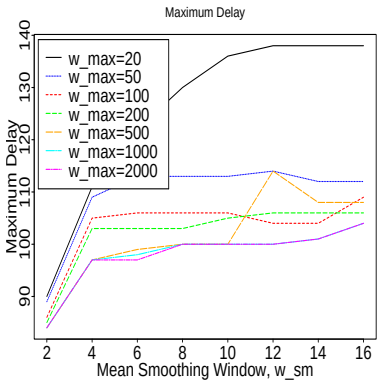


Figure 14. MAXIMUM DELAY VS. w_{sm}, trace set A for varying $w_{\max}$.

Figure 9 shows the variation of the mean requested rate as both $w_{\max}$ and w_{sm} are varied. As we observe from the figure, there is very little sensitivity to the value of w_{sm}. There is a small reduction in the mean requested rate as w_{sm} is increased from 2 to 6 or 8 frames. Beyond that, there is almost no change in the mean requested rate. However, it is clear that there is considerable sensitivity to the value of $w_{\max}$. We isolate this in Figure 10 by plotting the average over w_{sm} of the mean requested rate as a function of $w_{\max}$. Note the logarithmic horizontal scale. As $w_{\max}$ is increased, the mean requested rate increases, reflecting the fact that one large frame tends to have an effect over a longer period of time. The mean requested rate increases slowly with $w_{\max}$–approximately logarithmically–going from 60000 bits/frame when $w_{\max}$ is 20, to nearly 90000 bits/frame when $w_{\max}$ is 1000.

The maximum requested rate is insensitive to $w_{\max}$, as shown in Figure 11: the curves of the maximum request as a function of w_{sm} coincide for $w_{\max}$ in a range from 20 to 2000. However, the maximum requested rate is a decreasing function of the smoothing window w_{sm}. It reduces rapidly as w_{sm} increases up to 8. Beyond that, we begin to reach a point of diminishing returns, and the maximum requested rate reduces slowly as w_{sm} goes beyond the GOP value of 12. The form of dependence on $w_{\max}$ and w_{sm} indicates that the largest requested rate is governed by the short-term average over a few large frames and the mean requested rate is governed by the large frames observed over a longer, scene-level, timescale.

The impact on quality is likely to be of more interest as the two parameters are varied. Figure 12 shows how the proportion of frames cropped by more than 20% varies. The primary sensitivity is once again to $w_{\max}$. For values of $w_{\max}$ greater than 100, the proportion of frames cropped over 20% remains below 0.5% for the entire range of w_{sm} we examined. For smaller values of $w_{\max}$, less than 100, the proportion of cropped frames shows a small amount of sensitivity to w_{sm}. However, the primary sensitivity of the quality is to $w_{\max}$; see Figure 13. Note the log-log scale: the proportion of frames cropped more than 20% decays slowly, as a power law, in $w_{\max}$ (in fact $\approx$ const. $\cdot\, w_{\max}^{-0.85}$). For the acceptability criterion we selected (no more than 0.1% of frames cropped $> 20\%$), $w_{\max}$ needs to be reasonably large, of the order of 500 frames or more. This supports our initial intuition that $w_{\max}$ needs to capture the scene-level behavior, which is likely to be of the order of a few seconds. We also observed the statistics of the burst length of cropping with varying w_{sm} and $w_{\max}$; Generally, the mean burst length ranged from 1.5 to 3.5, increasing with w_{sm}. The maximum burst length was a decreasing function of $w_{\max}$, in a range from 25 to 2, and generally increasing with w_{sm}. Finally, Figure 14 show that maximum source buffer delay is reasonably insensitive to w_{sm} once $w_{\max}$ is greater than about 50 frames; it is within the 100ms target for $w_{\max}$ $\geq$1000. In summary, $w_{\max}$ is the main tunable parameter determining quality.

Sensitivity to Target Source Buffer Delay. Up to now, we have examined the performance of the algorithm with a source buffer delay target of 90 milliseconds. We now examine the sensitivity of quality to this delay target, as we vary it from 50 milliseconds to 120 milliseconds, for Trace A. Figure 15 shows the variation of the mean and peak for the requested rate from the source as we vary the delay target. The mean ideal rate (in terms of bits/frame) is 54823

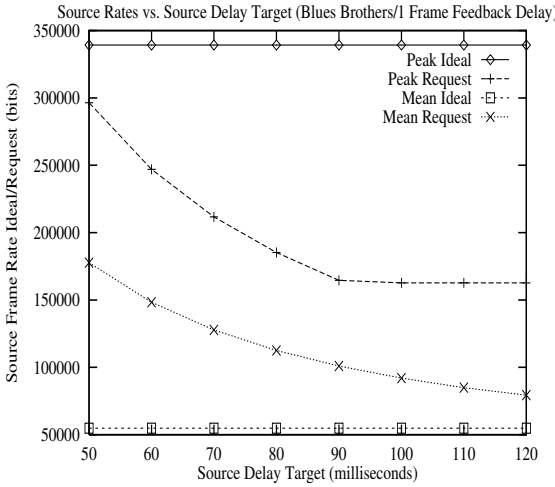


Figure 15. Behavior of the Requested Rate for Trace A for varying Source Buffer Delay Target

bits. The mean request rate (to achieve the desired quality target) obviously is higher, but decreases as the delay target at the source becomes larger. When the delay target is 50 milliseconds (the frame time is 41.6 milliseconds), then the mean request rate is almost 3.24 times the mean ideal rate. We need to request a large rate from the network to ensure that the buffer occupancy is kept low to accommodate the 50 ms delay target. However, when we go to a target delay of 100 milliseconds, the mean request rate is 1.68 times that of the ideal. The larger the source buffer, the smaller the overall rate that needs to be requested from the network while still meeting our quality target. Also shown in Figure 15 is the behavior of the peak for the ideal and requested rates. The peak requested rate drops rapidly from 296400 bits for a source target delay of 50 milliseconds, to 164653 bits for a delay of 90 milliseconds. However, the peak rate does not drop further, and in fact remains flat for this trace, when the delay target increases further from 100 to 120 milliseconds. It is important to note that the ratio of the peak ideal rate to the peak requested rate (when the target delay is 90 milliseconds) is about 2.06, which is a substantial benefit derived from using SAVE.

Table 3 shows more details on the degradation in quality relative to the source target delay, varying from 50 to 120 milliseconds. Also shown is the mean delay at the source. The 99.99^{th} percentile and higher (also the maximum delay) are consistently below the target delay we chose for the particular experiment.

We observe that the proportion of frames that suffer a degradation in quality increases slightly with increasing delays – a somewhat counter-intuitive result. This is because the uncertainty increases as the size of the source buffer increases. For larger target delays, we should see the source

Target Delay (ms)	Cropping $\geq 0\%$	Cropping $\geq 20\%$	Mean Delay (ms)
50	0.0012	0.00017	13.9
60	0.0014	0.00022	16.7
70	0.0017	0.00034	19.3
80	0.0023	0.00071	21.9
90	0.0032	0.00106	24.5
100	0.0041	0.00120	27.1
110	0.0048	0.00145	29.8
120	0.0054	0.00182	32.4

Table 3. Proportion of frames suffering a degradation in Quality as a Function of Source Target Delay

Over-request β	Cropping $> 0\%$	Cropping $> 20\%$	Mean Delay (ms)
0.9	0.0886	0.0212	34.2
1.0	0.0082	0.0018	27.8
1.05	0.0032	0.0011	24.5
1.10	0.0022	0.0007	23.0
1.20	0.0007	0.0002	20.8
1.30	0.0003	0.0001	19.1

Table 4. Proportion of frames suffering a degradation in Quality as a Function of Source Rate Over-request

buffer is full enough to cause a subsequent frame to be cropped slightly more frequently. However, no frame suffers a degradation of 50% or more in any of these experiments. We are also within the quality target of no more than 0.1% of frames suffering more than 20% degradation.

It is clear from the above that there is a tradeoff between the source target delay and the rate requested of the network. The requested rate goes up as the target delay becomes smaller. Note that SAVE is still able to maintain quality within acceptable levels even while meeting the much smaller delay target.

Sensitivity to Over-request of Network Rate. As described in Section 2, the SAVE algorithm systematically over-requests a rate from the network by a factor $\beta > 1$ relative to its calculated needs; see equation(3). This allows a built-up source buffer to drain while awaiting feedback from the network, especially on arrival of a large frame. Subject to achieving our quality targets, the smaller we can make β the better, since this will use less network resources.

In the results reported so far, we have used a β of 1.05. In this subsection, we examine the sensitivity to β, using

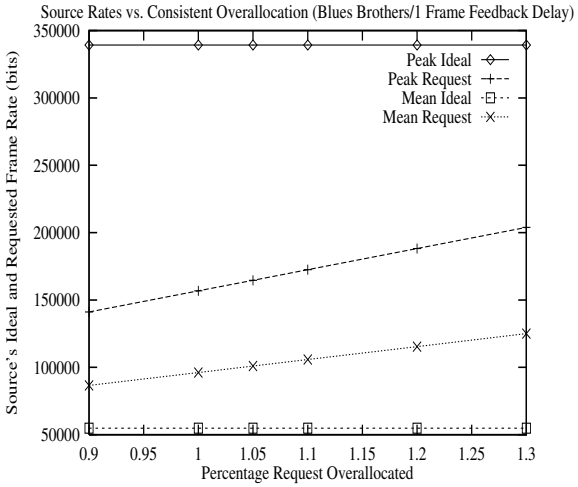


Figure 16. Behavior of the Requested Rate for Trace A for varying extent of over-request

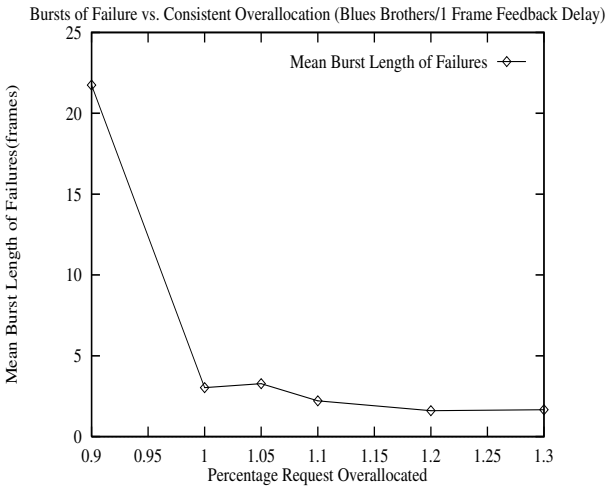


Figure 17. MEAN BURST LENGTH OF CROPPING $>$ 20%. Trace set A, for varying extent of over-request.

trace A, varying β from 1.3 down to 0.9, especially with respect to the extent to which frames are cropped. When β is 1.0, this implies that there is no over-request. For β of 0.9, we are persistently asking for only 90% of the required rate, causing a buildup at the source buffer. This has the potential of degrading the quality, since it would trigger the reduction in the target rate to the encoder and result in cropping the frame.

Figure 16 shows the mean and peak for the requested rate, with varying β. As expected, the peak request rate increases linearly with β. The mean requested rate has a slightly lower slope reflecting a small amount of short-term smoothing that occurs with the source buffer. Also shown in the figure are the mean and peak values for the ideal rate for comparison. As observed before, the requested rate is higher than the long-term mean ideal rate, but substantially lower than the peak ideal rate. However, we believe that the increase in the requested rate is small, even with β of 1.3, but the improvement in the quality we observe appears to be worthwhile.

Examining quality, when β = 1.0, Table 4 shows that the proportion of frames suffering cropping greater than 20% is 0.18%, slightly above our target of 0.1%. However, even when we consistently only ask for 90% of the rate required, the percentage of frames that suffer a cropping greater than 20% is still only 2%. In fact less than 9% of the frames suffer *any* cropping at all. SAVE also doesn't allow the mean delay in the source buffer to grow excessively. With β of 0.9, the mean source delay is 34.2 milliseconds, still quite reasonable.

We also examined the impact of β on the dynamics of cropping, shown in Figures 17 and 18. The mean burst length of cropping $>$ 20% is relatively small, ranging from less than 1 to about 3 frames, as long as β is 1.0 or greater. The lengths of these bursts are relatively insensitive for β greater than 1.0. However, when β =0.9, we see that the mean burst length of cropping $>$ 20% grows quickly to over 20, indicating that the quality may not be acceptable. Thus, it is useful to look more carefully into the dynamic behavior of SAVE, to show that in fact we are impairing quality by bringing β down to 0.9.

It is revealing to examine the effect of β on the length of the complementary bursts of consecutive frames cropped less than 20%. Increasing β from 1.0 to 1.3 dramatically increases the mean complementary burst length from near 2000 frames (a duration of over 1 minute) to nearly 14,000 frames (over 7 minutes). Getting β down to 0.9 results in a mean length of complementary bursts of about 250 frames. This is comparable to the mean burst length of frames cropped $>$ 20%, of nearly 25 frames. We believe that there is a benefit in having a $\beta \geq 1.0$, so that we can have a long enough period of almost unimpaired video.

We saw in Section 6 that SAVE is reasonably robust with respect to statistical reduction in the allocated rate that occurs as a result of network congestion. The results of this section show that SAVE is also robust with respect systematic rate-reductions, when the allocated rate is lower than the requested rate.

8 Conclusions

In this paper we have examined the quality that can be achieved with an online source smoothing algorithm for compressed video, called SAVE. The adaptation of com-

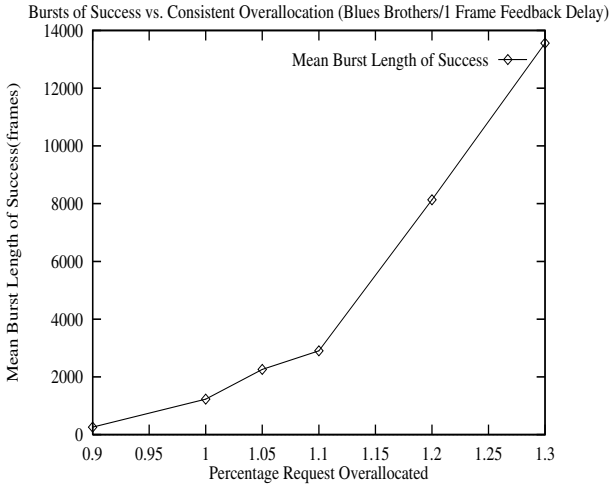


Figure 18. MEAN BURST LENGTH OF CROPPING $<$ 20%.. Trace set A, for varying extent of over-request

pressed video is based on negotiation of the bandwidth with the network using feedback-based congestion control and adapting the quantization parameter when necessary. For this form adaptation, it appears necessary to evolve the quality metrics typically used to evaluate the efficacy of mechanisms to transport video. The metrics relate to how frequently we have to adapt the video and by how much. We are quite stringent in setting quality targets for SAVE, suggesting that no more 1 in 1000 frames suffer more than a 20% cropping. Further, we examined the burstiness of the impairment, because we believe that having a long string of consecutive frames that are cropped results in noticeable degradation in quality. Our paper examined how effective SAVE is, in meeting our quality targets and builds on the results reported in [2]. We also looked at the "traditional" measures of effectiveness, in the context of our algorithm: the multiplexing gain, the source buffer delay and the behavior of the rate requested from the network.

We believe our algorithms need to be robust to variations in the feedback delay and to network congestion, failing which we would see buffer occupancy at the source increase. We find that SAVE's adaptation based on smoothing the short-term variations in ideal frame rate (of the order of a GOP) and tracking the maximum ideal frame size over a scene time-scale allows us to be relatively insensitive to feedback delay.

Smoothing and aggregation of flows help reduce variability in the rate requested of the network. When we go to a 38 fold aggregation of the traces in set E, the various quantiles of the requested rate (ranging from 100% to 50%) become quite close to the mean aggregate requested rate. (Although we don't detail it here, the same convergence of quantiles holds for the ideal rate; see [2]. But we find that convergence to the mean is faster for the requested rate.) In addition, SAVE tolerates a reasonable amount of reduction in the rate allocated to the flows. With a 5 fold aggregation of set E, we see that we can tolerate the situation where the capacity of the network is as low as 70% of the peak requested rate (in this case), and still maintain acceptable quality.

We examined the sensitivity to network congestion by considering the effect on one trace (trace A) when there are 10 other traces (from set E) that are multiplexed together. Even when the allocated rate from the network is 90% of the requested rate, we are able to maintain quality in all the different dimensions: delay, the overall proportion of frames that are cropped more than 20% and the size of the burst of frames that are consecutively cropped.

Taking this 90% of the requested rate being allocated by the network as a reasonable rule of thumb for SAVE to compare multiplexing gain, we compared the benefit of the rate-adaptive video (SAVE) with a source-buffered CBR model. We see that for equivalent quality, the bandwidth required by SAVE (90%ile) is 60% of the peak bandwidth required by a buffered CBR service. And this benefit of SAVE only seems to grow with aggregation size, a highly desirable characteristic. Moreover, SAVE has lower sensitivity to the bursty cropping of frames when the rate allocated by the network is less than the requested rate. Buffered CBR has a significantly larger burst of frames that would be cropped if the allocated rate from the network reduces below 90% of what is desired.

We demonstrated SAVE's sensitivity to its parameters - the target of the source buffer delay, the amount by which the source "over-requests" a rate from the network, and its smoothing windows. There is a tradeoff between the source target delay and the rate requested (especially the mean rate) of the network. The requested rate goes up as the target delay becomes smaller. But, SAVE is still able to maintain the quality targets with a smaller delay target - all the way down to 50 milliseconds. We showed that it is important to understand the dynamic nature of the quality reduction, where long bursts of frames are impaired. When the source systematically under-requests the rate needed from the network (asking for 90% of the rate needed by SAVE), it appears that the average "quality" measured over the length of the trace (trace A) doesn't seem to degrade much. But we suffer in having longer bursts of frames that are impaired in quality.

Finally, we showed that the most significant parameter in SAVE is the maximum smoothing window $w_{\max}$. The proportion of frames cropped decreases as a power law with $w_{\max}$, and the mean requested rate goes up logarithmically with $w_{\max}$. $w_{\max}$ serves as a dial that we may use to trade-

off between quality and the average of the rate we request from the network. The short-term smoothing window, w_{sm} helps us in reducing the peak requested rate from the network, without much impact on the quality. The maximum requested rate goes down as w_{sm} goes up to about a GOP, beyond which we see diminishing benefits. Thus, the combination of $w_{\max}$ and w_{sm} can be used manage the resources that we use from the network, while managing quality to be within acceptable limits.

We believe SAVE demonstrates considerable promise as a method of online rate-adaptation for compressed video.

Acknowledgments

We gratefully acknowledge the assistance of Partho Mishra in letting us use the network level simulator he has developed. We thank Frank Kelly, Peter Key, and the anonymous reviewers for their comments.

References

[1] A. Charny, K.K. Ramakrishnan, A. Lauck, "Time Scale Analysis and Scalability Issues for Explicit Rate Allocation in ATM Networks", *IEEE/ACM Transactions on Networking*, Vol. 4, No. 4, August 1996.

[2] N.G. Duffield, K.K. Ramakrishnan & A.R. Reibman, "SAVE: An algorithm for smoothed adaptive video over explicit rate networks", Proceedings of IEEE Infocom'98, San Francisco, April 1998.

[3] M.W. Garrett, W. Willinger, "Analysis, modeling and generation of self-similar VBR traffic", In *Proceedings ACM Sigcomm 94*,London, UK, August 1994, pp.269-280.

[4] B. Girod, "Psychovisual aspects of image communications", *Signal Processing*, vol. 28, pp. 239-251, 1992.

[5] M. Grossglauser, S. Keshav, & D. Tse, "RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic", Proceedings of the ACM SIGCOMM'95 Conference, Sept. 1995.

[6] H. Kanakia, P.P. Mishra, A.R. Reibman, "An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport", Proceedings of the ACM SIGCOMM'93 Conference, Sept. 1993.

[7] T.V. Lakshman, P.P. Mishra, K. K. Ramakrishnan, "Transporting Compressed Video over ATM Networks with Explicit Rate Feedback Control", Proceedings of the IEEE Infocom 1997 Conference, Kobe, Japan, April 1997.

[8] T. J. Ott, Lakshman T. V., Tabatabai A., "A Scheme for Smoothing Delay Sensitive Traffic Offered to ATM Networks", Proceedings of IEEE Infocom 1992, pp. 776-785, May 1992.

[9] A.R. Reibman and A.W. Berger, "On VBR video teleconferencing over ATM networks," in IEEE Global Telecommunications Conference (GLOBECOM), December 1992.

[10] A. R. Reibman and A. W. Berger, "Traffic descriptors for VBR video teleconferencing over ATM networks", *IEEE/ACM Trans. on Networking*, vol. 3, no. 3, pp. 329-339, June 1995.

[11] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, D. Towsley, "Online smoothing of live, variable bit-rate video", Proceedings NOSSDAV, 1997.

[12] O. Rose. "Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems". University of Wuerzburg. Institute of Computer Science Research Report Series. Report No. 101. February 1995.

[13] "ATM Forum Traffic Management Specification Version 4.0," af-tm-0056.00, ATM Forum, April 1996, available as: ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.pdf

[14] James D. Salehi, Zhi-Li-Zhang, James F. Kurose, Don Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing", Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, pp. 222-231, May 1996.

Real-Time Reliable Multicast Using Proactive Forward Error Correction*

Dan Rubenstein, Jim Kurose, and Don Towsley
Computer Science Department
University of Massachusetts at Amherst
{drubenst, kurose, towsley}@cs.umass.edu

Abstract

Real-Time reliable multicast over a best-effort service network remains a challenging research problem. Most protocols for reliable multicast use repair techniques that result in significant and variable delay, which can lead to missed deadlines in real-time scenarios. In this paper we present a repair technique that combines forward error correction (FEC) with automatic repeat request (ARQ). The novel aspect of the technique is its ability to reduce delay in reliable multicast delivery by sending repairs proactively (i.e., before they are required). The technique requires minimal state at senders and receivers, and no additional active router functionality beyond what is required by the current multicast service model. Furthermore, the technique uses only end-to-end mechanisms, where all data and repairs are transmitted by the data-originating source, leaving receivers free from any burden of sending repairs. We simulate a simple round-based version of a protocol embodying this technique to show its effectiveness in preventing repair request implosion, reducing the expected time of reliable delivery of data, and keeping bandwidth usage for repairs low. We show how a protocol using the technique can be adapted to provide delivery that is reliable before a real-time deadline with probabilities extremely close to one. Finally, we develop several variations of the protocol that use the technique in various fashions for high rate data streaming applications, and present results from additional simulations that examine performance in a variety of Internet-like heterogeneous networks.

1 Introduction

Multicast has become an important component of the Internet within the past decade. Deering's work [1] describes a framework for distributing data to multiple receivers via *multicast groups*. When multicast groups grow large, simple reliable multicast protocols suffer from a condition known as *feedback implosion*: an overload of network resources due to many receivers trying to send repair requests (henceforth referred to as NAKs) for a single packet. A number of solutions exist to avoid this implosion effect, using techniques such as randomized timers, local recovery (receivers sending repair packets), and hierarchical recovery. While such techniques are effective in providing reliability without implosion, they can result in significant and unpredictable delays, making them unsuitable for applications that have stringent real-time constraints.

In this paper, we present a technique that uses a novel combination of forward error correction (FEC) and automatic repeat request (ARQ) to reliably deliver data, with an emphasis on reducing delay and meeting real-time constraints without using randomized timers, local recovery, or hierarchical recovery. We call this technique *proactive FEC*, because it forwards error correcting packets into the network prior to their necessity.[1] It is this idea of having the data source forward repairs before they are required that differentiates our work from previous reliable multicast protocols that make use of FEC.

Our results through simulation indicate that this technique 1) offers a significant decrease in the expected time to reliably receive data, 2) can be used to meet hard real-time deadlines of reliable delivery of data with probabilities extremely close to one, 3) reduces feedback implosion, 4) competes well with other protocols in terms of required network bandwidth, 5) can reduce bandwidth requirements for meeting real-time deadlines compared to non-proactive approaches, and 6) scales to groups containing thousands of members configured in various topologies with various spatial and temporal loss characteristics. We also show that the technique functions well for high-rate data transfers in

*This material was supported in part by the National Science Foundation under Grant No. CDA-9502639, NCR-9508274, and NCR-9527163. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[1] The term is actually somewhat redundant, since the *F* in FEC implies the sending of information before its necessity. However, FEC in a multicast context commonly refers to sending encoded repair packets, as opposed to direct retransmission of the data, in response to NAKs.

networks containing large multicast groups, where receivers are at varying distances from the sender, with varying end-to-end loss rates and latencies.

The remainder of this paper is structured as follows. Section 2 gives an overview of related work, followed by a brief description in Section 3 of the coding method used to provide forward error correction, and its application within the domain of reliable multicast. The technique used to reduce delay and provide hard real-time guarantees is described and evaluated in Section 4, followed by a description and examination of low latency, reliable protocols in section 5. Real-time, probabilistically reliable (i.e., resilient) protocols are presented later on in this section as well. Section 6 further examines performance of our protocol as we vary a wide variety of network features. We conclude the paper in Section 7.

2 Related Work

In this section we discuss previous work that addresses various issues that arise when attempting to reliably multicast data in the Internet. To reduce NAK and repair transmissions, reliable multicast protocols have incorporated random delays [7]. However, this comes at the cost of increased latency. A variety of approaches have been proposed that limit the bandwidth used by NAKs and repairs, including scoping [7], and communicating via unicast to nearby receivers [10, 11] or designated repair servers [8]. Using such approaches, repair latency and bandwidth depend heavily on the location of both the repair entity and the point of loss within the network: the benefit is reduced as their distance to the receiver increases. Another approach restricts the number of entities that provide immediate feedback. This is used in [15], with the data source periodically choosing a small set of representative receivers that have priority in sending feedback. Here, the protocol's effectiveness depends on the source's ability to select a good set of receivers with its limited knowledge of the group topology and network loss characteristics.

There has also been recent interest in providing *resilient multicast* service for real-time data, where retransmissions occur only if data can be delivered before the real-time deadline. Data is not reliably delivered, but a higher good-put can be achieved than without any retransmission. Two protocols that that are designed to provide resilient multicast are STORM [10] and LVMR [12]. Both approaches form virtual trees with the source as the root and receivers as internal and leaf nodes. Recovery is implemented by sending all repair requests and retransmissions via unicast along this tree. Repairs can be performed with low latency provided that they do not need to traverse numerous links within the receiver-based tree. However, substantial delays can occur when losses occur close to the source. In such cases, the transmission path to a receiver can be significantly longer than the multicast route direct from the source. This is because repairs occur as a series of unicast transmissions between receivers.

Additional functionality within routers can also improve real-time reliable multicast performance. Several mechanisms have been suggested as a means of improving reliability [16, 9, 18]. However, this is at the cost of additional router state and / or processing.

Forward error correction (FEC) [3] is a technique that reduces the bandwidth overhead of repairing errors or losses in bit streams. It has been shown in [6] that a hybrid approach combining FEC with ARQ can significantly reduce bandwidth requirements of a large reliable multicast session. Hybrid approaches that combine FEC and ARQ have been proposed and classified for repairing loss and noise at the bit-level [23]. The Type I Hybrid approach involves sending repair bits within a packet that can correct bit errors or erasures, and retransmitting packets if the number of repair bits is insufficient to reliably receive the packet. Type II Hybrid approaches place the repair bits in packets that are distinct from the packets that contain the data. Our approach is considered to be a Type II Hybrid approach. However, our forwarding of repair packets before their necessity incorporates certain favorable features of a Type I approach as well.

[17] presents a preliminary analysis that compares the benefits of combining local recovery with an FEC/ARQ hybrid technique, and concludes that for many multicast scenarios, such a combination offers little improvement over an FEC/ARQ hybrid technique without local recovery. [13] compares real-time performance of reliable multicast techniques that use FEC to those using ARQ techniques, but does not consider hybrid approaches. An interesting approach using FEC is presented in [14], where data is delivered reliably without requiring ARQ through multicast group joins and leaves. A recent implementation utilizes ARQ as a last resort to obtain any data that could not be obtained via the joining of a particular multicast group. Present join-leave latencies for multicast groups make the approach too bandwidth inefficient to support real-time applications. Other works are showing the promise of the proactive FEC technique. One such work is [21], in which a round-based proactive FEC approach is combined with a receiver-based hierarchical recovery scheme.

3 Overview of Forward Error Correction (FEC)

Error correcting codes were initially applied in domains where bits could be erroneous or missing, but have more recently been applied to repairing packet losses at the net-

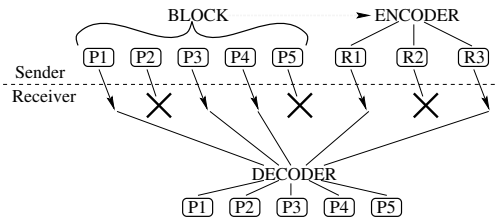


Figure 1: A sample decoding using a block built from 5 packets.

work layer. The following description is sufficient for understanding how systems, equipped with Reed-Solomon encoders and decoders, can make use of repair packets to recover from loss. The sender forms **blocks**, where each block consists of a subset of the data packets it wishes to deliver reliably. The number of data packets that are used to form a block is commonly referred to as the *block size*, k. The sender inputs the k packets into its encoder which then generates *repair packets* for that block. A receiver uses its decoder to recover the k data packets from any combination of k distinct packets that use data packets from the block, and/or repairs generated for the block. An example is shown in Figure 1, where a sender groups 5 data packets into a block, encodes 3 repair packets from this block, and transmits all 8 packets to the receiver. As soon as the receiver receives any 5 distinct packets related to the block (in the example, 3 data and 2 repair), it activates the decoder and recovers the lost data packets.

Detailed discussions of Reed-Solomon coding techniques and of packet-level FEC techniques can be found in [4] and [6] respectively; implementation issues are considered in [5, 6]. For our purposes here, we simply note that FEC techniques exist that can be used to generate as many repair packets as needed, and that this can be done at data rates on the order of 8 Mbytes/sec on commodity PC's.

4 A Proactive FEC+ARQ Technique

In this section, we present a proactive technique that delivers data reliably to a set of receivers through a combination of ARQ and FEC, and examine the impact that proactivity has on the performance of reliable data transfer to large multicast groups. The technique is *proactive* in that it allows the sender to send repairs for packet losses prior to receiving any indication that the repairs are necessary. This allows receivers to tolerate a certain amount of packet loss and still receive all data reliably without requiring retransmissions. As a result, the average number of receivers that require retransmissions decreases, reducing implosion. We will also observe that proactive FEC can reduce the number of repair packets that are needed.

We first examine the proactive FEC technique using an idealized round-based protocol. During each round, the sender sends data and repairs and awaits feedback from all receivers needing additional repairs. The sender waits for a response from the slowest responding receiver before proceeding to the next round. As a result, the protocol introduces additional latencies in attempting to meet individual receivers' real-time deadlines. In Section 5, we study protocols that do not require such synchronization.

As in most protocols that use FEC, the sender packetizes its data for delivery, and builds blocks of k packets. For simplicity of presentation, we assume a fixed block size for the entire data stream, and that blocks are built from consecutive packets within the data stream.

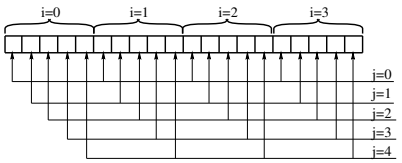


Figure 2: Partitioning a data set into 4 blocks of size 5 so that repairs can be generated using FEC.

Each packet transmitted by the sender contains a pair of identifiers (i, j), where i identifies the block, and j the position of the packet within the block. For the purpose of this paper, we assume that i and j are non-negative integers. The data packets within a block i are assigned values of $(i, 0)$ though $(i, k-1)$ corresponding to their ordering in the data stream. As repairs are required, the sender creates and sends them. For a block i, repair packets are assigned sequence numbers (i, j) with $j \geq k$. These sequence numbers are used by the decoder at the receivers' end to determine the operations that must be used to retrieve lost data. Using a block identifier within a packet permits the interleaved transmission of the various blocks, so that a transmission of a new block can commence before other blocks have completed their reliable delivery. Figure 2 gives an example of a data stream partitioned into four blocks each of size 5.

Associated with the protocol is a proactivity factor, ρ, which is a rational number larger than or equal to one. For a block of size k, the sender initially transmits $\lfloor \rho k \rceil$ packets,[2] consisting of the k data packets plus an additional $\lfloor (\rho - 1)k \rceil$ repair packets. No other repair packets are sent for the block unless receivers specifically request them.[3] Such requests are unicast by a receiver only when it fails to obtain the entire block of data, either through the reception of the original block, or through application of its decoder. Otherwise, a receiver need not send any feedback to the sender. In order to satisfy all receivers' requests, the sender responds to the NAKs by multicasting a number of repair packets that

[2]The notation $\lfloor x \rceil$ means round x to the nearest integer.

[3]Encoders such as the one presented in [5] are capable of generating a single repair packet at a time (i.e., it is not necessary to know in advance the number of repairs needed.)

satisfies the maximum from all of the receivers' requests for that round. This process continues until each receiver either obtains a sufficient number of packets to perform decoding or passes some deadline after which there is no point in retrieving the block.

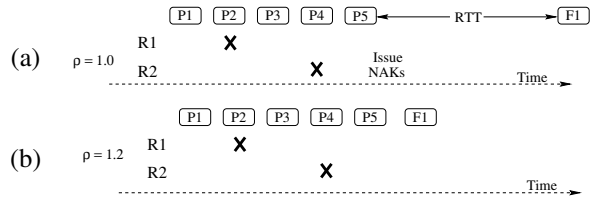


Figure 3: Illustration of possible savings in time and reduction of feedback through proactivity

Figure 3 demonstrates the potential benefits of setting the proactivity factor to a value larger than one. In Figure 3(a), two receivers, R1 and R2, are sent a block of five data packets, and each receiver loses one of these data packets. If $\rho = 1.0$, then no repair packets are sent with the initial transmission, and the receivers must NAK and wait for the sender to retransmit the data. Alternatively, in Figure 3(b), if $\rho = 1.2$, then a repair packet is transmitted with the data, and the receivers do not need to request repairs nor wait a full round trip time for the sender to transmit the repair. We point out that the round-based approach creates no interdependencies among the various blocks being transmitted, and so the sender can deliver multiple blocks at a given time.

Sender and receiver state diagrams depicting the idealized round-based protocol are given in Figures 4 and 5, respectively.

By sending repairs proactively, more receivers will obtain at least k packets in the initial round, resulting in the return of fewer repair requests to the sender. Thus, the sender can effectively control NAK implosion by sending an appropriate number of repairs proactively in the initial round. The sender benefits little from adding proactivity to subsequent rounds because the number of repair requests in each subsequent round is most often significantly smaller than the number of repair requests in the initial round. On the other hand, receivers continue to benefit when proactivity is added in subsequent rounds, since it can often reduce the number of rounds needed to reliably receive a block.

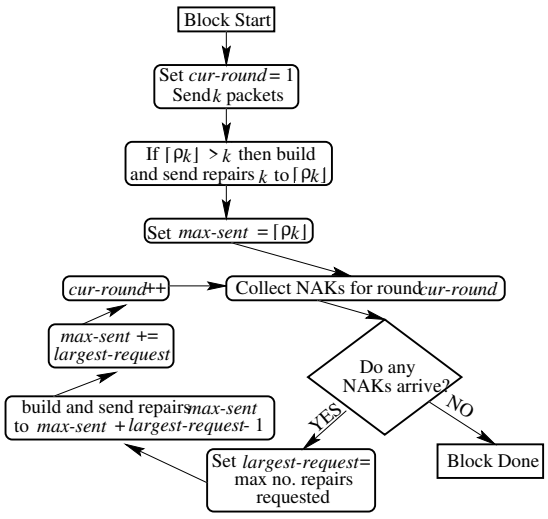


Figure 4: The round-based sender algorithm

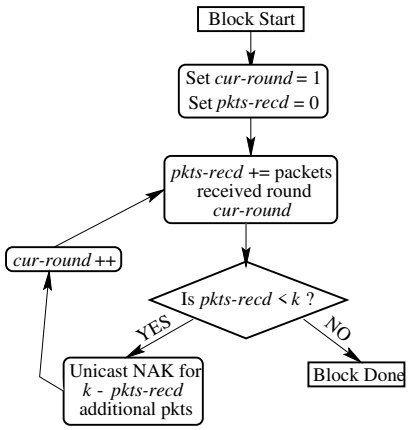


Figure 5: The round-based receiver algorithm

Receivers can control the amount of proactivity used in subsequent rounds by requesting more repair packets than are needed. This allows each receiver to request a level of proactivity that best satisfies its own requirements. The amount of needed proactivity can be determined in several ways. For instance, a receiver could apply the sender's proactivity factor to its own request, and if it needs n packets, request $\lfloor \rho n \rceil$ packets. Alternatively, it could simply request an additional packet so that it can tolerate the loss of one repair packet, or if it knows the loss rate from the sender to itself, it can request a number of repairs such that it can obtain a sufficient number of repairs with some fixed probability. In Section 4.1, we show how receiver-initiated proactivity can be used to allow each receiver to meet its own hard deadline.

4.1 Meeting hard deadlines

If a receiver can estimate the loss rate from the sender to itself, it can request more repairs for a block than the minimum number needed in order to perform decoding. Doing so allows the receiver to meet a deadline with probabilities that are extremely close to one. To do so, the receiver must be able to calculate the probability of receiving at least n packets out of a group of m packets. We represent this value by $\mathcal{D}(m, n)$. If the loss is modeled as a temporally independent loss process with packet loss probability p, then we

have:

$$\mathcal{D}(m,n) = \sum_{i=n}^{m} \binom{m}{i} (1-p)^i p^{(m-i)}.$$

Since $\lim_{m\to\infty} \mathcal{D}(m,n) = 1$ (a simple proof is presented in [25]), a receiver that needs n packets can guarantee that the packets are received with a probability larger than any $\mathcal{P} < 1$ by determining an appropriate m that satisfies $\mathcal{D}(m,n) \geq \mathcal{P}$. A search for an appropriate m can be performed quickly, since $\mathcal{D}(m,n)$ satisfies the recurrence relation:

$$\mathcal{D}(m,n) \quad = \quad \mathcal{D}(m-1,n) + \binom{m-1}{n-1} (1-p)^n p^{m-n}.$$

with the initial condition $\mathcal{D}(n,n) = (1-p)^n$. The computation of $\mathcal{D}(m,n)$ for a two-state loss model is presented in the [25].

A receiver can use a conservative estimate of the length of a round to determine the last round by which it must receive repairs in order to meet a deadline. If the receiver still needs n more repairs upon entering this last round, it makes a request for m repairs, choosing m large enough so that sufficient repairs arrive with a high enough probability.

We present two types of guarantees that receivers can make to meet a hard deadline:

Last round guarantee. Here, the receiver guarantees that if a last round is necessary, then enough repairs will be delivered in that round to insure that the conditional probability of being able to decode all packets in the block, given the number of packets still needed before starting the last round, is greater than $\mathcal{P}$. To make this guarantee, the receiver simply chooses m such that $\mathcal{D}(m,n) \geq \mathcal{P}$, where it needs n packets going into the last round.

Block good-put guarantee. Alternatively, the receiver may wish to achieve some overall block good-put rate, such that the probability that a block is received on or before the last round is $\mathcal{P}$. We prove in the Appendix that if a receiver needs n packets going into the last round, it is sufficient to choose m such that $\mathcal{D}(m,n) \geq (\mathcal{P} - \mathcal{D}(l,k))/(1 - \mathcal{D}(l,k))$, where l is the number of packets sent over all previous rounds, and k is the block size. Simple algebra reveals this to often be smaller and never larger than what is required to meet the last round guarantee. If $\mathcal{D}(l,k) \geq \mathcal{P}$, then no attempt is made on the last round to retrieve the block. We emphasize that these results apply for any model of loss between source and receiver. The receiver must simply compute $\mathcal{D}()$ using the appropriate loss model.

By attempting to meet a hard deadline, receivers can request more repairs than it would otherwise, thereby increasing the number of packets that the sender transmits. When a hard deadline condition exists, added proactivity can reduce the expected delay of reliable receipt by receivers as well as the expected number of packets that are transmitted by the sender. This is because additional proactivity makes it more likely that receivers will have already obtained the entire block before entering the last round, thus obviating the need to request a large number of repair packets in order to meet the guarantee probability, $\mathcal{P}$.

4.2 Examination: round-based protocol

The focus of this paper is to demonstrate the effectiveness and simplicity of using the proactive FEC technique. Performance is measured against a traditional end-to-end approach where repairs are data packets which are multicast to the entire multicast group in response to a NAK. In effect, this traditional approach does not use FEC, and repairs are not sent proactively. Furthermore, the traditional approach does not scope repairs, so all receivers in the multicast group will receive each repair transmission (regardless of where the repair transmission originated from). We believe that our proactive FEC technique competes well with many protocols that use scoping approaches such as local recovery, additional router support, or multiple multicast groups. However, we avoid direct comparison for two reasons. First, benefits due to scoping are difficult to measure: they depend heavily on network and multicast group topologies, as well as on how the various amounts of increasing and decreasing traffic on various links affects protocol and network performance. Second, we believe it is possible to combine the proactive FEC technique with a scoped approach to further improve reliable multicast protocol performance. Since scoping reduces the effectiveness of FEC techniques, determining the right way to combine FEC with scoping is still an open problem.

We evaluate the performance of the round-based protocol through simulation for networks containing up to 10,000 receivers. Simulation allows us to examine a much richer set of network scenarios than is possible via a mathematical analysis, and makes it easier to observe effects of large multicast sessions than it would be through experimentation. We have performed an analysis of a star topology network; it appears in [25]. Results presented here are simulated over a model which we now describe. Other topologies, receiver placements, loss models, and block sizes are considered in depth in [25]. A brief summary of their impact is given at the end of this section.

The model used in the simulations presented here is a randomly generated tree network, where nodes on the interior of the tree represent routers with downstream fan-outs of one, two and three, with loss probabilities at each outgoing link uniformly distributed between 0.00075 and 0.00125. Nodes on the leaves of the tree that connect to the sender or receivers had downstream fan-outs ranging from 1 to 5 with loss probabilities uniformly distributed between

Amortized cost to deliver a packet reliably					
	Multicast Group Size				
Blocksize	1	10	100	1000	10000
1	1.09	1.62	2.49	3.35	4.21
5	1.14	1.36	1.60	1.82	2.04
10	1.16	1.29	1.43	1.56	1.70
15	1.11	1.25	1.37	1.46	1.56
20	1.12	1.23	1.34	1.41	1.48

Table 1: Comparison of the amortized, expected forward bandwidth used to transmit a packet reliably for various block sizes and multicast group sizes over the sample tree topology used in the experiments for this paper. There is no proactivity ($\rho = 1.0$). Note that a block size of 1 is identical to having no FEC, where each multicast retransmission is the sole packet within the block.

.0375 and .0625. These loss rates are similar to those observed in [19], and the fan-outs coincide roughly with what is observed within the Internet. These values also provide for realistic end to end properties within the Internet: the number of hops from sender to receiver varies from 3 to 28 with a mean hop-count of 15.75 and a mean end to end loss rate of .0896. The algorithm used to construct the tree is presented in [25].

Once a tree has been created with 10,000 leaves, where each leaf node represents a potential receiver, the number of receivers is varied by selecting a subset of leaf nodes in the tree of the desired size, and placing receivers only at these nodes. We have experimented with a set of approximately 20 randomly generated trees, and have found that the results vary little for the different topologies generated. For this reason, the results in this Section are computed using a single, randomly constructed tree.

We fix the set of active receivers that are used to examine a multicast group of a particular size. The set of receivers that are active in the smaller multicast groups are proper subsets of those receivers that are active in the larger multicast groups. We model loss at each router as a Bernoulli process, so that there is no temporal correlation between consecutive packets being forwarded. The block size is fixed at 10.

For each configuration described above, we performed several experiments on the same topology (around 20) and computed the average values of our measuring criteria to generate a distribution that was close to normal, and then used as many of these averaged values as needed to calculate 95% confidence interval widths that were within 5% of the point value.

We will evaluate performance using three general criteria:

Delay. The manner in which delay will be examined will depend on the goal of the protocol. For protocols that do not impose hard real-time deadlines, we will be interested in the expected delay of reliable delivery. For those that *do* impose hard deadlines, we will be interested in the expected percentage of blocks that can be received before the deadline expires. We elaborate further on the way we measure delay later in the paper.

Implosion factor. Here, we will be interested in the expected number of NAKs a sender receives from the receivers per block.

Forward bandwidth. This is the expected number of packets (repair and data) sent by the sender per block. Because the sender multicasts each packet it sends, and each packet traverses all links in the multicast tree (except links on the tree that lie below points of loss of a packet), the bandwidth usage in the network is roughly linear to this number of packets transmitted.

We begin by examining the effect of block size on the expected forward bandwidth used by any protocol that multicasts repairs to an entire group. We define the amortized (per packet) expected forward bandwidth to be the expected number of packet transmissions needed to deliver a block divided by the blocksize. Table 1 gives the amortized, expected forward bandwidth needed to reliably deliver a packet to all receivers as a function of receiver population and block size, where no proactivity is used ($\rho = 1.0$). The configuration used to compute the results for the table is the same as that used to compute the results in the remainder of this section. An examination of the table reveals that for large multicast groups, one obtains a significant reduction in expected forward bandwidth by increasing the block size. We note that a protocol using a block size of one is equivalent to a protocol that does not use FEC, where each repair can be used to repair a single loss. Thus, the top row represents the amount of forward bandwidth that would be used by a traditional end-to-end reliable multicast protocol.

Figures 6(a), 6(b), and 6(c) respectively show the expected number of rounds, implosion factor, and forward bandwidth per block for various multicast group-sizes as a function of the sender's proactivity factor. An increase in the proactivity factor decreases the expected number of rounds in a roughly linear manner. The implosion factor decreases exponentially as a function of the proactivity factor, so that NAK implosion can be reduced significantly with small increases in the proactivity factor. The most interesting result is the effect that the proactivity factor has on forward bandwidth. For large multicast groups, increasing the proactivity factor up to a certain value has an insignificant effect on the expected number of packets transmitted to reliably deliver data to all receivers. We note that for a block size of 10 and a receiver group-size of 10,000, if the sender's proactivity factor is 1.6 then the mean number of rounds is 2, the implosion factor is approximately 10 NAKs, and the forward bandwidth is 17 packets (10 original and 7

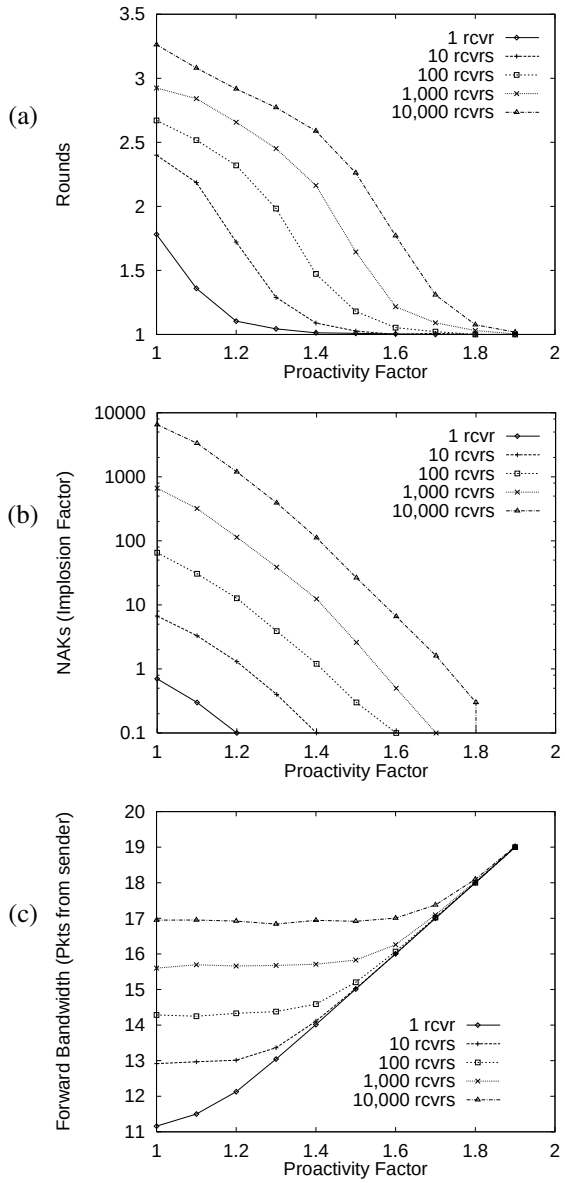


Figure 6: Results from a round-based simulation on a tree topology of receivers. Note that the y-axis of (b) is plotted on a log-scale.

repair packets). This compares favorably in every respect to an ARQ protocol without local recovery (i.e., a proactivity factor of 1.0), which would take close to 4 rounds and require the transmission of around 40 packets. In addition, the need for randomized delays to prevent NAK implosion would further increase the latencies associated with an ARQ protocol.

We can summarize these observations:

By using proactive FEC, we can reliably deliver data sooner than with traditional ARQ and with considerably less bandwidth. Furthermore, proactivity provides large decreases in repair bandwidth and delivers data reliably sooner when compared to a non-proactive FEC-ARQ hybrid approach. The proactive approach achieves these gains without using significantly more forward bandwidth than its non-proactive counterpart.

This is because with a large group, a certain number of repairs must be expected. With proactive FEC, these repairs are simply sent before it is known for certain that they are needed. Note that after some point, however, the bandwidth begins to increase along the asymptote $y = kx$, where k is the block size.

An alternate view of the data in Figure 6 is provided in Figure 7. Here, we directly plot the tradeoff between forward bandwidth and implosion factor (Figure 7(a)) and forward bandwidth and rounds (Figure 7(b)). Each curve is obtained by varying the proactivity factor. We see clearly from the flat portions of the curves that the proactivity factor can be chosen so that the implosion factor and number of rounds (delay) are low, without significantly increasing the amount of forward bandwidth required.

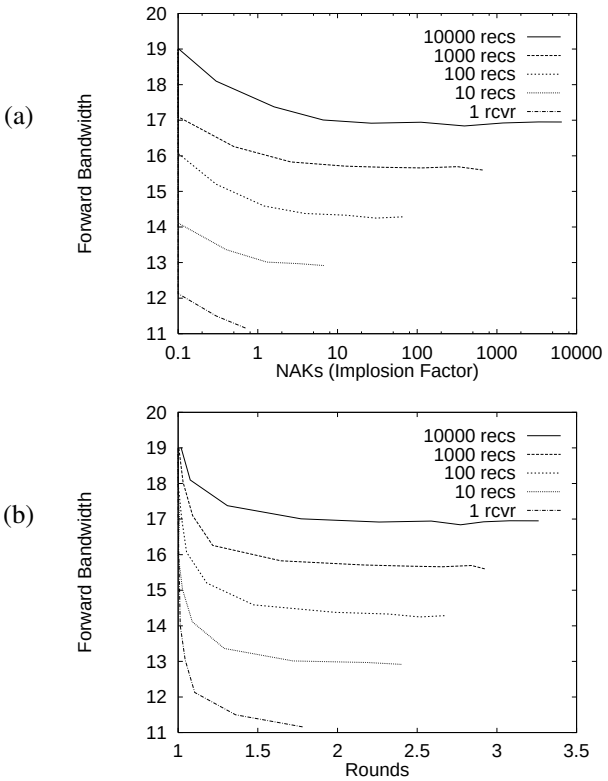


Figure 7: Tradeoff between forward bandwidth and NAK implosion and number of rounds.

We now examine the performance of the protocol in the presence of real-time constraints in the same network. Recall that in Section 4.1 we presented two ways in which re-

ceivers can use proactivity to guarantee a certain type of deadline-driven reliability. Figure 8(a) and (b) respectively show the expected forward bandwidth needed to meet a last round guarantee and a block good-put guarantee where $\mathcal{P} = 1-10^{-6}$, with all receivers requiring the data in two rounds. In other words, each receiver only participates in 2 rounds, sends at most a single NAK, and is able to meet its desired guarantee with probability greater that $1 - 10^{-6}$.

It is interesting to note that adding proactivity can actually *decrease* the expected amount of bandwidth required in order to make a last round guarantee. In contrast, a block good-put guarantee uses the least amount of bandwidth when the proactivity factor is 1.0, though the bandwidth increases slowly as a function of the proactivity factor until it starts to increase along the asymptote $y = kx$. Another interesting fact is observed by comparing the expected forward bandwidth in Figure 6(c) to Figure 8(b). We see that meeting a block good-put guarantee can actually reduce the expected forward bandwidth used in a session, even when the guarantee is a rate as high as $1 - 10^{-6}$. This is due to the fact that the receiver aborts all further attempts to obtain a particular block after some final round. The forward bandwidth savings seems to more than compensate for the additional forward bandwidth that is used on this final round to meet the guarantee.

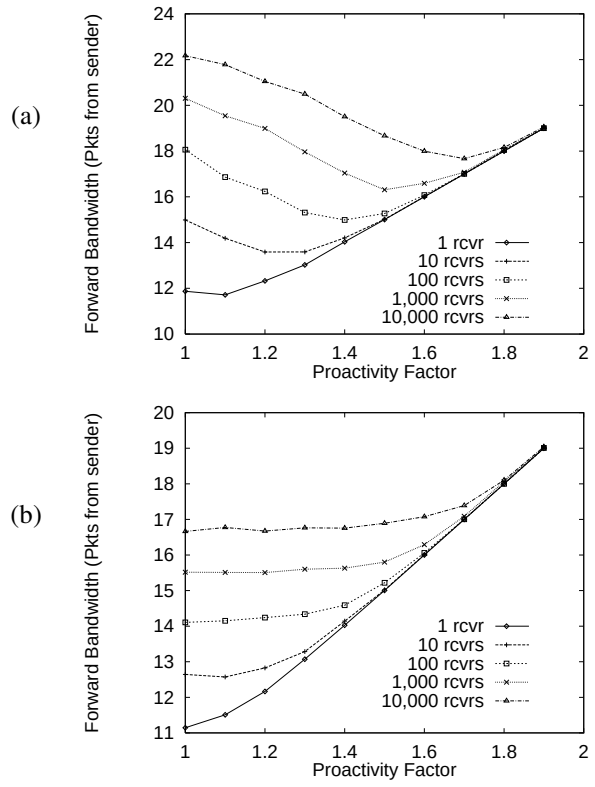


Figure 8: Bandwidth usage to provide (a) last round and (b) block good-put guarantees of $1 - 10^{-6}$ by the end of 2 rounds.

In addition to the results described above, we have examined the performance of the round-based proactive FEC protocol in a large variety of alternative network scenarios, including a variety of heterogeneous and homogeneous network topologies to examine the effects of various spatial loss correlations among the receivers. We also considered models where losses on links are similar to what is observed in [20] (i.e., various links in the backbone have high loss rates, links near the edges have low loss rates), and we examined the impact of bursty loss using 2-state loss models at routers. Finally, we also examined various ways in which receivers could add additional proactivity by sending NAKs requesting additional repairs beyond the minimum that would be necessary to complete the block. The observations made from these additional experiments vary slightly from the results presented in this section. The technique was still effective when using the loss model observed by Handley in [20] for two reasons. First, receivers that experienced high losses required high levels of proactivity. Second, the drop in loss at the link next to the source made using FEC more effective, even though there was an increase in loss (which also caused correlated loss at receivers) in the backbone. The only drawback is that the disparity in receiver loss rates caused receivers with lower loss rates to typically receive many more repairs than they actually require, in order to satisfy the high repair needs of receivers with high loss rates. In general, we found that the proactive FEC technique was less effective for very low loss rates, very bursty loss, and/or highly correlated (i.e., upstream) loss. Details are provided in [25].

5 Asynchronous Protocol

In Section 4, we demonstrated the effectiveness of the proactive technique in a heterogeneous network using a round-based protocol. Such a protocol adapts its reliable delivery rate to the requirements of the slowest receiver, reducing the protocol's effectiveness for faster receivers. We now examine a protocol that eliminates the synchronous behavior imposed by the use of rounds. This allows each receiver to communicate with the sender at a rate that is independent of the number or locations of other receivers. We assume that all receivers multicast their NAKs for the purposes of suppression. In [25], we show that this does not provide considerable benefit over unicasting NAKs.

We are able to allow receivers to operate in an asynchronous fashion by changing the format of the information that each receiver reports in its NAK. Other hybrid FEC/ARQ approaches require each NAK to request the *number of additional repairs* needed by the receiver in order to decode a particular block. We take advantage of the fact that repair packets must contain sequence numbers for decoding purposes, and perform the following subtle change to the

way receivers NAK.

We use the packet sequencing format described in Section 4, where each data and repair packet contains two sequence numbers (i, j), where i indicates the block, and j the packet's its position within that block. NAKs also contain two sequence numbers (i, j) where i represents the block to which the NAK pertains. However, rather than j indicating number of additional packets desired, it indicates to the sender that it should send all data and repair packets for block i with sequence number less than or equal to j that it has not already sent. As we shall see below, performing NAKs in this fashion allows for a simple sender algorithm.

5.1 Protocol: Sender's algorithm

The sender's actions are simple and the only aspect that depends on network conditions is the value it chooses for the proactivity factor. A state diagram demonstrating the sender's algorithm for a block is given in Figure 9. The algorithm proceeds as follows: For each block i, the sender sends the data and a number of repairs which is determined by the proactivity factor. It keeps track of the sequence number (i, j) for the packet it has sent with the largest value for j. We refer to this value of (i, j) as the *largest sent sequence number for block* i, or *LSSN*. If a NAK arrives with sequence number (i, j'), $j' > j$, the sender sends all packets with sequence numbers $(i, j + 1)$ up to (i, j'), and (i, j') becomes the new value for the LSSN. It should be clear to the reader that multiple blocks can be transmitted in the same period of time, as long as the sender maintains the LSSN for each block that it is in the process of transmitting.

There are several advantages to having a sender use this simple protocol. First, the sender maintains one item of state per block: the LSSN. This makes it easy for the protocol to scale as the multicast group size grows, since the sender's state is constant with respect to the multicast group size. Second, the sender does not require any knowledge of the group topology. This is important because topology information is difficult to obtain and can vary during a session. Third, it operates in an event-driven manner, so that it doesn't need to maintain timers, except perhaps a single timer that expires when a block becomes stale and no longer requires buffering. Fourth, the sender always reacts to the repair request in a manner that satisfies the request. Alternatively, if each NAK uses the approach where it only specifies a number of repairs that are needed by the receiver, an additional burden is placed on the sender: it must determine for each arriving NAK whether or not it has already satisfied or partially satisfied that NAK's request. The answer depends on loss rates and propagation delays to the receiver that transmitted the NAK.

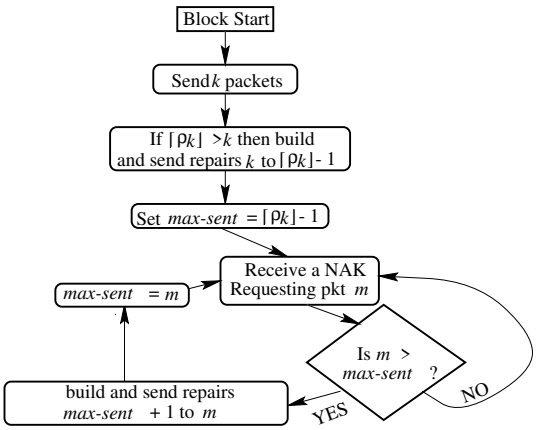


Figure 9: The sender protocol

5.2 Protocol: Receiver's algorithm

The state diagram that describes the receiver's behavior while receiving a block is shown in Figure 10. Each receiver reacts to three possible events: the arrival of a data or repair packet arrival, a NAK from another receiver, or a timeout. The way in which the algorithm reacts to these events is determined by two functions: the $NAKSeqno()$ function is used to calculate the sequence number that should appear in a NAK that is about to be sent. The $ComputeBlockNextTimeout()$ function calculates the point in time when the next timeout for a particular block should occur. After a timeout, and possibly after a data or repair packet arrival, a receiver sends a NAK requesting additional repairs, using the $NAKSeqno()$ function. After processing an event, the receiver recalculates the time at which it should timeout using the $ComputeBlockNextTimeout()$ function. It then blocks until it receives another packet or another timeout occurs, at which time it repeats the process. As with the sender, multiple blocks can be transmitted at the same time, as long as the receiver maintains an independent set of states for each block that is in transmission.

The receiver algorithm is more complicated than that of the sender. However, the algorithm need only maintain a small amount of per-block state information: the time of arrival of the most recently arrived packet from the sender, the maximum sequence number over all repairs received in the block, and the time of arrival and sequence number of the NAK received with the maximum sequence number. Additionally, the receiver must maintain an upper bound on the round trip time to the sender. To meet hard guarantees, additional state is required: an upper bound on the loss rate from the sender. It should be clear that, similar to the sender's algorithm, the receiver state neither depends on the group size nor on the topology of the other receivers in the network.

Because NAKs contain intra-block sequence numbers instead of the number of repairs requested, the receiver must

calculate the appropriate sequence number in order to have the sender send the appropriate number of repairs. This sequence number can be computed at the receiver by determining the desired number of repairs the sender must transmit to allow for decoding the block, and adding this value to the receiver's estimate of the sender's current LSSN for the block.

Due to loss and propagation delays, a receiver does not always accurately determine the sender's current LSSN. An overestimate or underestimate of its value respectively causes an overestimate or underestimate in the desired sequence number in the NAK. An overestimate can occur due to a previous NAK that was processed or sent by a receiver, but was lost by the sender. It can also occur if repairs incur larger than expected delays so that the receiver prematurely considers the repair lost. An underestimate can occur if the receiver loses a NAK from another receiver that reaches the sender. We note that the receiver can adjust its estimate with each packet arrival, so that the estimates are not likely to deviate dramatically from the true sender value. Also, there are several techniques that can be employed to reduce the damage that can occur due to inaccurate estimates. See [25] for details.

To prevent unnecessary NAK transmissions, after receiving a NAK, a receiver waits a conservative amount of time before reacting to a loss. When at some time t, a receiver R receives a NAK from another receiver, R', it must then give the sender an appropriate amount of time to respond to the NAK. By waiting until time $t + r$, where r is the round trip time from R to the sender, R gives the sender enough time to provide repairs based on the NAK from R'. The approach is conservative, since often the sender's repairs will arrive sooner than $t + r$.

We have experimented with two versions of the receiver algorithm that differ only in a variation of the $ComputeBlockNextTimeout()$ function. The first these, referred to as the *patient receiver protocol*, only issues a NAK for a block when it determines (with the information available to it) that all repairs issued thus-far have already been received or lost, and no NAKs are en route to the sender that will trigger additional transmissions. The second variation, referred to as the *aggressive receiver*, sends a NAK whenever it determines that the sender's current LSSN is not large enough to send enough repair packets to permit decoding for the block. Figure 11 illustrates the difference in the reaction rates of the patient and aggressive receivers.

5.3 Performance Evaluation

We now report results from simulation of the asynchronous protocol for multicast groups containing 500 receivers. The simulation was written using the TeD network simulator [22]. The group size is restricted to 500 due to the memory

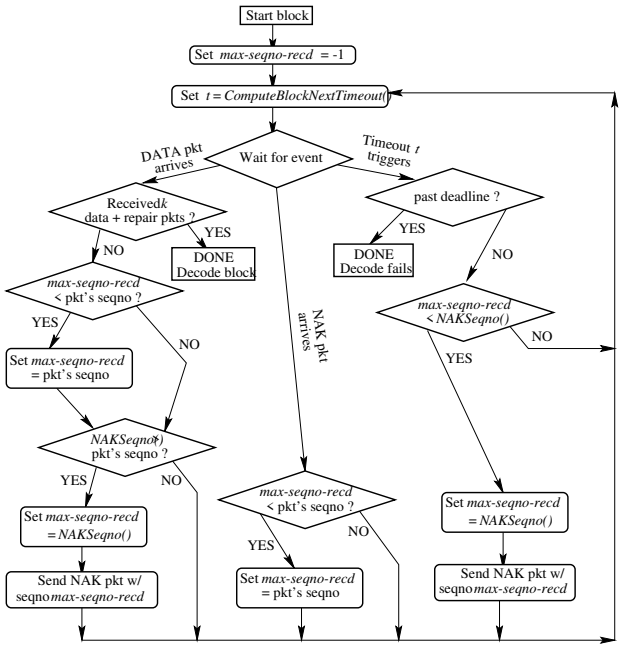


Figure 10: Receiver protocol: the different variations are constructed using different functions for $NAKSeqno()$ and $ComputeBlockNextTimeout()$.

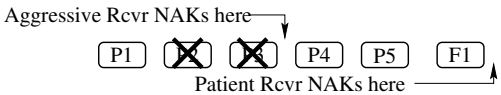


Figure 11: Patient vs. Aggressive Receiver: If the block size is 5 and the proactivity factor is 1.2, a patient receiver will allow time for receipt of all 6 transmissions of the initial packets before sending a NAK, regardless of which packets are lost. An aggressive receiver will NAK as soon as it detects 2 losses, since it knows it cannot recover the block with only 4 packets.

and time constraints imposed by the multicast group size in such a simulation.

The results presented here used a randomly constructed tree topology, where the minimum, average, maximum hops to the sender were 7, 15.1, and 25 respectively. The topology was constructed in a similar manner to the trees used in the round-by-round simulation in Section 4. We use a single tree for all multicast transmissions, including receiver NAKs. This corresponds a routing protocol like CBT. The results from our simulation are likely to be pessimistic for non-CBT-like routing, which would likely result in reduced NAK propagation delay between receivers.

We have experimented over a small set of such randomly generated topologies. With group sizes containing hundreds of receivers, we have observed similar results over the various random topologies. This is not surprising, since distance to the sender is the factor that we expect to have the largest impact on performance. Since we randomly select a

End Host processing rates used in simulation		
Operation	Time (μsec)	Performed by
Data Send	502	S
Data Process	487	R
NAK Send	87	R
NAK process	86	S & R
FEC code 1 packet	180 * block size	S (build) R (decode)
Data rate	5208	S

Table 2: Processing rates used in the simulation. An 'S' means the operation is performed by the sender, an 'R' means by each receiver.

large set of receivers within a randomly generated tree, their distances to the sender form a uniform distribution. All results presented in the remainder of the paper are based on a single, randomly generated tree topology.

Here we compare the performance of the patient receiver and aggressive receiver protocols in a simulation where the sender's proactivity factor remains fixed for the entire session, and examine how the proactivity factor affects session performance. First and last hop loss rates varied between 2% and 4% and backbone loss rates varied between 0.05% and 0.1% per router, giving end-to-end loss rates between 4.4% and 10.4%. Each router added a delay to a packet passing through it, where this delay fell in a range $[x, y]$: x was chosen from a uniform distribution between 5 ms and 6 ms for backbone routers and 0.5 ms and 3.5 ms for last hop routers. y was then chosen by adding a uniformly distributed value to x between 0 and 5 ms for backbone routers, and between 0 and 10 ms for last hop routers. To prevent out of order delivery of packets, a packet would always be delayed beyond its chosen value to prevent its arrival on a link before a previously injected packet. In other words, if packet A arrives before packet B at a router, and scheduled departure times are respectively t and t' with $t' < t$, then B is rescheduled to depart at time t plus 170 μs. Maximum round trip times varied between 74 ms and 294 ms for the various receivers.

Figure 12 gives results from 6 different runs of our simulation that run for 5 seconds of simulated time with a data rate of 1.5 Mb/s using packets of one kilobyte each (roughly 93 blocks, or 930 data packets are transmitted). In each simulation, 500 receivers appear in an identical network configuration. In the first three simulations, all receivers execute the patient protocol and the sender's proactivity factor is fixed for each simulation at a level of 1.0, 1.3, and 1.6. For the last three simulations, all receivers execute the aggressive protocol, and the proactivity factor is again set at a level of 1.0, 1.3 and 1.6. The protocols are run reliably, meaning that receivers continue to send NAKs until they have received all of the data reliably. End-host processing rates are presented in Table 2, and are taken from measure-

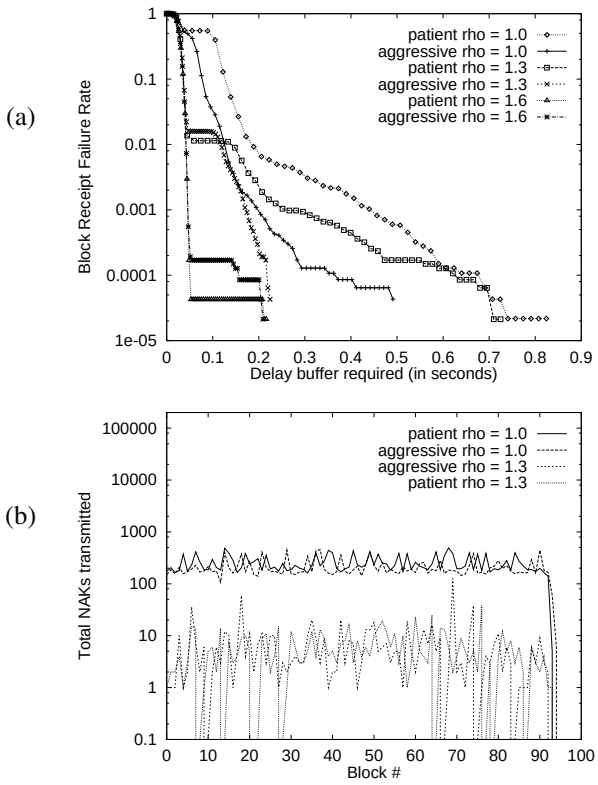


Figure 12: Examination of 500 aggressive and patient receivers with sender proactivity factors of 1.0, 1.3, and 1.6 in terms of (a) likelihood of failure to obtain an entire block vs. delay buffer, and (b) Number of NAKs transmitted per block.

ments from [6, 24].

We examined how the proactivity factor affects delay, the implosion factor, and wasted forward bandwidth and found that adding proactivity affected the performance of the asynchronous protocols in much the same way that it affected the synchronous protocols. We used the same metrics to measure implosion factor and forward bandwidth as we used for the round-based protocols in Section 4. However, the asynchronous nature of the protocols does not permit us to measure delay using the notion of rounds. Instead, the delay in reliably receiving a block is measured by taking the time at which the data packets could be decoded, and subtracting this from the latest time at which the first packet within the block could arrive, barring loss (i.e., the longest time it could take to arrive given its latency distribution in the model). We refer to this amount of time as the *delay buffer*.

We observed that for a proactivity factor of 1.4 or less, the forward bandwidth averaged around 15 packets, and increased linearly with the proactivity factor when the proactivity factor was at least 1.5. Figure 12(a) plots the fraction of blocks that cannot be recovered given the delay buffer.

We see that additional sender proactivity decreases the rate at which blocks fail to be decoded for a fixed amount of buffer. We also see that for a fixed proactivity factor, the fraction of blocks that can't be decoded given a fixed delay buffer is slightly smaller for the aggressive receiver than for the patient receiver. Figure 12(b) shows the number of NAKs received per block for proactivity factors of 1.0 and 1.3. We omit plotting results from the cases when the proactivity factor was 1.6, since for most blocks the number of NAKs sent was 0, and never rose above 2. As with the round-based protocol, a linear increase in the proactivity factor results in an exponential decay in the proactive factor.

5.4 Meeting real-time deadlines

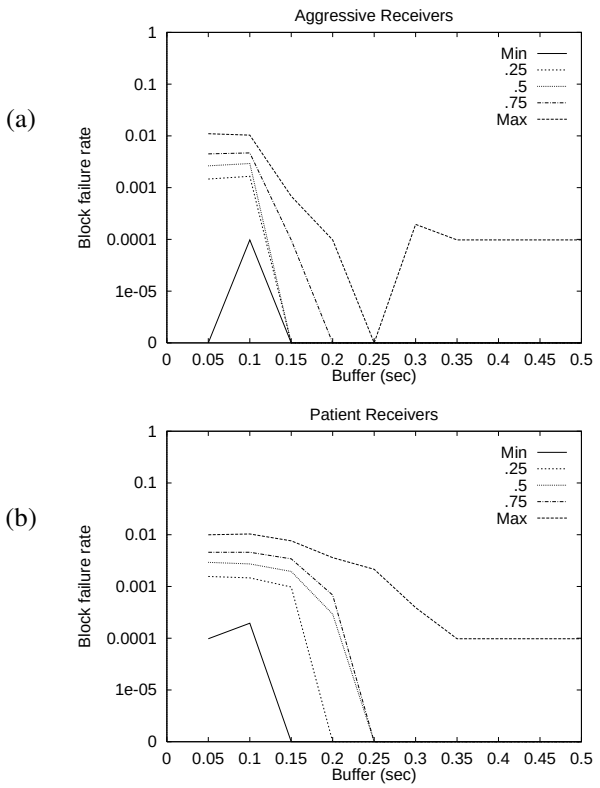


Figure 13: Block failure rates for meeting hard deadline with the block good-put guarantee set to a failure rate of .9999, where all receivers allow identical latency for retransmission. 10,000 blocks were transmitted.

Receivers can meet real-time deadlines with a high probability by determining a point in time before its deadline that gives a sufficient amount of time for it to send its NAK and receive the repairs before its deadline expires. We use a simple computation to determine what this time should be. Details on adapting the asynchronous protocols to meet real-time deadlines are discussed in [25].

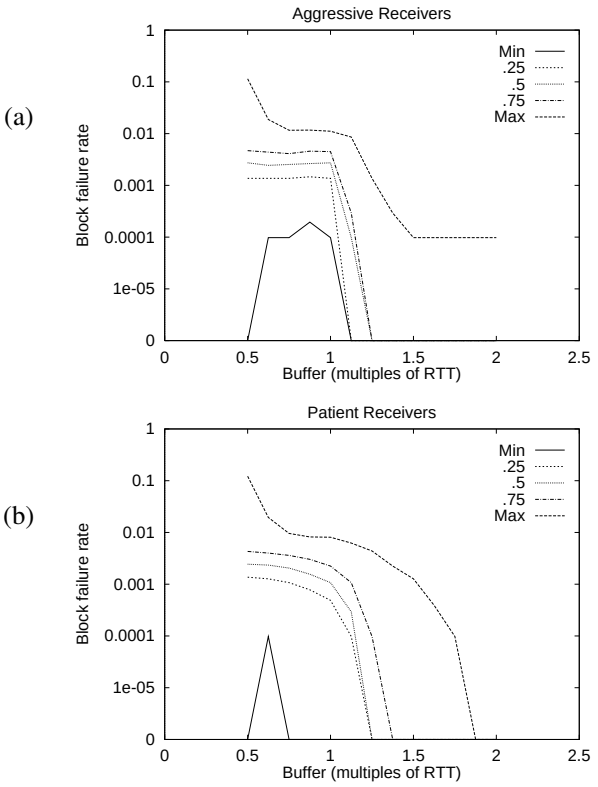


Figure 14: Block failure rates for meeting hard deadline with with the block good-put guarantee set to a failure rate of .9999, where each receivers' allowed latency is a multiple of its RTT to the sender. 10,000 blocks were transmitted.

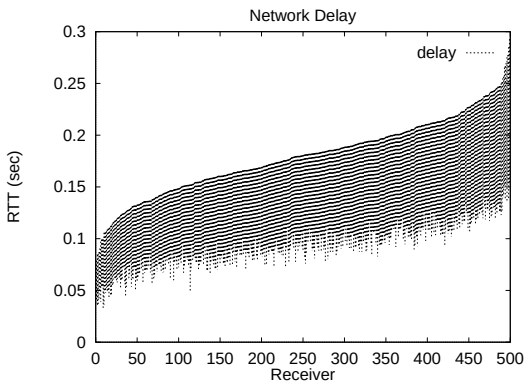


Figure 15: Variation in receiver round trip times to the source.

Results of applying the real-time algorithms are shown over the same topology of 500 receivers, with the sender using a fixed proactivity factor of 1.4. Figures 13(a) and 13(b) demonstrate the block failure rates for patient and aggressive receivers respectively in terms of the fraction of blocks that a particular receiver fails to decode, where each receiver employs an identical delay buffer. The plot labeled Min plots the fraction of lost blocks in each experiment by

the receiver that observed the lowest rate of loss. The plot labeled Max plots the loss rate as seen by the receiver with the highest rate of loss, and the others plot the loss rate observed by receivers whose rates of loss lie at the 25%, 50%, and 75% quartiles when compared with the loss rates of the remaining receivers.

Figures 14(a) and (b) are similar to Figures 13(a) and (b), except that each point compares failure rates where the ratio of each receiver's delay buffer to its round trip time to the sender is identical. In all plots, the block good-put guarantee rate is set to .9999, such that the desired probability of failing to receive a sufficient number of packets to decode a block is .9999. For each point, we only transmit 10,000 blocks due to the time such an experiment takes to run on such a large scale simulation. Figure 15 displays the round trip time delays that can occur for the various receivers to and from the sender. The delay for each receiver is uniformly distributed along its vertical strip of the darkened interval. The delay shown here accounts only for the propagation time, and does not include any processing that might occur at end-hosts.

We observe from the graphs that given identical delay buffers, aggressive receivers typically reduce failure rates beyond their patient receiver counterparts. Figure 14 shows that for block good-put guarantees to be successful, receivers require at least a round trip time in order to be able to meet the guarantee. This is clearly due to the fact that any time less than a round trip time does not give a receiver enough time to send a NAK and elicit a response. We observe from Figure 13 that receivers with larger round trip times can expect some improvement in the ability to meet their block good-put guarantees a result of receivers with smaller round trip times imposing their own block good-put guarantees.

6 Varying the loss model

So far, we have demonstrated the performance of our protocol in a heterogeneous environment. We now examine how further variations in this environment affect performance.

We begin by reexamining two assumptions that were made within our model. All previous experiments assumed that NAKs sent by receivers were never lost. We considered what happens if NAKs are dropped at a router according to a Bernoulli process identical to that used to describe data and repair packet losses. We found that for large multicast groups, such loss had no noticeable impact. We believe this is due to the fact that the protocol limits, but does not suppress, all redundant loss requests. Therefore, there is a high probability that even when NAKs can be lost, there are enough duplicate NAKs sent that at least one NAK will still reach the sender.

We also consider the impact of bursty loss at each router has through the use of a continuous two-state loss model,

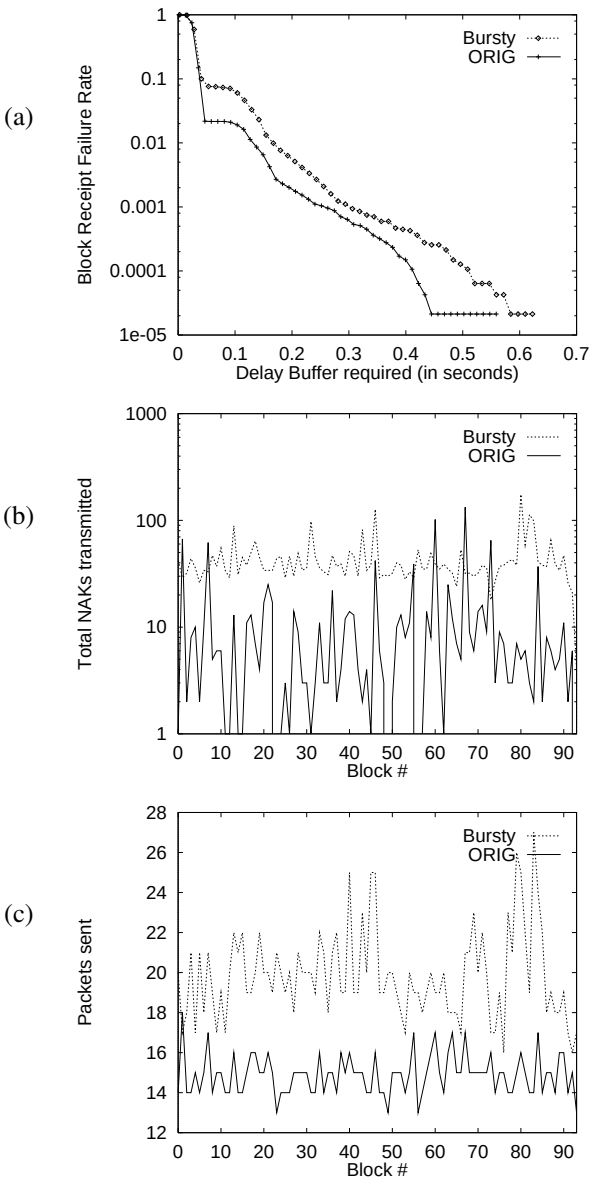


Figure 16: How results change as we add features of a more realistic network in terms of (a) good-put failure rate vs. buffer latency, (b) NAKs received per block, (c) packets sent by the sender that weren't needed by any receivers (i.e., wasted forward bandwidth), and (d) Total number of packets sent per block.

where the expected length of a burst at each router is 6 ms, while keeping the overall probability of dropping a packet fixed. Our algorithm that prevents reordering creates packet trains where the time lag between packets is 1.7 ms, thus the expected loss burst length can cover 4 packets passing through a router.

Figure 16 demonstrates the impact that bursty loss has on protocol performance with all other network features identical to those used in Section 5. Each plot here is for the case

that all receivers use the aggressive receiver algorithm, and a proactivity factor of 1.3. In each figure, plots labeled ORIG assume a Bernoulli process for data and repair loss, whereas those labeled Bursty use the bursty loss model. As in Figures 12(a) and 12(b), Figures 16(a) and 16(b) plot performance in terms of block receipt failure rate and total NAKs transmitted, respectively. Figure 16(c) plots the total number of packets that were transmitted for a block.

We observe that a network that exhibits high amounts of bursty loss can alter the performance substantially. We see that much of the time, an increasing number of repairs must be sent to satisfy all receivers. Also, latencies and the number of NAKs transmitted increases. Similar results hold for the patient receiver algorithm. One way to minimize the impact of burstiness is to increase the block size used. However, increasing the blocksize increases the amount of time between the first and last packets within a block, which increases the expected minimum latency between the first lost packet and the first repair.

7 Conclusions and Future Work

We have presented a technique that uses a hybrid of FEC and ARQ approaches that can repair data in an extremely efficient manner. We show that it is possible to make significant improvements over current commonly used techniques, and that FEC can be used to keep bandwidth requirements and NAK implosion to a minimum. We show that efficient support for large multicast groups can be performed in an end-to-end, unscoped manner, without significantly increasing bandwidth utilization over other unscoped FEC techniques, and uses considerably less bandwidth than unscoped ARQ approaches such as SRM. Also, we show that through minor bandwidth increases, receivers can improve upon performance in terms of expected time or reliability and rate of failure to meet their deadlines. We present a variety of protocols that use the technique and show their success in medium-sized multicast groups in heterogeneous networks through simulation. A prototype implementation has been developed and has been tested on the MBone. A detailed report of the results can be found in [26].

We have developed and experimented with an adaptive mechanism that is executed by the sender which varies the its proactivity factor. The mechanism works well for the network conditions that we model in this paper. However, we are exploring alternative mechanisms that are simpler and more intuitive. For this reason, we do not present an adaptive mechanism at this time.

Acknowledgments

We would like to thank Rohan Kumar who has converted our simulation code into a prototype implementation, the anonymous reviewers, Jon Crowcroft, and Joerg Nonnenmacher for their helpful comments.

References

[1] S.E. Deering, D.R. Cheriton. *Multicast Routing in Datagram Internetworks and Extended LANs*, ACM Trans. on Computer Systems , 8, 2, pp. 85-110, May 1990.

[2] T. Ballardie, J. Crowcroft, P. Francis. "Core based trees (CBT) An architecture for Scalable Interdomain Multicast Routing" Proceedings of ACM SIGCOMM'93, pp. 85-95, September 1993.

[3] Richard E. Blahut, *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, MA, 1983.

[4] Anthony J. McAuley, *Reliable Broadband Communication Using a Burst Erasure Correcting Code*. Proceedings of ACM SIGCOMM'90, pp. 297-306, Sept. 1990 Philadelphia, PA.

[5] Luigi Rizzo, *Effective Erasure Codes for Reliable Computer Communication Protocols*, Computer Communication Review, April 1997.

[6] Jorg Nonnenmacher, Ernst Biersack, and Don Towsley, *Parity-Based Loss Recovery for Reliable Multicast Transmission*, Proceedings of ACM SIGCOMM'97, pp. 289-300, Sept. 1997 Cannes, France.

[7] S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang. *A reliable multicast framework for light-weight sessions and application level framing*, IEEE/ACM Trans. on Networking, 1998.

[8] J. C. Lin and Sanjoy Paul. *Reliable Multicast Transport Protocol (RMTP)*, IEEE JSAC, 407, 3, 1414-1424,April 1997.

[9] Christos Pappadopoulos, Guru Parulkar, and George Varghese, *An Error Control Scheme for Large-Scale Multicast Applications*. Proceedings of IEEE INFOCOM'98, San Francisco, CA.

[10] Rex Xi Xu, Andrew C. Myeres, Hui Zhang, and Raj Yavatkar, *Resilient Multicast Support for Continues Media Applications*, NOSSDAV 1997.

[11] B.N. Levine, David Lavo, and J.J. Garcia-Luna-Aceves, *The Case for Concurrent Reliable Multicasting Using Shared Ack Trees*, Proceedings of ACM Multimedia 1996 Boston, MA, November 18-22, 1996.

[12] Xue Li, Sanjoy Paul and Mostafa Ammar, *Layered Video multicast with Retransmissions (LVMR): Evaluation of Hierarchical Rate Control*, NOSSDAV, 1997.

[13] Matthew T. Lucas, Bert. J Dempsey, and Alfred C. Weaver, *MESH: Distributed Error Recovery for Multimedia Streams in Wide-Area Multicast*, Proceedings Sixth International Conference on Computer Communications and Networks (IC3N), 1997

[14] Luigi Rizzo and Lorenzo Vicisano, *RMDP: An FEC-based Reliable Multicast Protocol for Wireless Environments*, Mobile Computing and Communications Review, Volume 2, Number 2, April 1998.

[15] D. DeLucia, K. Obraczka, *Multicast Feedback Suppression Using Representatives*, Proceedings of IEEE INFOCOM'97, Kobe, Japan, March 1997.

[16] S. Kasera, J. Kurose, D. Towsley, *A Comparison of Server-Based and Receiver-Based Local Recovery Approaches for Scalable Reliable Multicast*, Proceedings of IEEE INFOCOM'98, San Francisco, CA, March 1998.

[17] J. Nonnenmacher, M. Lacher, M. Jung, E. W. Biersack and Georg Carle, *How bad is reliable multicast without local recovery?*, Proceedings of IEEE INFOCOM'98, San Francisco, CA, March 1998.

[18] T. Speakman, D. Farinacci, S. Lin, and A. Tweedly, *Pretty Good Multicast (PGM) Transport Protocol Specification*, Internet Draft draft-speakman-pgm-spec-00.txt, January 1998.

[19] M. Yajnik, J. Kurose, D. Towsley, *Packet Loss Correlation in the MBone Multicast Network*, IEEE Global Internet Conference. London, UK, November 1996.

[20] M. Handley, *An Examination of MBone Performance*, Technical Report, UCL and ISI, January 1998.

[21] R. Kermode, *Scoped Hybrid Automatic Repeat Request with Forward Error Correction (SHARQFEC)*, to appear in ACM SIGCOMM'98, Vancouver, CA, September 1998.

[22] K. Perumalla, A. Ogielski and R. Fujimoto, *MetaTeD: A Meta Language for Modeling Telecommunication Networks*, GIT-CC-96-32, Technical Report, College of Computing, Georgia Institute of Technology, 1997.

[23] H.R. Deng and M.L. Lin, *A Type I Hybrid ARQ system with Adaptive Code Rates*, IEEE Transactions on Communications, COM-43 (2/3/4):733-737, February/March/April 1995.

[24] S. Kasera, J. Kurose, and D. Towsley, *Scalable Reliable Multicast Using Multiple Multicast Groups*, CMPSCI Tech Report 96-73, University of Massachusetts, October, 1996 (A shorter version of this paper appeared in ACM SIGMETRICS '97).

[25] D. Rubenstein, J. Kurose, and D. Towsley, *Real-Time Reliable Multicast Using Proactive Forward Error Correction*, CMPSCI Tech Report 98-19, University of Massachusetts at Amherst, March 1998.

[26] R. Kumar, *Design and Implementation of a Proactive Real-Time Reliable Multicast Protocol using Forward Error Correction (FEC)*, Masters Thesis, University of Massachusetts at Amherst, May 1998, in preparation.

[27] Jean Bolot, Private Communication, 1998.

Appendix

The following lemma applies for generating block good-put rates within hard deadlines. If C is the event that a block is decodable, then ensuring that $P(C) \geq \mathcal{P}$ gives a block good-put rate of at least $\mathcal{P}$. In section 4.1, we propose that a receiver wait until the last possible moment to send a final NAK for that block. At this point, the receiver knows the number of packets, l, that it still needs to recover the block. Regardless of the number of packets received from this final request, a receiver does not request any further NAKs.

Lemma 1 *If the receiver chooses m such that $\mathcal{D}(m,l) \geq (\mathcal{P} - \mathcal{D}(n,k))/(1-\mathcal{D}(n,k))$, then $P(C) \geq \mathcal{P}$.*

Proof: Let N be an R.V. that equals the number of packets from the block sent by the sender prior to the final NAK transmission, and L be an R.V. that equals the number of packets required by the receiver at this time. Then:

$$P(C \wedge (N=n)) = \sum_{l=0}^{k} P(C \wedge (L=l) \wedge (N=n))$$
$$= P(C \wedge (L=0) \mid (N=n))P(N=n) +$$
$$\sum_{l=1}^{k} P(C \mid (N=n) \wedge (L=l))P((N=n) \wedge (L=l)).$$

If $L = 0$, then C is guaranteed to hold, since the receiver needs no further packets to complete the block. Thus, $P(C \wedge L = 0 \mid N=n) = P(L=0 \mid N=n) = \mathcal{D}(n,k)$. It is also the case that if the final NAK requests m packets, then $P(C \mid N = n \wedge L = l) = \mathcal{D}(m,l)$. Selecting m as stated in the lemma therefore gives us that $P(C \mid N = n \wedge L = l) = \mathcal{D}(m,l) \geq (\mathcal{P} - \mathcal{D}(n,k))/(1-\mathcal{D}(n,k))$. Thus:

$$P(C \wedge N=n) \geq \mathcal{D}(n,k)\ P(N=n) +$$
$$\sum_{l=1}^{k} \frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} P(L=l \mid N=n)\ P(N=n)$$

$$= \mathcal{D}(n,k)\ P(N=n) + \frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} \times$$
$$P(N=n) \sum_{l=1}^{k} P(L=l \mid N=n)$$

$$= \mathcal{D}(n,k)\ P(N=n) + \frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} \times$$
$$P(N=n)(1 - P(L=0 \mid N=n))$$

$$= \mathcal{D}(n,k)\ P(N=n) +$$
$$\frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} P(N=n)(1-\mathcal{D}(n,k)) \quad (1)$$

$$= \mathcal{P}P(N=n).$$

It follows that $P(C) = \sum_{i=0}^{\infty} P(C \wedge N=n) \geq \sum_{i=0}^{\infty} \mathcal{P}P(N=n) = \mathcal{P}$. ∎

Cost-optimal Data Retrieval for Video Servers with Variable Bit Rate Video Streams

Ernst W. Biersack
Institut Eurecom
06904 Sophia Antipolis, FRANCE
Email: erbi@eurecom.fr

Maher Hamdi
Institute for computer Communications and Applications (ICA)
Swiss Federal Institute of Technology (EPFL), Switzerland
Email: Maher.Hamdi@epfl.ch

Abstract

We consider the problem of data retrieval from disk storage in a video server where the data is stored in constant size blocks. The two critical resources of a video server are the amount of disk I/O bandwidth and the main memory buffer required to support a video stream. We propose a new algorithm that allows us to determine the cost-optimal trade-off between these two server resources as well as bounds and statistics of the start-up latency seen by the user.

Keywords – Video Server, disk storage, continuous media, VBR video, retrieval.

1 Introduction

Video servers store digitized, compressed, continuous media information on secondary or tertiary storage. The secondary storage devices allow random access and provide short seek times compared to tertiary storage. Video server design differs significantly from that of traditional data storage servers due to the large size of the objects stored and the real-time requirements for their retrieval. A video server must meet the requirements that stem from the continuous nature of audio and video and must guarantee the delivery of continuous media data in a timely fashion. The critical resources in a digital video system are the disk bandwidth, storage volume, and main the memory on the server side (see figure 1). Dimensioning and utilizing these resources efficiently is a critical task because they largely determine the cost of the video server and therefore the cost of the service.

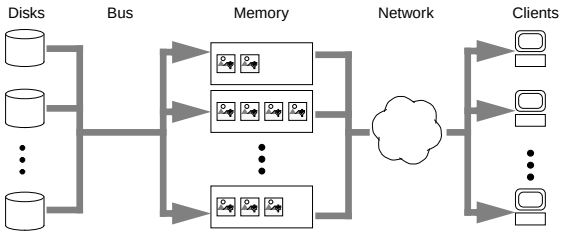


Figure 1. Components of a Video Server System.

The scheduling of the requests to read from disk determines the order in which the requests are served and influences the disk I/O efficiency and the buffer requirement. We assume that the reads from disk are organized in rounds and starvation is avoided by reading ahead an amount of data that lasts in terms of playback duration at least until the end of the next round. The requests that need to be served during one round can be scheduled using SCAN or C-SCAN scheduling [14] to minimize the seek overhead between adjacent retrievals of a single round.

The data retrieval technique determines the way data is read from the disk during a service round. Using VBR as a data model for a video, we can map video data onto data blocks stored on the disk in two ways:

- Variable size blocks of constant playout dura-

tion, referred to as **constant time length** (CTL) or

- Fixed size blocks of variable playout duration, referred to as **constant data length** (CDL) [2].

  Throughout this paper we assume CDL retrieval. For a comparison between CDL and CTL see [5].

Constant data length (CDL) retrieval performs non-periodic retrieval of constant amounts of data from the disk. To make CDL compatible with round-based disk retrieval, we introduce the notion of active and idle rounds. During an **active** round, a constant size data block is read from the disk. During an **idle** round, no data at all is retrieved. Whether a round will be active or not, can be decided on-line. The details will be explained below.

Given a fixed amount of these resources, a video server can only deliver a limited number of video streams simultaneously. The quality of service in a video server is determined by the timely delivery of the video information, which is encoded as a *variable bit rate stream* (VBR) of constant quality. If the retrieval rate of data falls behind the transmission rate, the client will experience starvation and the quality of service will be affected negatively. If the arrival of data is ahead of the transmission, the difference between the amount of data that has been retrieved and the amount of data transmitted, which is referred to as backlog, must be buffered until transmission. In this paper we assume that:

- Buffering is done at the server side. Although the buffering can be done on the client side, we prefer the buffer to be at the server side, which allows for sharing the buffer among the different streams.(see figure 1).
- the service is *deterministic*, i.e. there will never be starvation.

2 Video Traffic Characterization

To offer a deterministic service for a VBR video we need a deterministic, i.e. a worst case traffic characterization of the video stream. The novelty of our approach consists in using a so called "burstiness function" to characterize a video. The burstiness function was used by several authors (e.g. [4], [9], [8], [7], [11]) as a deterministic characterization of VBR video streams. It can be seen as a set of leaky buckets to which the traffic conforms in a lossless way with leak rates ranging between the average and peak rate of the traffic of a video stream. More formally, given a VBR video traffic defined by its **instantaneous generation bit rate** $c(t)$ (which defines its consumption rate as well), its burstiness function $b(r)$ is defined as:

$$b(r) = \max_{t_a \le t_b} \int_{t_a}^{t_b} (c(t) - r)dt$$

where $\lambda \le r \le P$ and λ and P denote the **average** and **peak rates** of the video stream. In the context of video servers, we use the burstiness function $b(r)$ to establish a relationship between

- the amount of disk I/O bandwidth, which can vary between the peak and mean bit rates of the video, and
- the amount of buffer required on the server side.

3 System Model

Next we introduce the notations and definitions used to describe the system model:

- N: number of videos
- $[0, T]$: duration of video
- $t_s \in [0, T]$: Point in the video where the playout starts
- $c_i(t)$: transmission (and consumption) rate of video $i \in N$ at time $t \in [t_s, T]$
- r: maximum retrieval rate of video from disk
- $R_i(t)$: current data retrieval rate of video $i \in N$ at time $t \in [t_s, T]$
- $b_i(t_s, r) = \max_{t_a \le t_b; t_a, t_b \in [t_s, T]} \int_{t_a}^{t_b} (c_i(t) - r)dt$: **burstiness function** of video i
- $p_i(t_s, r) = \max_{t_o \in [t_s, T]} \int_{t_s}^{t_o} (c_i(t) - r)dt$: **prefetch function** of video i
- $d_i(t_s, t_x, r) = \int_{t_s}^{t_x} (c_i(t) - r)dt$: difference between cumulative consumption and production
- $Ts_i(t_s, r) = p_i(t_s, r)/r$: **startup latency**
- $B_i(t_s, t_x, r)$: **buffer level** of video i at time t_x. Any time the buffer level falls below zero is referred to as **starvation**.

3.1 Buffer Preloading

Variable bit rate video is highly bursty with ratios P/λ of peak to mean bit rate often larger than ten. To offer a deterministic service, the cumulative retrieval (production) of the video data from disk must always be *larger* than the cumulative consumption, i.e. $B_i(t_s, t_x, r) > 0, \forall t_x \in [t_s, T]$.

Theorem PL: If we retrieve data at a constant rate r lower than the peak bit rate P , we can avoid starvation by *preloading* the amount $p_i(t_s, r)$ of data into the buffer before the transmission of video starts.

Proof: $p_i(t_s, r) - d_i(t_s, t_x, r) = \max_{t_o \in [t_s, T]} \int_{t_s}^{t_o} (c_i(t) - r)dt - \int_{t_s}^{t_x} (c_i(t) - r)dt \geq 0, \forall t_x \in [t_s, T]$. QED.

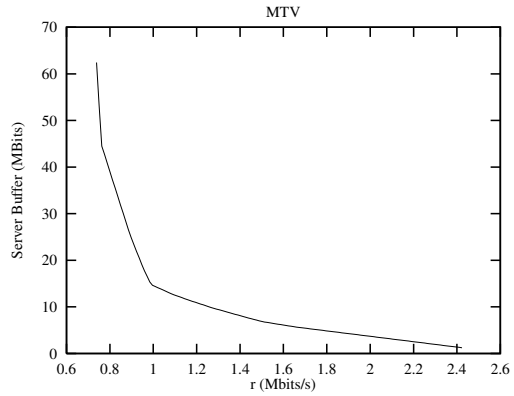
We can easily show that it is sufficient to compute the prefetch function as:
$p_i(t_s, r) = \max_{t_o \in [t_s, t_b]} \int_{t_s}^{t_o} (c_i(t) - r)dt$, where t_b is the right hand side of the interval $[t_a, t_b]$ for which $\int_{t_a}^{t_b} (c_i(t) - r)dt = b_i(t_s, r)$ since due to the definition of $b_i(t_s, r)$ we have:
$\forall t_x \in (t_b, T] : \int_{t_b}^{t_x} (c_i(t) - r)dt \leq 0$.

3.2 Burstiness Function

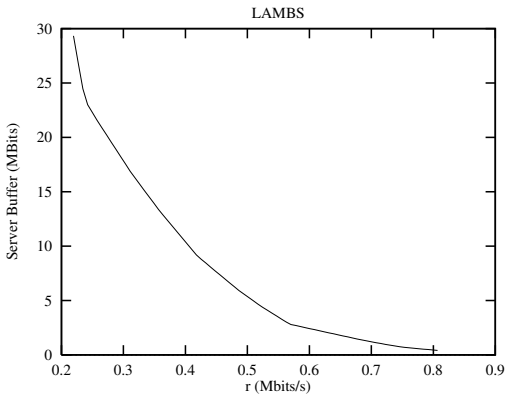
The burstiness function indicates by how much, in the worst case, for any interval $[t_a, t_b] \subset [t_s, T]$ of time, the consumption will be higher than the production, i.e. the amount of video data retrieved from the disk. As we will show later, $b_i(t_s, r)$ is the maximum amount of video data we need to buffer to avoid starvation at any point of time.

If we plot $b_i(t_s, r)$ as a function of the rate r at which video i is retrieved from the disk, we see (figure 2) that the burstiness function is monotonously decreasing and convex. $b_i(t_s, r)$ is defined for the range of rates r, $\lambda \leq r \leq P$ between the videos mean bit rate λ, for which the value of the burstiness function is highest, and the peak bit rate P, for which the value of the burstiness function is zero (figure 2). For different videos the burstiness function takes different absolute values and its shape (steepness) varies. We see, for instance for the MTV trace (figure 2(a)) , that a small increase in the rate r from 0.8 Mbit/s to 1.0 Mbit/s leads to a drastic reduction of the amount of server buffer required. For Lambs, the impact of an increase of r on the buffer requirement is less pronounced.

We use in this paper the MPEG-1 video traces produced by O. Rose [12] that are publicly available. The different names such as MTV, Lambs, or Starwars refer to the traces of films with the same names.



(a) MTV



(b) Lambs

Figure 2. Burstiness Function.

3.3 Retrieval Algorithm

If the data retrieval rate $R_i(t)$ of video i remains constant (equal to r) during the whole session, the server buffer keeps growing at an average rate of $(r - \lambda)$ and its maximum size becomes proportional to the session duration. It is therefore necessary to interrupt the retrieval process to keep the buffer size bounded while still ensuring that no starvation occurs. The retrieval proceeds as follows:

- When a client requests a video i to start at time t_s, the associated buffer (at the server side) will first be filled with the amount $p_i(t_s, r)$ of data. Then the transmission starts at the variable rate $c_i(t)$. Only a small buffer is used at the *client side* to absorb transmission jitter.

- During video transmission, the buffer will be filled at a rate $R_i(t) = r$ while the buffer level is below $b_i(t_s, r)$. If a buffer level of $b_i(t_s, r)$ is reached, the filling of the buffer will be interrupted, i.e. $R_i(t) = 0$, until the buffer level falls below $b_i(t_s, r)$[1].

Theorem SV: The retrieval algorithm will avoid starvation at all times, i.e. .$B_i(t_s, t_x, r) \geq 0, \forall t_x \in [t_s, T]$.

Proof: We distinguish the two possible scenarios:

- if the retrieval rate has never been interrupted between t_s and t_x (equivalently the buffer size has never reached $b_i(t_s, r)$), then we have $B_i(t_s, t_x, r) = p_i(t_s, r) - d_i(t_s, t_x, r) > 0$ (given by theorem PL).
- otherwise, let t_y be the last time in $[t_s, t_x]$ where $B_i(t_s, t_y, r) = b_i(t_s, r)$. Then we have $B_i(t_s, t_x, r) = b_i(t_s, r) - \int_{t_y}^{t_x} (c_i(t) - r)dt$ which is necessarily non-negative (by definition of $b_i(t_s, r)$. QED.

Figure 3 summarizes the retrieval and delivery of the video data and how the sender controls the retrieval as function of the buffer level.

4 Performance

In this section we present a performance evaluation, in terms of cost and delay, of the proposed scheme.

4.1 Cost Study

Theorem SV allows us to use the burstiness function to optimally trade-off buffer memory for disk retrieval rate. Assume that the price P_{ram} per **Mbit of main memory** and the price $P_{dI/O}$ per **Mbit/s of disk I/O** are known, then, if we ignore for the moment all other cost factors, the **cost** $P_{tot}(i, r)$ **of servicing a single video** i is given as

$$P_{tot}(i, r) = P_{ram} * b_i(t_s, r) + P_{dI/O} * r \qquad (1)$$

Figure 4 depicts the cost function for a given ratio $\frac{P_{dI/O}}{P_{ram}} = 20$, which means that the price for one Mbit/s of disk I/O is 20 times higher than the price for one Mbit of RAM buffer. We see that for both videos there is a **optimal retrieval rate** r^* for which

[1] In practice this is implemented using a threshold, whose value takes into account the discrete way of transferring data blocks

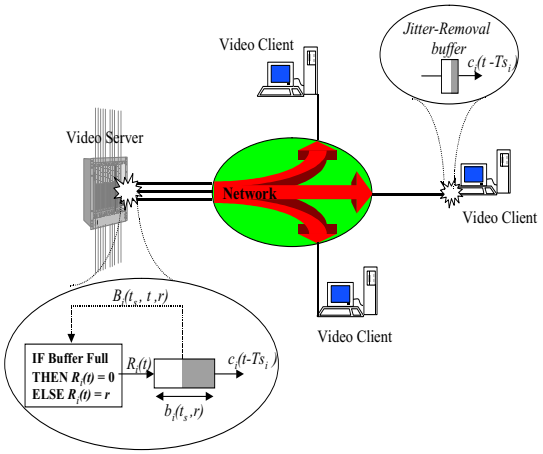


Figure 3. Systems Model.

the total cost is minimal. The chosen ratio of 20 is a realistic value given todays prices for disks and RAM buffer.

If we take the first derivative of $P_{tot}(i, r)$ and set it to zero, we obtain the rate r_i^* for which the delivery of the video i is cost-optimal. We get

$$\frac{dP_{tot}(i, r)}{dr} = \frac{db_i(t_s, r)}{dr} * P_{ram} + P_{dI/O} \qquad (2)$$

$$\frac{db_i(t_s, r_i^*)}{dr} = -\frac{P_{dI/O}}{P_{ram}} \qquad (3)$$

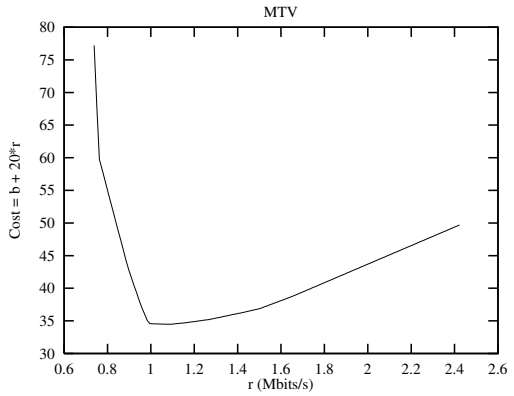
Figure 5 depicts the optimal rate r as a function of the cost ratio $\frac{P_{dI/O}}{P_{ram}}$. As the cost ratio increases, i.e. the cost of the disk IO compared to the cost of memory increases, it becomes more cost efficient to decrease the rate r at the expense of an increasing buffer size.

4.2 Start-up latency

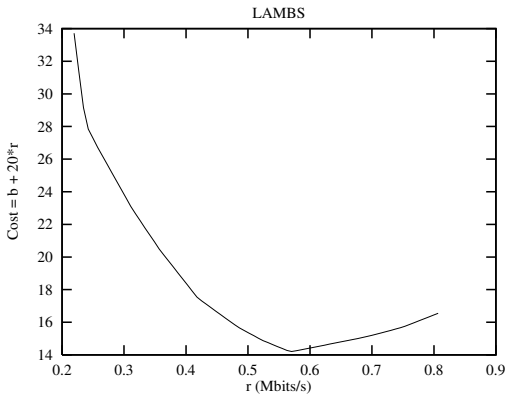
The start-up latency can be defined in different ways. The **worst case** start-up latency $Ts_i(r)$ is defined as

$$Ts_i(r) = \max_{t_s \in [0, T]} Ts_i(t_s, r) \qquad (4)$$

The worst case start-up latency, as function of the rate r depicted in figure 6, has a shape similar to

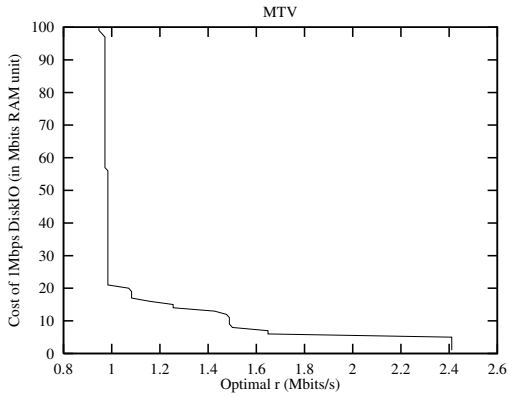


(a) MTV

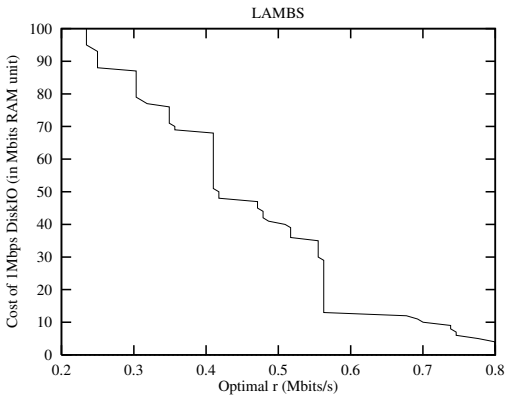


(b) Lambs

Figure 4. Cost function P_{tot} for $\frac{P_{dI/O}}{P_{ram}} = 20$.



(a) MTV



(b) Lambs

Figure 5. Optimal I/O bandwidth vs. cost ratio $\frac{P_{dI/O}}{P_{ram}}$.

the burstiness function (cf. figure 2). Indeed one can easily show that:

$$Ts_i(r) = b_i(t_s, r)/r$$

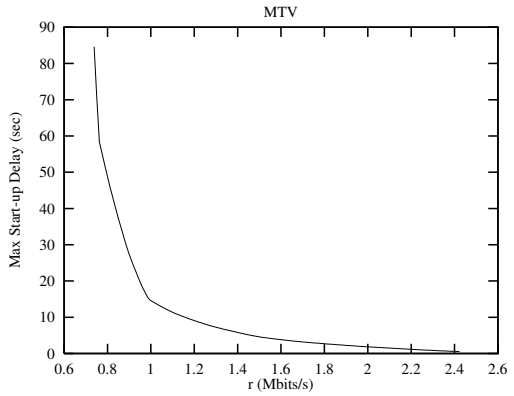
For values of r that are close to the mean rate, the worst case start-up latency is to the order of a minute or higher. Such high values are clearly not acceptable for a user and do not allow any meaningful interactivity during the playout.

However, the worst case start-up latency gives overly pessimistic values as we can see from figure 7 that depicts the **probability distribution function** of the start-up latency for a given rate r. There we see that the probability that the start-up latency will take its maximal value is very low.

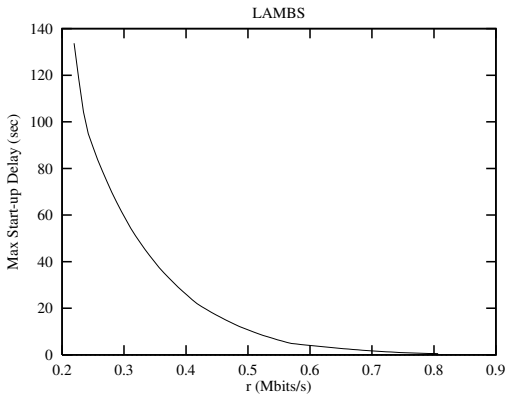
Clearly, the start-up latency depends on the starting time t_s of the playback. We could think about the average value of $Ts_i(t_s, r)$ (over all t_s values) as a good measure of the start-up latency. However this assumes that users start watching movies at a time that is uniformly distributed over the movie duration. A more reasonable assumption would be that the probability of starting at time t_s decreases exponentially in time which reflects that most of the users start watching the sequence from the beginning. We therefore suggest another measure for the start-up latency that we call **weighted mean start-up latency** and this is defined as

$$\hat{Ts}_i(r) = \int_0^T k.e^{-kt} Ts_i(t, r)\, dt \qquad (5)$$

where k is a tunable parameter that is fixed (for our

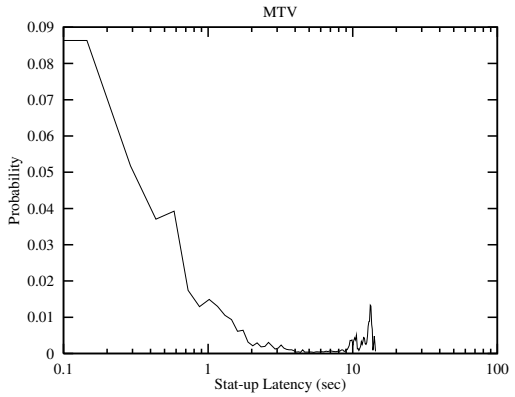


(a) MTV

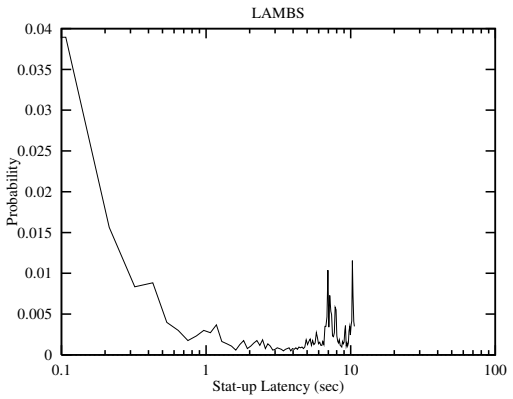


(b) Lambs

Figure 6. Worst case start-up latency.



(a) MTV



(b) Lambs

Figure 7. Distribution of the start-up latency.

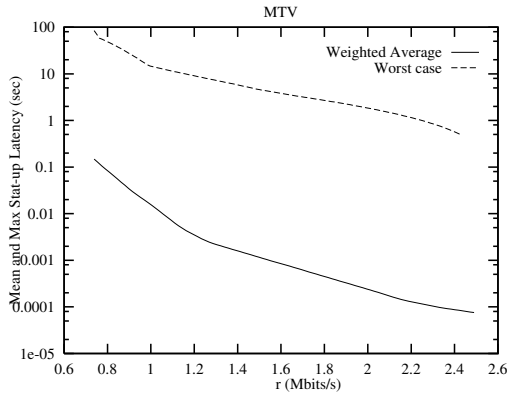
experiments) so that 90% of sessions start before $t_s = 60$ sec.

The weighted mean start-up latency, together with the worst case start-up latency, as function of the rate r, is depicted in figure 8. We see that the weighted mean start-up latency is about two orders of magnitude lower than the worst case start-up latency.

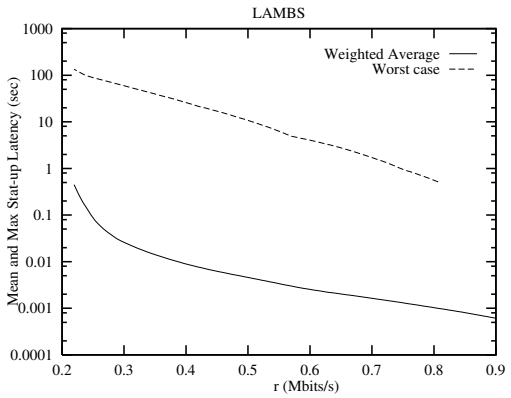
In figure 9 we see the start-up latency as a function of the starting time t_s. This figure confirms that in most of the cases the start-up latency is much lower than the worst-case start-up latency. To reduce the worst-case start-up latency, one can think of a modification of the scheduling algorithm: When a client starts a movie at a point of time where the start-up latency perceived would be higher than a few seconds, data can be sent with a rate higher than r until the buffer is full. This "fast load" [10] is possible since the video server has, for each client, idle rounds where no data are retrieved. The percentage of idle rounds for stream i with mean bit rate λ_i that is allocated a rate r_i, with $r_i > \lambda_i$, is given as $(r_i - \lambda_i)/r_i$. Note that these idle rounds occur even when the maximum number of clients that can be served is admitted.

5 Related Work

Previous work has been done in the area of data retrieval in video servers for VBR streams. Chang and Zhakor were one of the first to discuss CTL and CDL-based retrieval for VBR streams [2, 3]. For CDL, there are no idle rounds and the bit rate allocated is equal to the mean bit rate of the video, which

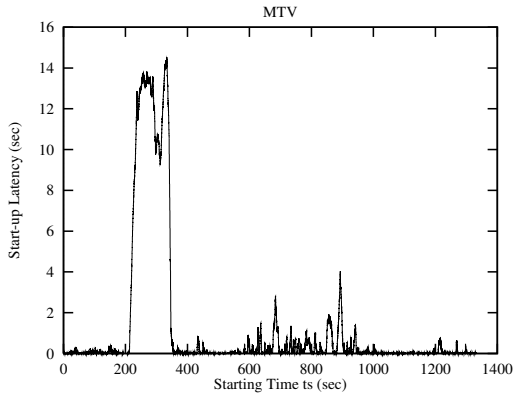


(a) MTV

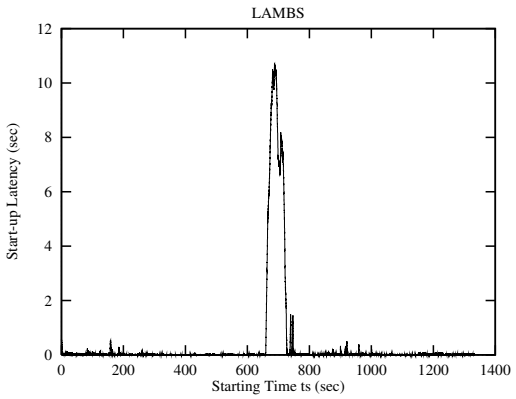


(b) Lambs

Figure 8. Worst-case and weighted mean start-up latency.



(a) MTV



(b) Lambs

Figure 9. Start-up latency as function of time t_s where playout starts.

corresponds to one point on the burstiness curve. As we know, choosing the mean bit rate results in a large start-up latency and a large buffer requirement.

In [5, 1] the authors introduced the idea of idle rounds for CDL. They also distinguish the disk service round, during which data for each stream are read exactly once from the disk, and the smoothing interval, which is used to compute the amount of I/O bandwidth that must be reserved for a stream: the smoothing interval is a *multiple* of a disk service round. The separation of disk service round and smoothing interval reduced the amount of I/O bandwidth required, which in turn improved the overall efficiency of the video server. However, smoothing was still limited to an interval of several seconds of video.

Recently, Sahu [13] looked at *deadline-based* scheduling as opposed to round-based scheduling, which was assumed by the papers cited above. Sahu showed that deadline-based scheduling allows smoothing over much larger intervals than round-based scheduling, which reduces the rate variability and improves overall disk utilization. However, deadline-based scheduling does not allow for seek optimizations as does SCAN that is used in combination with round-based scheduling.

Our work also increases the smoothing interval while still allowing the use of round-based scheduling. To our knowledge, we are also the first to use explicit relationship between the amount of I/O bandwidth and the amount of main memory. Since the burstiness function is decreasing and convex, there

exits an optimal bandwidth for each VBR video, in the sense that the overall cost for the resources required to serve the video will be minimal.

There is a large body of work on smoothing techniques for the *transmission* of prerecorded VBR video [6] that aims at reducing the variability of the video stream that is transmitted over the network: A prefetch buffer at the *client* is used to smooth the video stream transmitted over the network. This work is complementary to ours since it does not at all consider the aspect of video retrieval from a video server but focuses on the video transmission aspect. It would be interesting to combine both aspects to conceive an end-to-end solution for efficient video retrieval and transport.

6 Conclusion

An appropriate characterization of pre-coded VBR video is required to efficiently dimension a video server's resources. The two main resources usually considered are the disk I/O bandwidth and the memory used to buffer data before transmission. The benefits of using the burstiness function as a deterministic video traffic characterization is that it provides an explicit relationship between the retrieval bandwidth and the required memory. In this way, it can be considered the most suitable traffic model for a deterministic service. Although the proposed mechanism is described at the fluid scale its implementation is straight forward, in particular in the constant data length version.

References

[1] E. W. Biersack and F. Thiesse. A new class of constant data length retrieval algorithms for video servers with vbr streams. In *Multimedia Storage and Archiving Systems, SPIE Photonics East '96 Symposium*, number 2916, pages 316–329, Boston, MA, Nov. 1996.

[2] E. Chang and A. Zakhor. Admission control and data placement for vbr video servers. In *Proceedings of the 1st International Conference on Image Processing*, pages 278–282, Austin, Texas, November 1994.

[3] E. Chang and A. Zakhor. Cost Analyses for VBR Video Servers. In *IS and T/SPIE International Symposium on Electronic Imaging: Science and Technology, Volume 2667: Multimedia Computing and Networking*, volume 2667, pages 381–397, San Jose, Jan. 1996.

[4] R. Cruz. A calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transactions On Information Theory*, 37(1):114–131, Jan. 1991.

[5] J. Dengler, C. Bernhardt, and E. W. Biersack. Deterministic admission control strategies in video servers with variable bit rate streams. In B. Butscher, E. Moeller, and H. Pusch, editors, *Interactive Distributed Multimedia Systems and Services, European Workshop IDMS'96, Berlin, Germany*, volume 1045 of *LNCS*, pages 245–264. Springer Verlag, Heidelberg, Germany, Mar. 1996.

[6] W. Feng and J. Rexford. A Comparision of Bandwidth Smoothing Techniques for the Transmission of Prerecorded Video. In *Proc. IEEE INFOCOM'97*, Kobe, Japan, Apr. 1997.

[7] M. Hamdi, J. Roberts, and P. Rolin. Rate Control for VBR Video Coders in Broad-band Networks. *IEEE Journal on Selected Areas on Communications*, 15(6):1040–1051, Aug. 1997.

[8] S. Low and P. Varaiya. Burstiness bounds for Some Burst Reducing Servers. In *INFOCOM'93*, pages 2–7, Mar. 1993.

[9] D. Lucantoni, M. Neuts, and A. Reibman. Methods for Performance Evaluation of VBR Video Traffic Models. *IEEE/ACM Transactions On Networking*, 2(2):176–180, Apr. 1994.

[10] G. Petit, D. Deloddere, and W. Verbiest. Bandwidth resource optimization in video-on-demand network architectures. In *Proceedings of the 1994 1st International Workshop on Community Networking*, pages 91–97, San Francisco, CA, July 1994.

[11] J. Roberts and M. Hamdi. Video Transport in ATM Networks. *Interoperable Communication Networks Journal, Baltzer Science*, 1:121–143, 1998.

[12] O. Rose. Statistical Properties of MPEG Video Traffic and Their Impact on Traffic Modelling in ATM Systems. Technical Report 101, Institute of Computer Science, University of Wuerzburg, FTP address of video traces: ftp://ftp-info3.informatik.uni-wuerzburg.de/pub/MPEG/, University of Wuerzburg, February 1995.

[13] S. Sahu, Z. Zhang, J. Kurose, and T. D. On the efficient retrieval of vbr video in a multimedia server. In *Proc. 1997 Int'l Conf. on Multimedia Computing and Systems (ICMCS97)*, Ottawa, Canada, June 1997.

[14] R. Steinmetz and K. Nahrstedt. *Multimedia: Computing, Communications and Applications*. Innovative Technology Series. Prentice Hall, Englewood Cliffs, NJ, 1995.

Adaptive Quality-of-Service Session Management for Multimedia Servers

John Reumann and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
{kgshin, reumann}@eecs.umich.edu

Abstract

Contemporary multimedia applications usually require servers to provide static QoS (Quality-of-Service), such as constant bit rate or fixed average transfer rate. These requirements do not reflect the application needs, but result from lack of support for adaptive QoS in applications as well as in operating systems. Multimedia servers should provide "acceptable" service under a variety of QoS configurations. When they are ported to systems that do not support explicit QoS guarantees (e.g., the Internet), the exploitation of multimedia's inherent adaptability is an absolute must.

We propose a model for server design that addresses the need for adaptive QoS. Using economic theory, the server is designed to allocate resources so as to maximize the weighted sum of clients-perceived QoS. Our model views the QoS defined for a session as a variable that can be changed on-the-fly to improve user-perceived QoS on an aggregated basis. This on-the-fly change capability allows the server to degrade or upgrade the QoS of individual sessions within a pre-specified range, depending on the loading condition and/or resource availability. Simulation of our model showed that it (i) increases the number of clients served by the system between 30 and 130%; (ii) improves clients' perception of the service; and (iii) effectively controls the number of adaptation operations needed for these improvements.

1 Introduction

A main disadvantage of today's multimedia server architectures is their inability to support load-sensitive yet "acceptable" Quality-of-Service (QoS). Conventional multimedia server architectures are based on making "binary" admission decisions: either admit a client while guaranteeing a specific level of QoS or reject him. This design may work well for small intra-nets with a few clients, e.g., in-house training videos. However, if one tries to serve a greater audience, one would need a solution that offers more flexibility in handling active sessions, i.e., degrades (upgrades) the QoS for each active client within a pre-specified range as the load on the server increases (decreases). Such flexibility will greatly benefit the design of on-demand services because it enables the server to provide degraded (upgraded) yet satisfactory service to many clients under heavy (light) load. As a result, clients may try to receive better service by avoiding peak load times. Finding an appropriate adaptation model that helps achieve this is very important. We propose such a model, combining adaptability with commercial applicability.

Multimedia servers should operate under various resource constraints, as they must usually adapt their clients' QoS to changes in resource availability. A client should be able to specify his preference of a particular QoS configuration over others. Under certain constraints these preferences can be translated into utility functions [4, 11]. Since utility functions provide a compact representation of preferences, we can use them in making resource-allocation decisions inside a multimedia server in a load-sensitive manner.

Evolution of integrated multimedia services will expand the domain of commercial multimedia applications. Server design will be influenced by both economic incentives (primary) and technical challenges (secondary). The primary objective is to implement the service provider's economical policies while providing service to the clients. Therfore, each multimedia server has to fit into its economical environment, rather than forcing the environment to adapt to yet another type of multimedia technology. More specifically, we want ser-

vice policies defined in economic terms so as to translate them directly into concrete policies of client session management within the multimedia server, thus interfacing economic theory to the application domain by the means of price and utility. This will require some economic modeling.

Modeling every client of a multimedia server as its own independent economic entity would require too much of modeling overhead, especially if there are frequent client arrivals. We therefore present a model, which reduces economic theory to an acceptable tradeoff between the efficiency of representation and "natural feel" of the price–utility interface. The tradeoff chosen in our model is to consider groups of similar clients, instead of considering each client individually. Membership in such a group should be associated with a price/fee, so that clients have to evaluate whether the description of a group fits their interests and affordability; they may or may not subscribe to it, depending on their evaluation results. In this way, the decision problem of assigning a client to an "interest group" and determining his utility function is off-loaded to the client himself.

The problem of defining groups and the membership fee is entirely an economical decision, subject to technological limitations and market behavior. The impact of this decision extends far beyond the scope of multimedia systems. Hence we will assume that the service provider has come up with group definitions, including the groups' utility functions for different levels of QoS, probably derived from opinion polls or estimation. The provider will make these descriptions visible to its clients, so clients are aware of the price, QoS, and utility of the group they may subscribe to. The supported levels of QoS are coded into the actual implementation of the multimedia service and may depend on the particulars of the server's operating system (OS).

In real economic systems, resources are allocated adaptively to changes in the environment, such as the arrival or departure of clients, which, in turn, alter the amount of each available resource. If more resources become available, some clients will be given better QoS, thus increasing average user-perceived QoS. Conversely, if resource supply is reduced due to new clients and resource failures, some clients' QoS may have to be degraded. This calls for the need for flexible QoS: a client may receive at one of several "acceptable" levels of QoS.[1] Flexible QoS enables the server to adapt to fluctuations in resource supply, load, and different client preferences, but we do not know how to change the resource allocation inside the server for each client session as the environment changes.

We propose a new model called the **Fl**exible **Mu**ltimedia **S**ession **S**cheduling (**FluSS**), which facilitates session management for a multimedia server with multiple service classes. The model includes definitions of server-side, client-side, and an algorithm designed to re-allocate resources as new clients arrive and old ones leave. The objective of this algorithm is to increase aggregated utility, i.e., the sum of all clients' perceived utilities.

The **FluSS** model leaves price determinaton up to the service provider. It implements the simplest economic configuration interface by allowing the service provider to group clients, estimate their assumed utilities, specify the fee they are supposed to pay, and configure the service levels supported by the server.

The rest of the paper is organized as follows. Section 2 gives a comparative perspective of our approach relative to other similar approaches. Section 3 models the allocation-optimization problem, and presents a solution algorithm. Section 4 adapts this model to the demands on high-end multimedia servers. Section 5 discusses how to integrate the proposed mechanism into off-the-shelf OSs, and performance issues related to the model. The paper concludes with Section 6.

2 Related Work

The multimedia architecture developed as part of the *Pegasus* Project at Universities of Twente and Cambridge [1, 10] addresses several questions related to the problem treated in this paper. In particular, their focus on adaptive QoS supports our adaptive QoS server architecture.

Mullender and Sijben [10] presented an architecture for dynamic QoS multimedia applications. They emphasize the need for adaptive QoS when considering the transmission of continuous media. One of the main motivations for their implementation of the *Nemesis* kernel [1] was to provide resource protection for competing multimedia applications against each other and support for QoS adaptation on-the-fly. The Nemesis kernel implements the desired functionality by offering an easy-to-use thread abstraction that allows external triggers for changes in the main function of a thread. It also implements the management of resource access rights on a per-process basis. Since most of the work is done in threads, the application can build its own thread scheduling and adaptation mechanisms on top of resource shares granted on a per-process level. Process-level guarantees are very firm. Thus, unlike most other OSs, Nemesis' virtual resources can be assumed to have a fixed capacity and are much more predictable than the virtual resources in today's standard OSs. The strong support for resource

[1] This differs from "best-effort" service in that clients will receive predictable QoS by specifying their acceptable QoS levels and passing admission tests.

shares and on-the-fly adaptation allows us to consider the allocation problem at a higher level of abstraction. In this sense, our work is complementary to the *Pegasus* Project, since our model deals with on-line adaptation, assuming that resource reservation can be implemented and the server code — that implements different algorithms — can be changed on a per-thread basis. In Section 5 we will discuss to what extent it is possible to achieve our goals on standard OS without requiring any non-trivial change to the OS itself.

The solution by Real Networks Inc. for streaming continuous media over the Internet is widely deployed in broadcasting live contents over the Internet [9]. Their audio and video servers use statistical information about the end-to-end connection between their RealAudio server and the client to accomplish a stable continuous media connection. The scheme depends mainly on buffering at the client side to mask the effects of varying network loads on the client-to-server connection. This is one building block in load-sensitive QoS provision. Nevertheless, buffering will only help mask temporary changes on the client-to-server connection. If a connection has become a permanently congested or stays congested for an extended period, it would be desirable to upgrade the QoS for other clients whose connection could carry additional information.

RealAudio's allocation policy is what we refer to as the *Null* model or best-effort. It simply checks whether the client's request can be granted without considering the possibility of degrading existing connections for a new client. This leads to low system utility under light load and only a small number of clients served under heavy load, as we shall see in Section 5. Furthermore, RealAudio primarily supports broadcasting one stream to many clients, but it does not provide any powerful means to serve many clients with CD-player-like semantics (start, stop, jump to, skip), where every client-to-server connection is unique. Adaptation to overload, like giving higher-paying clients better service or maximizing the number of clients served, is not implemented.

A similar architecture — only for video — is the Oracle video cartridge [8]. It avoids the unpredictability caused by a platform that does not support QoS guarantees by running one and only one application on the platform. Since the Oracle video server is designed to be ignorant of the contents it is shipping to the clients, one cannot expect the server to dynamically adjust the QoS in an application-sensitive way. In fact, once a connection is established, the server "pumps" the video at a constant rate according to the parameters that were agreed upon at the time of connection establishment. At connection-establishment time, the client and the server negotiate the versions of the multimedia contents that the client will receive, thus providing some basic adaptability. Unlike Nemesis, Orcale's architecture does not allow for on-the-fly adaptation. Buffering is its only means to adapt to changes in the environment. As stated earlier, this is useful for dealing with transient overload conditions and transient network failures, but fails to increase the availability of service or maximize the client-perceived utility. In contrast to the RealAudio architecture, Oracle's video server is designed to support playback semantics. For this reason, we also find extensive on-demand video retrieval and interface capabilities all built into the server.

RealAudio and similar on-demand services would be more attractive if they could exploit system support for dynamic, utility-based session management (that we propose here).

3 The System Model

The service model plays a key role in solving the allocation problem as well as in implementing the service. The model also affects both the user's perception of the service and the provider's ability to configure the server. Any adequate service model must:

- Guarantee clients to receive service at one of the several QoS levels they specified,
- Allow the clients $\rightarrow$ QoS levels mapping to be changeable without terminating any ongoing session,
- Represent the fact that upgrading the QoS of clients may lead to different rewards, depending on the client's perceived utility,
- Optimize allocations with respect to the total reward to be accrued while accounting for the associated overhead,
- Permit service providers to make tradeoffs among maximal total reward, service availability, and average connection-establishment overhead; and
- Allow clients to define their perceived utility at various QoS levels.

We have developed an abstract model that meets all of the above requirements. The model can be broken down into three main parts: the service definition (*service model*), service perception (*subscription model*), and the service provision (*interaction model*). The service model enables the provider to express his server's capabilities to the OS or a middleware layer, depending on the implementation of the **FluSS** model. The subscription model enables the provider to specify service

classes available to clients, including QoS levels and the corresponding fees. The interaction model specifies how the server should implement session setup, tear-down, and adaptation with respect to client arrivals and departures. Resource failures may be integrated into this model as well.

3.1 Service Model

The server's ability to maintain a given QoS level for an individual session depends on the availability of necessary resources on the server. At this point the mapping of a multimedia service to specific technology becomes relevant because it determines the amount of resources consumed by each client at a certain QoS level. Although this mapping is not trivial (as discussed in Section 4), here we will assume its availability. Furthermore, we want to scale resource requests with respect to the total amount of a resource available. For example, if there are one million CPU cycles per second to be allocated in "1000-cycle units," then one unit represents 0.1 % of the CPU resource. This simple representation helps separate resource-allocation algorithms from the particulars of the underlying technology. By not committing to a particular implementation of resource shares, we define an OS-independent, implicit representation of resource allocation. Our particular percentage resource shares are tailored to bandwidth allocation. The algorithm presented in Section 3.4.1 works directly with any additive resource measure. Otherwise, it only needs to be slightly modified. In general, to abstract from particular implementation issues, we define a *service class* as the aggregation of code that needs to be executed to achieve a certain level of QoS along with a measured amount of resources that it requires to satisfy some high-level QoS constraint.

Only if all resources were local then a service class would represent the resources needed at a local node. However, since a distributed collection of workstations may cooperate to service a client, a resource-reservation "vector" is used to represent resource percentages at all nodes involved. It has to account for all resources at all involved nodes that need to be reserved for service provision at a specific QoS level.

A multimedia server is assumed capable of providing a certain service in several different ways, and clients are assumed capable of receiving service at any of the QoS levels specified, unless they are above a certain maximal level, beyond which the client's resources are exhausted. This is reasonable, because clients who want to interact with a specific server are usually willing to install the necessary software on their system that will manage QoS adaptation. Furthermore, we are not arguing for improved client-server inter-operability but for QoS adaptation. Support for different service provisions must therefore be designed to increase the server's flexibility in dealing with client arrivals and departures, and resource failures.

Definition 1 (Service Class) *For a server that supports b service classes, service class $i \in \{0, 1, \ldots, b-1\}$ contains information about which part of the server code to run, its parameters, and the corresponding resource consumption on the server to maintain a session for each client receiving service in that class. In case of k resources and b service classes, we define the function* **rescon**: $\{0, 1, \ldots, b-1\} \rightarrow [0, 1]^k$, *with* $rescon(i) = (x_1, x_2, \ldots, x_k)^t$, *where $rescon(i)$ represents the amount of all resources needed to deliver service to a client in service class i (t represents transposition).*

Note that the definition of a service class alone does not imply anything about the user-perceived QoS, which can be achieved in many different ways. For example, to transmit a video, one may use more CPU cycles to process (e.g., compress or smooth) the video signal, thus reducing network-bandwidth requirements; or use less CPU cycles, thus requiring more bandwidth. The actual optimization problem turns out to be much easier when there are fewer such tradeoffs and the canonical preferences over *service classes* coincide with the order achieved by the $\leq$ ordering on the resource consumption vectors for each service class. An example will be given in Section 3.2.

3.2 Subscription Model

Having discussed client preferences over service classes, we now want to have a closer look at the client side of our model.

Clients can be classified into groups, each with similar preferences on the multimedia signal quality. By increasing the number of groups, we can make finer distinctions between user preferences. Thus, grouping clients together is not a serious limitation in dealing with multimedia clients. A group of similar preferences is called a *subscription class*. Such grouping is used in many real-world settings. For example, one can lease different-bandwidth lines from public network providers, each offered at a specific price/rate. This rate is the same for all clients who want to receive service at a particular level of QoS. In case of QoS upgrade or degrade, clients within a particular *subscription class* are
discriminated.

Most models that offer group distinction exert their admission control by identifying the utility a group has for different service classes and setting a charge rate for

that group. The group membership charge rate regulates access to the service. On the client's end, his subscription represents agreement with the group's pre-defined preference relation.

Our admission-control scheme differs from dynamic pricing models because the price for a service is known in advance, but the actual QoS doesn't have to be. Such flexibility is not feasible in dynamic pricing models, because the possibility of receiving better service would already be figured into the price, so the client would be very dissatisfied if he were ever served at lower quality than what he paid for. This inflexibility is too limiting for adaptive services. Another argument against dynamic pricing as a means of admission control for multimedia services, is the prohibitive overhead of price negotiation. This is true, especially when the service is requested frequently, like audio on-demand.

For the above reasons, slowly-changing subscription prices (or membership fees) charged over many distinct service sessions appear to be the best choice. They allow more flexibility in changing the service class for each client during an active session. In general, prices and client preferences are external to the multimedia server. We will use a combination of prices and client preferences as a measure for the utility the provider receives from serving clients of a specific subscription class. The price of a subscription class corresponds to the utility a client has in order to be fully satisfied with the QoS he receives. When service to the client is degraded, his utility does not drop to zero immediately but decreases according to a relative utility function u with $Image(u) \subset [0,1]$ and the domain of all possible service classes. The relative utility helps the provider recognize how the client's utility reacts to changes in the service class.

Now, we can define the absolute utility function U (Fig. 1) as the product of the price of a subscription class and its relative utility function. We also ensure that the utility does not saturate after the client is being served at his "highest satisfaction" service class (u reaches 1), but we distribute an additional unit ("bonus") of utility equally over all remaining service classes, for which the client has sufficient resources. This is done to ensure that a client can receive better service than he pays for, if the server can provide it. There is one utility function u and one price for each subscription class.

Example: A service provider may decide to offer three subscription classes: CD quality, FM quality, FM or CD with natural preference for CD. Each of these will be priced differently. Assume CD quality costs \$50 per month, FM \$20 per month and the mixed subscription class \$40 per month. Assuming that the multimedia system can provide AM, record, FM, CD qualities — these are the service classes — we still need to define a relative utility function for each subscription class in addition to the price of a subscription class (Fig. 1(a)).

Our model relies heavily on the determination of clients' sensitivity to changes in such QoS as frame rate, resolution, etc., since we want to use the absolute utility function U (Fig. 1) defined for each service class, as part of an objective function for the evaluation of allocations, i.e., client $\rightarrow$ service class mappings.

Definition 2 (Subscription Class) *Suppose there are t active clients and the service provider offers v subscription classes, $s(i) \in \{0,\ldots,v-1\}$, and each client $i \in \{0,1,\ldots,t-1\}$ is associated with exactly one subscription class $s(i)$. Each subscription class j is specified by the price price(j), its corresponding relative utility function u_j, and a timeout. The relative utility function is specified over all possible service classes offered by the multimedia server. "Timeout" specifies the amount of time for which a client of subscription class j is guaranteed to receive service. The price and timeout of a subscription class may be subject to change, whereas the utility function cannot be altered once a subscription class is instantiated.*

Note that these subscription classes are maintained on the server. Contracts between the provider and the client are established on the basis of particular instantiations of subscription classes. Subscription class instances should, therefore, closely model clients' real preferences. Furthermore, the server should devise the utility functions so that it may maximize aggregated system utility while providing high service availability.

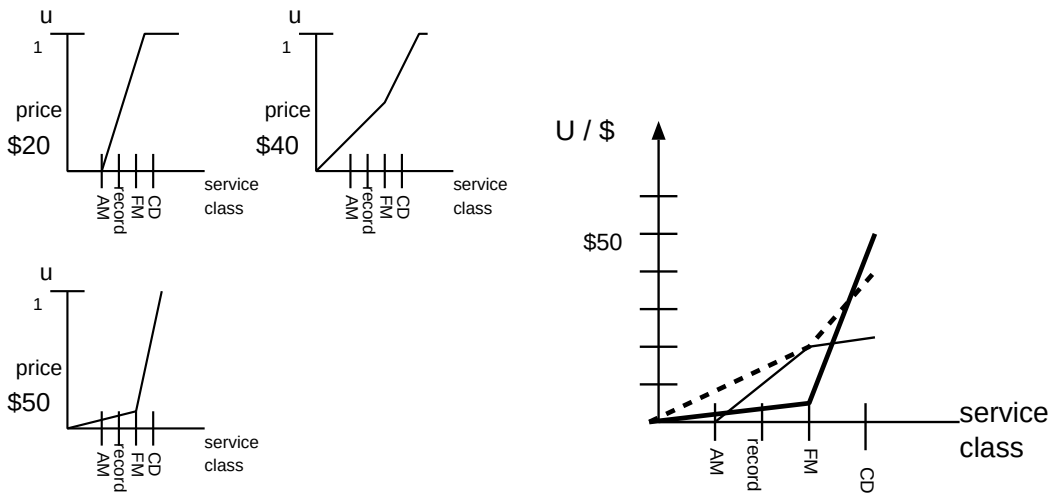
3.3 Subscription-Model $\rightarrow$ Service-Model Mapping

An *allocation* is a mapping of active *subscription class* members to service classes, and it must satisfy the following two conditions:

AC1: Each client receives service at exactly one QoS level specified by exactly one service class. No client receives service in the trivial class **null** (no service), unless the client's connection timed out. The function *sclass* that maps a client to a service class takes the client's ID as a parameter.

AC2: In case of t clients, the following constraint must be satisfied:

$$\sum_{i \in \{0,1,\ldots,t-1\}} rescon(sclass(i)) \leq (1,1,\ldots,1)^t$$



(a) Subscription class definition

(b) Derived absolute utility

Figure 1: Subscription class and absolute utility.

where $\leq$ is a vector comparison in which the left-hand side is less than or equal to the right-hand side in every component.

These conditions ensure that (i) an allocation specifies a service class for every client and (ii) resources will not be over-booked. Note that resource allocation is done implicitly by chosing a service class for a client. Resource constraints restrict the set of possible client→service-class mappings.

Our discussion of the model and the algorithm focused mainly on QoS adaptation to client arrivals and departures, but the model is general enough to accommodate other sources of change, such as the one caused by failures. In case of a resource failure, **AC2** may be violated and the allocation algorithm or the interaction model has to guarantee that **AC2** will be met as soon as possible, upon detection of the failure.

3.4 The Interaction Model

We now describe the core part of the model by formulating the problem to find an applicable solution. The resource-allocation problem turns out to be a slight modification of the well-known *knapsack optimization problem* [6]. Since the knapsack problem is known to be NP-complete, we will relax it somewhat for on-line use. If we strictly maximize aggregated system utility, the optimization problem has to be solved every time a client enters or leaves the system. (The same applies in case of resource failures.) The optimization problem for t clients is to *maximize* $\sum_{j \in 0,1,\ldots,t-1} u_{sj}(sclass(j))$ by changing the *sclass* mapping, subject to **AC1** and **AC2**.

If the dynamic programming is used, the time complexity is typically in $O(SIZE * M)$ where M is the number of items from which we can choose and *SIZE* is the size of the knapsack. The items from the knapsack problem correspond to the service classes in our problem. Each client has to be served at exactly one of the service levels offered by the server. The profit function defined in the knapsack problem corresponds to the absolute utility of a service class. A more difficult problem is to identify the size of the knapsack. For simplicity, we will first look at the one-resource case in which it is easy to identify the size of the knapsack. (This will later be generalized to the case of multiple resources.) If the resource could be allocated at the granularity of $1/x$ ($x \geq 1$) relative to the total amount of a resource available, we can dedicate the resource to each client at x different consumption levels. The size of the knapsack would be x, which is usually not very large, but as we add one more resource, we can dedicate one more resource to each client. Assuming addition of one resource, we have to multiply the reciprocal of the granularity of the first resource by that of the newly-added resource to get the size of the knapsack for the extended problem. This implies an exponential growth of the problem in the number of resources considered. The exponential growth in the optimization space as the resource-allocation granularity becomes finer, makes the problem intractable. So, we propose an optimistic approach to solving this problem.

3.4.1 An Optimistic Approach to Solving the Allocation Problem

In the one-resource case, the problem is equivalent to the simple knapsack problem. Our algorithm acts just like the dynamic programming solution for this one-dimensional resource problem [5]. Unfortunately, its complexity is higher than in the classical case because we have to enforce the additional constraint **AC1**.

OptCycle:

1. Pick the bottleneck resource for which the allocation will be optimized.
2. Pack the resource optimally for the first client under all levels of resource availability. Make sure that AC2 holds for every solution computed. Remember the maximum utility achieved for each allocation level of the resource considered.
3. Add next client to the allocation, and try to give him all possible shares of the considered bottleneck resource. Allocate the n-1 clients first by deciding the best utilization they can get out of the remaining resource share of the considered resource. Now, pick the best service class for the new class that is feasible. The optimal allocation is now computed by finding the maximum utility that can be achieved over all levels of resource allocation to the new client.
4. If there are more clients then go to Step 3 else output the optimal allocation.

Step 1 of the *OptCycle* algorithm may be non-trivial but we assume that in most cases the heuristic of picking the most heavily-overloaded resources may work well. Note that there are sophisticated heuristics for finding the bottleneck resource but the main goal of the "optimistic approach" is to reduce the problem's inherent complexity.

3.4.2 Remarks on the Optimistic Approach

A simple example shows that the above simplification weakens its applicability: we do not allow network bandwidth to be traded for CPU bandwidth, and vice versa, because we have to consider the allocation of both resources to optimize utility. More specifically, the amount of network bandwidth allocated to one client may affect the maximum utility achievable for the current client, but our algorithm might only be looking at the CPU resource to increase utility. (The resource tradeoff problem will be addressed in a separate forthcoming paper.)

If it is possible to determine the critical bottleneck resource — overbooked and not a candidate for resource tradeoffs — then we are again facing the "simple" one-dimensional version of the knapsack problem because in this case **AC2** would kick in only for the bottleneck resource. Thus, packing the bottleneck resource optimally becomes the goal, which is exactly what the optimistic approach implements. Because real systems often encounter one bottleneck resource at a time, the optimistic approach will perform well in realistic settings.

3.4.3 Need for a Simpler Solution

Despite the significant complexity reduction achieved, one may question whether this algorithm will perform efficiently even if client arrivals are as frequent as 1 Hz and higher. To obtain an applicable solution, we need to adjust the model to fit the demands on multimedia servers, which should be able to support at least 100 clients per server. The admission algorithm should be done, on average, well within a second. *OptCycle* might not cope with such demands if resource granularity is too fine.

An important fact to keep in mind is that *OptCycle* exhibits pseudo-global characteristics. For example, it is possible that the service class of every client will be changed. This adjustment requires changing scheduling parameters and potentially switching to another service algorithm and client adaptation, inducing significant overhead. We will in the next section reduce this overhead and show how its reduction influences the quality of resource allocations.

4 Efficient Solution to the Allocation Problem

Since it is time-consuming for the *OptCycle* algorithm to reallocate client sessions, we introduce a *short-term repository* (STR), from which new clients draw some resources until their admission decision is made; the resources needed for the STR are reserved in advance. We propose use of a STR for two reasons. First, it is time-consuming to make an admission decision, as it may have to invoke an optimization process. Second, the number of clients served without a STR is less than that with it, because new client arrivals might not increase aggregated utility sufficient enough to be considered "admissible," but should not be rejected so as to keep the rejection rate low. We also introduce a "cache" for connections that have just been terminated, so that a new request may take the place of a client who has just left, without triggering a time-consuming optimization process.

4.1 Short-Term Repository and Free-List

Instead of waiting until the admission test completes, we will set aside a fixed amount of resources for newly-arriving clients. The STR represents resources that are reserved on the server but not used to serve existing clients. Its amount is so chosen to accommodate the mean number of clients arriving during the execution of the admission test. A newly-arrived client that cannot receive any more resources except for those allocated to him from the STR will reduce his requirements. If more resources become available, the STR will be replenished before making any other changes to the allocation. The downside of this order of steps is that the maximum aggregated utility for the server is reduced because the resources in the STR cannot be used to increase the utility of active clients.

Another way of admitting clients is to use a *free-list* (FL). Instead of degrading active clients' QoS to accommodate a new client, we may take advantage of the fact that a client of the same subscription class, thus, by definition, sharing the same preference relation, may have left the server a very short time ago. Thus, the new client can take the place of the client who just left without triggering any adaptation operation, thus reducing the number of times the *OptCycle* algorithm to be invoked, and decreasing the overall adaptation overhead. Every time the *OptCycle* algorithm is invoked, all resources on the FL will be made available for improving the service class of active client sessions.

4.2 Session Termination

Timeouts need to be implemented in the server; otherwise, a client could block others from accessing the server forever. In the OS context, this problem is typically referred to as starvation. Taking a timeout is the first step towards terminating an active client session. Upon expiration of a client's time quantum, he is put on a *droppable-list* (DL). Once on this list, the client's absolute utility will start to decline steadily to ensure that he will eventually be dropped to accommodate new clients. The introduction of a DL is reasonable since it would not make any sense to terminate a client session immediately if nobody can benefit from the freed resources.

The resources allocated to those clients, who do not resign themselves in time but leave or are dropped from the DL, are not put on the FL. If this were not the case, a client could resume immediately after being disconnected, as his resources would be "buffered" in the meantime. Of course, the most common source for session termination will be the client's "logging off." Once a session of subscription class i is closed, we will put the resources reserved for that particular session on the FL for subscription class i, where the reservation will be kept for a predefined time period or until a new arrival uses it.

4.3 Admitting New Clients

The most convenient way to admit a new client is to take resources from its subscription class' FL. So, checking the FL is the first step in the admission-control algorithm. If this step is successful, the algorithm takes and reserves the resources from the FL, allocates them to the new client, and exits.

Another straightforward way to allocate resources to a newly-arrived client is to satisfy its resource requirements directly from the DL. Clients on the DL will be dropped in increasing order of their maximum utility. All of the k lowest utility client sessions whose aggregated utility is less than the new client's maximum, will be dropped, as they contribute to the congestion of the server. More specifically, the ratio of reward received from maintaining those "expired" sessions to the resources these sessions consume is unacceptable.

If the admission algorithm fails to allocate the new client by using the above two steps, it allocates resources from the STR to it, which will suffice to start serving the client at the lowest service class defined for the server. We will then start degrading active client sessions for the new client. If there are no clients who, if degraded, lose less utility than the gain by upgrading the new client to the next higher service class and also free sufficient resources for the upgrade, we leave the new client at its current level. Otherwise, we upgrade its service class again.

4.4 Periodic Check

Since the proposed model uses timeouts "generously," we want to guarantee system integrity only after periodic system checks. If we were to deal with timers while serving the clients, it would be impossible to serve many clients simultaneously. Periodic integrity checks may lead to inconsistencies between checks, e.g., clients whose timeout has occurred could still be receiving full service.

Periodically we release all resources that were tied up on FLs for too long, drop clients on the DL whose utility is almost zero, and run *OptCycle* to upgrade clients' service classes. We also need to replenish the STR if the resources allocated for it do not match its specified size. The period of this check is an important design parameter, and is defined in terms of how many adaptation operations the greedy strategy may perform before *OptCycle* must be run again.

4.5 Analysis of the Hybrid Approach

Our choice of a mixture of greedy and global (*OptCycle*) strategies for the **FluSS** model is based on the comparison of algorithmic complexity of both optimization strategies. Despite its simplicity, *OptCycle* still has a strong negative influence on the system performance, which we were able to observe as the simulation speed declines. The time per simulation run greatly depended on the frequency at which *OptCycle* was invoked. This observation is backed up by our time complexity analysis.

The complexity of the *OptCycle* algorithm depends on two parameters: the number of clients, n, and the granularity of the server's resource, $g \in [0,\ 1]$. The number g represents the minimal amount of the resource that can be allocated to a client.

Since the dynamic programming is used to solve the allocation problem, a table of size $O(n/g)$ must be constructed. An entry, (n_i, g_i), represents the optimal allocation for all n_i clients who were given g_i of the bottleneck resource. When creating new entries, we have to ensure that **AC2** holds, possibly leading to a search through all $1/g$ rows of the previous column. However, this is unlikely, since we are assuming abundant supply of all resources, except for the bottleneck. Nevertheless, to guarantee the correctness of the algorithm, we still need to check **AC2**, which will have a significant impact on the worst-case analysis for the algorithm. Thus, for n clients the complexity of *OptCycle* is computed as the cost of building the table, which is $O(n/g^2)$.

For the greedy approach, in the worst case all n clients are at the lowest service level and can all be upgraded to the highest service class. Each upgrade operation step costs $O(\log(n))$ per-client, since we will keep clients on a priority heap, ordered by their potential utility increase and $O(\log(n))$ is the cost of maintaining the heap condition after each insertion. An insertion must be made every time a client's potential utility gain changes, i.e. after each upgrade operation. The greedy approach for degradation is defined analogously. In this way, we compute the complexity of $O(n \log(n))$. Note that the number of service classes only accounts for a constant factor in the complexity analysis, since we can upgrade each client by at most the number of service classes that the server offers.

5 Application and Evaluation of FluSS

This section describes the application scenario that inspired **FluSS**, presents the advantages of **FluSS** over the common practice in multimedia server design, and evaluates the effects of server design parameters.

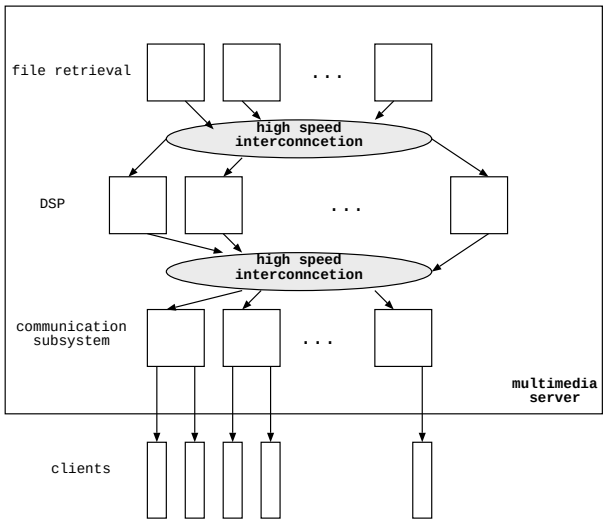


Figure 2: A multi-stage server.

5.1 A Multi-Stage Multimedia Server

In the *OptCycle* algorithm we made the implicit assumption that resources can be effectively allocated to individual server threads. This assumption doesn't hold for almost all of today's commercial OSs. The main difficulty lies with their inability of reserving resources, thus causing interferences among the server threads. For example, reservation of hard disk bandwidth for one thread will affect the amount of memory bandwidth available to another thread. The amount of cross-resource interference tends to be so significant that no simple solution for resource reservation may be derived and implemented. However, there is an exception: the problem of reserving/allocating CPU cycles is very well understood, solved and implemented in most modern OSs. POSIX [3, 12] defines the *de facto* standard for scheduling threads and enables the programmer to change the time-sharing policy for his threads from user-space. We propose to enhance the scheduling capabilities of POSIX-compliant OSs slightly to provide resource-reservation mechanisms necessary for **FluSS**, i.e., fair-share scheduling [7].

Our solution differs from conventional multimedia sever implementations in that, instead of integrating all functionalities into one node (workstation or PC), the service is broken up into functional components that would interfere with each other if they were run on the same node. So, we decided to distribute them over several nodes. Furthermore, each unit (Fig. 2) is built

around one particular bottleneck resource. This design makes it easier to ensure that all resources, except the bottleneck resource, are abundant at the local node. The composite server functionality is achieved by combining the components in a pipelined fashion. The performance of this "pipeline" will depend on the performance of the slowest component in the chain. Finding and optimizing the slowest component in the chain guarantees the *Opt-Cycle* algorithm's optimality. Note that separate boxes in Fig. 2 may be either separate workstations or separate processing elements integrated on one board. The only requirement is that the interconnection network should not be the bottleneck for the server, i.e., should connect the individual stages of the server with sufficient bandwidth, and the processing units should not interfere with one another.

Decomposition is the first step in providing general resource-reservation mechanisms. The second step is to exploit the available resource-reservation mechanisms — ideally fair-share reservation — to reserve different shares of the bottleneck resources at the involved nodes. If fair-share reservation is unavailable for a particular resource we must resort to mapping fair-share resource allocation to fair-share CPU allocation. The idea behind this mapping is that the frequency at which a thread can consume resources depends on the CPU share it receives. Hence there must be a CPU quantum to be allocated to a thread so that it receives its reserved share of the bottleneck resource. Note that this step can be skipped if the node runs an advanced multimedia OS like Nemesis [1].

One may ask: "Given a desired thread performance, as defined by the **FluSS** model, how does one optimally configure the scheduling parameters for this thread in order to achieve the target performance?" Although the answer to this question is important, we will only briefly describe a potential solution (Fig. 3), since the derivation of a complete answer is beyond the scope of this paper.

Since OSs must handle many unpredictable events like interrupts, failures, branch-prediction in the processor, etc., one cannot determine the mapping analytically. Instead, one has to measure the performance experienced by the resource-consuming thread subject to different scheduling parameters, such as minimum time-slice, priority, etc. The thread should therefore be instrumented for efficient monitoring and feedback mechanisms, which will help find the optimal scheduling parameters for a desired level of QoS. Franken's thesis [2] presents a model that achieves the needed mapping by monitoring and stochastic modeling. If the scheduling is as simple as fair-share scheduling, which only depends on the parameters thread latency and CPU utilization, the mapping can be accomplished efficiently.

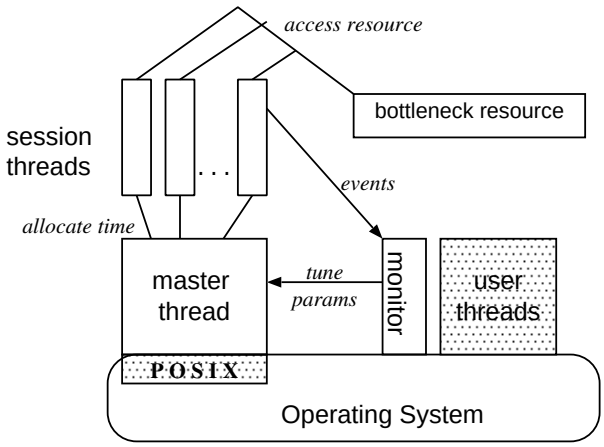


Figure 3: Potential node setup.

5.2 Simulating FluSS

We now want to evaluate the goodness of **FluSS** via simulation. The simulation results were very stable as the server activities were simulated for 27 hours, achieving less than 2% variations in our measurements of long-term averages. Due to their highly system-dependent nature, we determined the trend of utility, adaptation operations, and clients-in-service for different choices of system design parameters.

5.2.1 Simulation Parameters

To give a rough estimate of the performance of the proposed algorithm in realistic settings, we modeled the server using the information on RealSystem 5.0 [9], Real Network's audio-on-demand server. This widely-used system supports four service classes: 14.4, 28.8, ISDN, and Dual ISDN. According to Real Networks, it is possible to support more than 1,000 sessions per server PC at 14.4 kbps when broadcasting live contents. With individual streams the performance is significantly lower. Due to the unavailability of any more accurate information about the number of streams supported for "canned" contents, we modeled a server that can support up to 128 sessions at 14.4 kbps. This requires a bandwidth of 230 KB/s, which is manageable by a single workstation server. Table 1 shows the modeling parameters. We generated only one bottleneck resource, since the bottleneck-determination problem was not within the scope of this paper.

Furthermore, we defined the time granularity for the simulation to be 2 ms, so the server cannot handle client arrivals closer together than this. We chose 2 ms because it is a lower bound on the time in which we can receive a request for service, do the first "greedy" admission test,

serv. class ID	res. cons. vector
1	(0.0078125, 0.00390625, 0.00390625)
2	(0.015625, 0.0078125, 0.0078125)
3	(0.0234375, 0.015625, 0.015625)
4	(0.03125, 0.01953125, 0.01953125)

Table 1: Service class description

ID	$	Timeout/min	utility vector
1	15	45	(0.95, 0.98, 0.99, 1.0)
2	30	45	(0.3, 0.6, 0.98, 1.0)
3	50	45	(0.1, 0.6, 0.8, 1.0)

Table 2: Subscription class (SubCs) description. (utilities as utilities for each service class)

Load	SubCs1/s	SubCs2/s	SubCs3/s
light	0.005	0.02	0.01
low	0.02	0.07	0.03
high	0.035	0.1225	0.0525
heavy (overload)	0.05	0.2	0.1

Table 3: Arrival rates for different SubCs at different load levels.

Load	FluSS	Null
low	0.00	0.12
high	0.00	0.27

Table 4: Average rejection rates: **FluSS** versus **Null**

then generate a reply. This bound will be exceeded as soon as clients' workstations do not reside in the same workstation cluster as the server.

To get a handle on client inter-arrival times and session durations, we assume they are exponentially-distributed. This assumption is consistent with most queueing models of client arrivals. The parameter for session durations was the same for all service classes in all simulations and used the parameter 6×10^{-7} (a mean of 45 min). We evaluated the performance of **FluSS** under high, low, and very light loads. The corresponding parameters are given in Table 3.

The server resources considered in the simulation are CPU, hard disk, and communication (in that order of appearance in the resource vector). The CPU was considered the bottleneck resource because it is involved in signal enhancement, data retrieval, and protocol processing. Note that the bottleneck resource depends on the server setup.

The lowest quality is near AM radio with a 14.4 kbps connection. At 28.8 kbps one receives quality between AM and FM radio. Finally, ISDN facilitates true FM quality. Using both channels of ISDN will guarantee near-CD quality. Each of these QoS levels is represented by a service class. Clients are assumed to be capable to receive service at all of these levels.

The modeled server offers three different subscription classes. The lowest quality might be of interest to those who want to access voice transmissions, such as radio learning at 14.4 kbps for $15 per month. The second subscription class targets at those who want to listen to music at FM quality, ideally receiving service at 64 kbps. The cost is $30 per month. The last and most expensive class is for CD quality, and served best at 128 kbps. It costs a monthly fee of $50. See Table 2 for the exact utilities used in our simulation. The generated conditions for the simulation were mainly modeled after the motivating example in Section 3.2.

5.2.2 FluSS Enhances Service Availability

The **FluSS** model is compared against today's common practice in session management for multimedia servers that admits clients as long as there are sufficient resources to set up sessions for them. The server does not distinguish between clients, and simply tries to serve each client as best as it can. Furthermore, there is no concept of adaptation. Because of the model's simplicity we will refer to it as the **Null** model.

Intuitively, **FluSS** will outperform the **Null** model in terms of availability because it has the option of degrading clients' QoS, which is not supported by the **Null** model. What is surprising is the extent to which **FluSS** outperforms the **Null** model. Table 4 shows that the simple policy does not scale well to high-load environments whereas **FluSS** does. Lack of adaptability to load variations is "handled" by excessive over-design in current multimedia systems. By contrast, flexible QoS allows the service providers to tailor their systems more to their real needs, thus drastically decreasing the cost of the server and eventually the clients' cost.

Moreover, **FluSS** is found to be able to serve 30% more clients under low load than the **Null** model (40 as opposed to 30). Under high load this gap widens even more to 233% (70 versus 30). As we approach the overload region, the two policies drift apart even farther. The number of clients served in the **FluSS** model is sensitive to the removal of the short-term repository. Its removal decreases the average number of clients served by 10% and results in a non-zero rejection rate. This negative impact is amplified as load increases. Clearly, adapta-

tion and the ability to admit clients on a "conditional" basis improve the server availability notably.

5.2.3 Allocation Improvements

Since we aim at improving the aggregated utility of allocations, we need to evaluate the utility improvements of **FluSS** over **Null** as well as the impacts of design parameters in **FluSS**.

As expected, **FluSS** outperforms the **Null** model (Fig. 4) by 15–25%, depending on the choice of system design parameters. There is a clear anti-proportional relation between the inter-*OptCycle* interval and the achieved aggregated utility. This is because the local strategy might get stuck at local extrema if the solution is not optimized from time to time by a global strategy. The decline in the aggregated utility is larger under higher load, to an extent that even the **Null** model achieves greater aggregated utility if *OptCycle* is not run frequently.

As expected, FLs have a negative impact on the achievable aggregated utility, as they were not introduced to increase utility but to decrease overhead. The aggregated utility with FLs ranks approximately 20% lower than without them (Fig. 5). this is significant with respect to the gain achieved so far, one must find a compelling reason for using FLs. The immediate decline of utility in the presence of FLs is due to the high arrival rates with which we have experimented. The lower the arrival rate the flatter will be the slope of the decline in utility in Fig. 5.

5.2.4 Trading Allocation Optimality for Less Overhead

Decreasing overhead effectively means improving the aggregated utility, since the resources saved on adaptation operations can be used to admit more clients to the server, which would then increase the aggregated utility. The overhead per session adaptation is not negligible, since it requires to change both the resource reservation and the the QoS level of a server thread.

Knowing that the aggregated utility decreases linearly with the time between *OptCycle* runs, we would like to confirm that the number of operations decreases faster so that we can make an effective tradeoff.

The decrease in the number of adaptation operations with the increase of inter-*OptCycle* intervals resembles a negative exponential function, meaning that its decline will be larger than a linear function at zero and less as we approach positive infinity. If we increase the time between *OptCycle* runs from zero to 15,000 adaptation operations, we notice a 10-fold difference in the number of adaptation operations (Fig. 6). The aggregated utility for the same interval only changes by less than 10%, indicating the need for trading optimality for a reduction of adaptation overhead.

We noticed in the discussion of aggregated utility that FLs have a negative effect (Fig. 5) on utility. This effect is compensated by the fact that while achievable aggregated utility decreases by 20%, the number of adaptation operations changes by more than 800% (Fig. 7), because every time a client is admitted through FL, adaptation is not necessary. Extending the time to keep resources on the FL does not decrease the number of operations any further beyond the optimal point, but it does not increase it either.

5.2.5 Finding the Optimal Choice of Parameters

FluSS requires a few parameters to be adjusted to enhance the performance of the multimedia server. Especially, the tradeoff between maximum utility and adaptation operations makes the choice of parameters difficult. So, we suggest to measure the number of CPU cycles consumed per adaptation operation. Once this number is determined it can be multiplied to the rate of adaptation operations and expressed in terms of resources consumed per second.

Now, to determine the utility gain from increasing the time between *OptCycle* runs or the introduction of free-lists, one has to figure out the resource consumption by adaptation operations executed at the new inter-*OptCycle* intervals. One can then compute the reduction in resource consumption achieved by the optimization step.

We compare the sub-optimality (in terms of utility loss) resulting from performing *OptCycle* less frequently against the average utility gain if the freed resources were used to upgrade clients' service classes. The expected utility gain can be determined by extending the simulation or a rough analytic approximation. By iterating this process the rough estimation may work well; otherwise, the parameters can be fine-tuned after the server is set up.

The choice of parameters is system-dependent, so it must be made for every newly-configured system. Our simulation must be fed with the parameters that fit the description of the new multimedia server and derive a proper set of parameters from the simulation results.

6 Concluding Remarks

The proposed **FluSS** model is found to have significant advantages over current best-effort schemes in terms of service availability and allocation. Flexible QoS is the

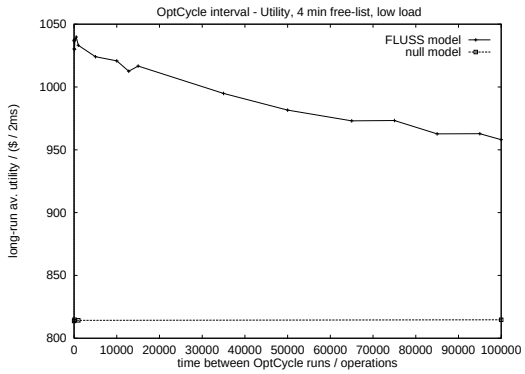


Figure 4. 4 min free-list, low load

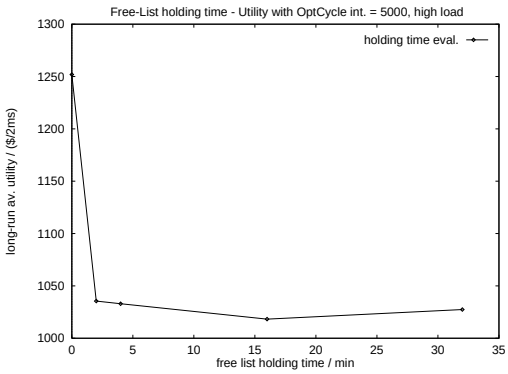


Figure 5. *OptCycle* interval 5000, high load

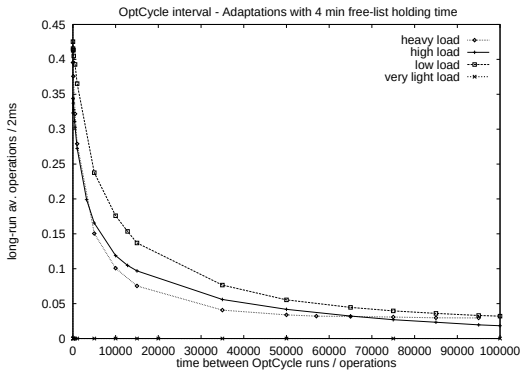


Figure 6. Four min free-list

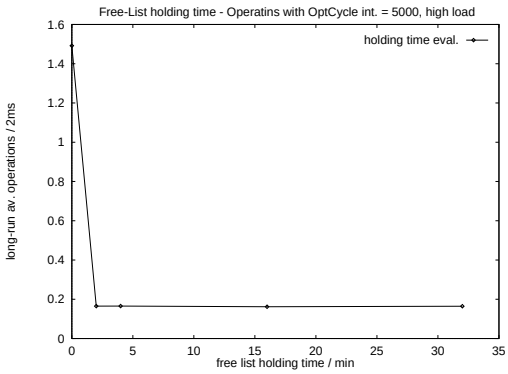


Figure 7. *OptCycle* interval = 5000, high load

key to building commercial multimedia servers. The resultant high service availability allows the provider to invest less money initially, because the server will still perform well even if it operates near its capacity. The provider can upgrade the server gradually to meet the need without excessive over-design. The fact that we were able to quantify the positive contribution of every component in the **FluSS** model is significant in that **FluSS**-like components can be used for adaptive session resource management, a key to future commercial multimedia systems.

We are currently exploring modifications of standard OSs to accommodate resource shares. This and the proposed session-management scheme together will eventually lead to building an advanced multi-stage multimedia server that supports adaptive QoS without requiring any non-trivial changes to commercial OSs. One of the design goals in building such a server is to use off-the-shelf OSs to show that, besides improving the service itself, adaptive QoS can also help reduce the cost of building high-end multimedia servers.

Acknowledgement

The authors would like to thank Hani Jamjoom of The University of Michigan for his contribution to the simulation tool and support during the early phase of this project.

References

[1] Robin Fairbairns, "Summary Report, Kernel Work Package", Esprit BRA 6865, University of Cambridge Computer Laboratory, 1993

[2] Leonard Franken, "Model-Based QoS Management", PhD. Thesis, Universiteit Twente, 1995

[3] Bill Gallmeister, "Posix.4: Programming for the real world", O'Reilly & Associates, 1995

[4] Reinhard John, "Definitionen und Sätze zur Vorlesung Entscheidungstheorie, Somersemester 1997", Rheinische Friedrich-Wilhelm-Universität, Bonn, 1997

[5] Thomas Leiserson, Charles Cormen, Ronald Rivest, "Introduction to Algorithms", New York, N.Y., McGraw-Hill, 1994

[6] Silvano Martello, Paolo Toth, "Knapsack Problems", Chichester, U.K., John Wiley & Sons Ltd., 1990

[7] Chen Lee, Raj Rajkumar and Cliff Mercer, "Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach" , In *Proceedings of IEEE Multimedia Systems*, Japan, 1996

[8] Oracle, Inc., "Oracle Video Cartridge", Technical White Paper, www.oracle.com, 1997

[9] Real Networks, Inc., "Real Networks, the Home of Real Audio, Real Video, Real Flash", http://www.real.com/, 1997

[10] Paul Sijben, Sape Mullender, "An Architecture for Scheduling and QoS management in Multimedia Workstations", Pegasus paper 95-5, http://www.pegasus.esprit.ec.org/default.html, Universiteit Twente, 1995

[11] Hal Varian, "Microeconomic Analysis", 3rd Edition, W.W. Norton & Company, New York, 1992

[12] Fred Zlotnick, "The Posix-1 Standard: A Programmer's Guide", Redwood City, CA, Benjamin/Cummings Publishers, 1991

Efficient Schemes for Broadcasting Popular Videos*

Lixin Gao
Department of Computer Science
Smith College
Northampton, Mass. 01060, USA
gao@cs.smith.edu

Jim Kurose
Don Towsley
Department of Computer Science
University of Massachusetts
Amherst, Mass. 01003, USA
{kurose, towsley}@cs.umass.edu

Abstract

We provide a formal framework for studying broadcasting schemes and design a family of schemes for broadcasting popular videos, called the Greedy Disk-conserving Broadcasting (GDB) scheme. We analyze the resource requirements for GDB, i.e., the number of server broadcast channels, the client storage space, and the client I/O bandwidth required by GDB. The analysis shows that GDB exhibits a tradeoff between any two of the three resources. We compare our scheme with a recently proposed broadcasting scheme, Skyscraper Broadcasting (SB). With GDB, we can reduce the client storage space by as much as 50% or the number of server channels by as much as 30% at the cost of a small additional increase in the amount of client I/O bandwidth. If we require the client I/O bandwidth of GDB to be identical to that of SB, GDB needs only 70% of the client storage space required by SB or one less server channel than SB does. In addition, we show that with small client I/O bandwidth, the resource requirements of GDB are close to the minimum achievable by any disk-conserving broadcasting scheme.

1 Introduction

Video is one of the most important communication media in all aspects of our lives. On average, each household has 1.4 television sets in the U.S.[3, Page216]. We turn to televisions to be informed, entertained, educated, or even to do business (e.g., shopping or video conferencing in offices). We wish to be able to make content available instantly at the touch of our fingertips. It was shown in [6, 7] that 80% of our demand is for a few (10 to 20) very popular videos. This drives future content providers to deliver popular videos to a large number of subscribers with the smallest latency possible. These popular videos can be "hot" news (such as the Oklahoma City bombing trial or Mars Pathfinder coverage), popular sport events (such as a U.S. Open final), taped lectures (during final exam period), or recently released movies.

In a VOD system, a server delivers videos to clients via a high speed network. Since video is an isochronous medium, a video server has to reserve a sufficient amount of network and I/O bandwidth for the video stream before committing to a client's request. We refer to the resources required to deliver one video stream while guaranteeing a client's continuous playback as a *server channel*. The simpliest scheme for scheduling server channels is to dedicate a channel to each client. However, this results in a tremendous demand for server channels in VOD systems. Therefore, two techniques for sharing a channel by more than one client have been proposed.

- **Sharing by batching requests for the same video together.** Here, a client makes requests for a video and waits until that video is eventually broadcast on a channel. When a channel becomes available, the server selects a batch of pending requests for some video according to some scheduling policy. While batching reduces the demand for server channels, it does so at the cost of introducing service latency.

- **Sharing by broadcasting a video via multiple dedicated channels.** Here, the client

* This work was supported in part by the National Science Foundation under Grant No. NCR-9729084. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

prefetches data from one of several channels for later playback. Prefetching is done in such a way as to ensure the continuous playback of the whole video. Broadcasting schemes take advantage of resources (e.g., disk) at the client end, and guarantee a service latency independent of the number of requests.

In this paper, we provide a formal framework for studying broadcasting schemes and design a family of schemes for broadcasting popular videos, called the *Greedy Disk-conserving Broadcasting* (GDB) scheme. The main idea of GDB is as follows. Each video is partitioned into segments. Each segment is broadcast periodically via a dedicated channel. We select segment sizes so as to ensure that the client can receive a segment before the playback time of the segment. Since a client can prefetch a segment during the playback of an earlier segment, the data closer to the end of the video can be broadcast less often than the data closer to the beginning of the video. Therefore, we partition the video into segments of monotonically increasing size. Since the maximum service latency (the time from the client request of the video until the start of its broadcast) is the length of the first segment, the later segments are greedily selected to be as large as possible so as to minimize the size of the first segment. Furthermore, to conserve the client disk space, a client receives a data segment at the latest possible time that ensures the continuous playback of the video.

We systematically analyze the resource requirements for GDB. In particular, we derive the number of server broadcast channels, the client storage space, and the client I/O bandwidth required by GDB. Our analysis shows that GDB exhibits a tradeoff between any two of the three resources.

We compare our scheme with the most recently proposed broadcasting scheme, *Skyscraper Broadcasting* (SB). We rigorously prove that GDB outperforms SB in terms of the resource requirements. Our simulation shows that with GDB, we can reduce the client storage space by as much as 50%, or the number of server channels by as much as 30%, at the cost of small additional client I/O bandwidth. If we require the client I/O bandwidth of GDB to be identical to that of SB, GDB needs only 70% of the client storage space required by SB or one less server channel than SB does. In addition, our data show that with small client I/O bandwidth, the resource requirements of GDB are close to the minimum achievable by *any* disk-conserving broadcasting scheme. Finally, we prove that the number of server broadcast channels required by GDB is close to the optimal achievable by any broadcast scheme.

The rest of this paper is organized as follows. In the remainder of this section, we review related work. Section 2 gives an overview of the problem. In section 3, we present and analyze disk-conserving broadcasting schemes. Sections 4, 5 and 6 propose and analyze GDB and compare it with SB. Finally, in Section 7, we conclude with ending remarks.

1.1 Related Work

The simplest scheduling scheme for a video server is to allocate a channel to each client. However, this scheme can quickly exhaust the most scarce resource in the server's system: the server network bandwidth. To avoid this, Dan et al. proposed to allocate on-demand channels for less popular videos and dedicated channels for popular videos [6, 7, 8]. The server uses on-demand channels to satisfy requests for less popular videos according to some batch scheduling policy, and uses dedicated channels to broadcast popular videos to clients. Both broadcasting and batch scheduling schemes allow a channel to be shared by more than one client as follows.

In a batch scheduling scheme, a group of requests that arrive close in time are batched together and when a channel becomes available, the server selects a batch according to some scheduling policy, satisfying the requests with the channel. Some efficient batch scheduling policies are studied in [6, 7, 2].

In a broadcasting scheme, the server broadcasts a video via multiple dedicated channels and each client follows a reception schedule to receive data from appropriate channels so as to continuously playback the whole video. The simplest broadcasting scheme is the EB scheme that divides the video into equal-length segments[6]. The EB scheme guarantees a worst case service latency equal to the length of one segment, independent of the number of of requests, since a client waits at most the length of one segment for the playback to start. Furthermore, the number of dedicated channels required for a video is inversely proportional to the guaranteed service latency.

To decrease the number of required channels, Viswanathan and Imielinski proposed an ingenious broadcasting scheme, called Pyramid Broadcasting (PB) [11]. The main idea of the PB scheme is to divide a video into segments of geometrically increasing size. Each segment is broadcast periodically via a dedicated channel that has a bandwidth greater than the playback rate. Each client saves the next segment of data while playing the current segment of video data. Since the service latency is proportional to the length

of the first segment, the PB scheme ensures an exponentially decreasing latency with an increase in the number of dedicated channels. The drawback of this scheme is that a client needs to have sufficient disk space to store more than 70% of one video file.

To address the problem of the disk space requirements, Aggarwal, Wolf and Yu devised a scheme called Permutation-Based Pyramid Broadcasting (PPB) [1]. The PPB scheme divides a video object the same way as the PB scheme and multiplexes several channels among one segment of video data. The similar partition strategy limits the disk space reduction and typically requires a client disk capable of storing at least 50% of a video's data.

Recently, Kua and Sheu proposed a broadcasting scheme [10], called Skyscraper Broadcasting (SB), that further reduces the client disk space requirement. The SB scheme uses a novel data partition strategy and exhibits a tradeoff between the client storage space and the server network bandwidth. However, these schemes have been proposed in an ad hoc manner, and there is no study of the tradeoff between the client I/O bandwidth and the server network bandwidth.

2 Overview of the Problem

For simplicity of exposition, we consider the problem of broadcasting a single video, since we can easily extend the scheme to multiple videos with a proportional increase of the number of the server channels. Consider Figure 1, which depicts a video server broadcasting architecture. The server broadcasts videos via dedicated network channels according to some *broadcast schedule*. The *number of server channels* is the total number of channels required for the dedicated network channels. The network uses a multicast communication facility to transmit the video streams to clients so that a client can select which channels it wishes to receive from [5, 9].

Each client contains a set-top box, a disk, and a display monitor. A client is connected to the network with a set-top box. The set-top box selects several network channels to receive video data from according to a *reception schedule*. The reception schedule indicates which channels to select and when, so as to ensure the continuous playback of the video. In this paper, we consider only *non-workahead reception schedules*, those which require that a client first request a video before it can receive any data for that video. The received video data are either stored on the disk or sent to the display monitor for immediate playback. The display monitor can either retrieve stored data from the disk or receive data from a channel directly. The *client storage space* is the maximum disk space required throughout the client playback period. The *client I/O bandwidth* is the maximum bandwidth required to store and retrieve data from the disk at any time. The *service latency* is the maximum amount of time that a client has to wait to start the playback once it requests a video. We use the following notations throughout the paper.

B: the server bandwidth dedicated to the video
b: the video display rate at the client
K: the number of dedicated server channels
D: the length of the video in seconds
l: the latency requirement

We focus here on continuous playback starting from the beginning of the video, deferring issues introduced by providing interactivity (such as fast-forward, rewind, and pause) to a future study. Our goal is to design broadcasting schemes that guarantee a service latency l and effectively utilize all three resources: the number of dedicated server channels, the client disk space and the client I/O bandwidth.

3 Disk-Conserving Broadcasting Schemes

All proposed schemes in this paper falls into the class of *disk-conserving broadcasting schemes*. The main idea behind the disk-conserving broadcasting scheme is to conserve the client disk space by letting a client receive data as late as possible. The three components of a broadcasting scheme are 1) data partition strategy, 2) broadcast schedule, and 3) reception schedule. We formally describe each of these in turn for the disk-conserving broadcasting scheme.

Partition strategy: The dedicated bandwidth B is divided into $K = \lfloor B/b \rfloor$ logical channels of b Mb/s each. A D-second video is partitioned into K segments $T_1, T_2, ..., T_K$ according to a partition function $f(n)$. The partition function $f(n)$ is an integer function (i.e., $f(n)$ is an integer) that defines segment T_i as follows. We let one time unit be $D/\sum_{n=1}^{K} f(n)$ throughout this paper for ease of the discussion. Segment T_n contains $f(n)$ time units of video data, and its data starts at the $\sum_{m=1}^{n-1} f(m)$ th time unit of the video and ends at the $\sum_{m=1}^{n} f(m)$ th time unit of the video. As we will see later, a partition function uniquely defines a disk-conserving broadcasting scheme. Therefore, the selection of the partition function is crucial for determining the resource requirements of the scheme.

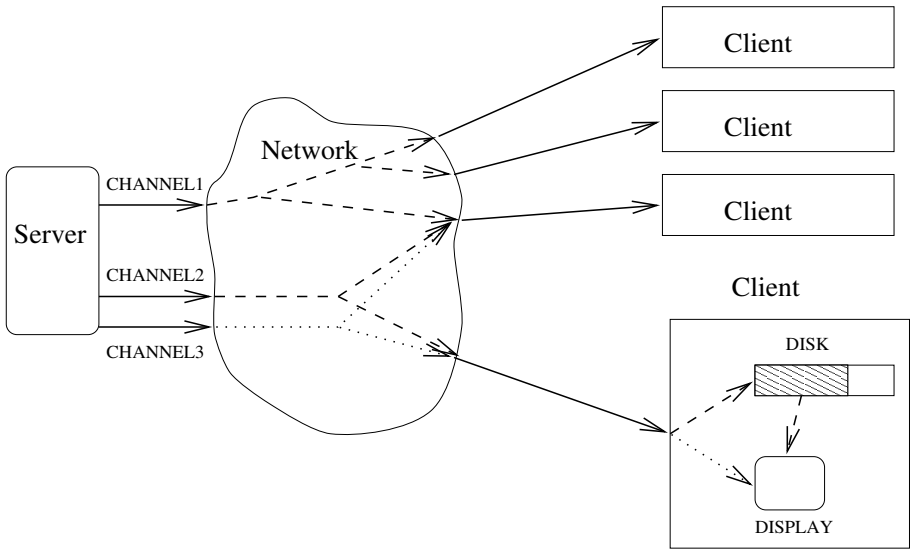


Figure 1: *Overview of the problem setting*

Broadcast schedule: The server broadcasts each segment periodically via a channel of bandwidth b starting at time 0. In other words, T_n is broadcast periodically starting at time $0, f(n), 2f(n), ...$ via channel n. For example, the τ th time unit data of T_n is broadcast at time $\tau, \tau + f(n), \tau + 2f(n), ...$ via channel n.

Reception schedule: a client arriving at time t waits until the beginning of the next broadcast cycle of T_1, at which point it can receive data from T_1 and begin playout immediately. Any other segment is received (either saved in the disk or displayed immediately) during the latest broadcast cycle prior to its playback time. To be more precise, we define the following notations.

$s(j,t)$: starting time of T_j's reception
given that the client arrives at time t.
$e(j,t)$: finish time of T_j's reception
given that the client arrives at time t.
$p(j,t)$: finish time of T_j's playback
given that the client arrives at time t.

The first time that T_1 is broadcast after time t is

$$p(0,t) = s(1,t) = \lceil t/f(1) \rceil f(1)$$

Since the client plays out the video continuously, the client starts to play out T_j at time

$$p(j-1,t) = p(0,t) + \sum_{n=1}^{j-1} f(n) \qquad (1)$$

Therefore, the client must begin to receive segment T_j at time

$$s(j,t) = \left\lfloor \left(p(0,t) + \sum_{n=1}^{j-1} f(n) \right) / f(j) \right\rfloor f(j) \qquad (2)$$

and complete its reception at time

$$\begin{aligned} & e(j,t) \\ &= s(j,t) + f(j) \\ &= \left\lfloor \left(p(0,t) + \sum_{n=1}^{j-1} f(n) \right) / f(j) \right\rfloor f(j) + f(j) \end{aligned} \qquad (3)$$

under a disk-conserving broadcast scheme with partition function $f(n)$. Note that a disk-conserving broadcasting scheme ensures that

$$s(j,t) > p(j-1,t) - f(j) \qquad (4)$$

This property of the disk-conserving broadcast scheme is the key in determining its client I/O bandwidth and client storage space requirement as shown in Section 3.2 and 3.3.

To ensure that our reception strategy is non-workahead, we consider only the partition function $f(n)$ that satisfies

$$\sum_{n=1}^{j-1} f(n) \geq f(j) - 1 \qquad (5)$$

for any $j > 1$. It is easy to verify that this is the necessary and sufficient condition for $s(j,t) \geq t$ for any j and t, i.e.,

$$\left\lfloor \left(p(0,t) + \sum_{n=1}^{j-1} f(n) \right) / f(j) \right\rfloor f(j) \geq p(0,t) \geq t$$

for any t and j.

3.1 The Number of Server Channels

Now, we analyze the number of server channels required by a disk-conserving scheme. Since a client waits at most the length of the first segment to start the playback, the service latency guaranteed by a disk-conserving broadcasting scheme with partition function $f(n)$ is the length of the first segment, i.e., $f(1)$ time units or $f(1)D/\sum_{n=1}^{K} f(n)$ seconds. In order to guarantee a service latency l, the server must dedicate K channels to the video, where K is the smallest number that satisfies $f(1)D/\sum_{n=1}^{K} f(n) \leq l$. Therefore, we have the following theorem.

Theorem 3.1 *A disk-conserving broadcasting scheme with partition function $f(n)$ requires K channels to guarantee a service latency l, where K is the smallest number that satisfies $f(1)D/\sum_{n=1}^{K} f(n) \leq l$.*

Therefore, the "growth" of the partition function completely determines the required number of server channels. We introduce the following relation between two partition functions, $f(n)$ and $g(n)$. $f(n) \succ g(n)$ iff $\sum_{n=1}^{j} f(n)/f(1) > \sum_{n=1}^{j} g(n)/g(1)$ for all j. Let f_A and f_B denote the partition function of schemes A and B respectively. We have the following corollary.

Corollary 1 *Given two disk-conserving broadcasting schemes A and B. If $f_A \succ f_B$, then Scheme A needs no more server channels than Scheme B does for guaranteeing the same service latency.*

3.2 The Client I/O Bandwidth

We now derive a sufficient condition to ensure that the client I/O bandwidth does not exceed ib. We will see that this I/O bandwidth requirement of ib results from the client's reception of at most $i-1$ channels (each of bandwidth b) and the need to play out a stored video segment (also having a bandwidth of b).

Theorem 3.2 *The disk-conserving broadcasting scheme using partition function $f(n)$ requires a client I/O bandwidth of at most ib if $f(j) \leq \sum_{m=j-i+2}^{j-1} f(m) + \gcd(f(j), f(j-i+1))$ for all $j > i$.*

The main idea of the proof is as follows. During the playback period of any segment T_n, only segments $T_n, T_{n+1}, ..., T_{n+i-1}$ might be received. In fact, segment T_{n+i} is received after the playback period of segment T_n. This is because we can bound the starting reception time of T_{n+i} by Equation (4) as follows.

$$s(n+i,t) \geq p(n+i-1,t) - f(n+i) + 1$$

Figure 2 shows the earliest starting reception time of T_{n+i}. By the same reasoning, for any j, segment T_{n+i+j} is received after the playback period of T_{n+j}, thus after the playback period of T_n. Furthermore, T_n and T_{n+i-1}'s reception periods do not overlap. If they did, they would overlap for at least $\gcd(f(n), f(n+i-1))$ time units because T_n is broadcast every $f(n)$ time units and T_{n+i-1} is broadcast every $f(n+i-1)$ time units. From Figure 2, we see that the length of their overlapping period is at most $\gcd(f(n), f(n+i-1)) - 1$ time units. Therefore, at most $i-1$ segments are received simultaneously. We prove the theorem formally as follows.

First, we identify relationships between the reception and playback times for two different segments.

Lemma 3.1 *Under any disk-conserving broadcasting scheme using partition function $f(n)$, a client arriving at time t has two properties. For any $m < n$,*

(a) *if $f(n) \leq \sum_{j=m+1}^{n-1} f(j)$, then the starting reception time of T_n follows the time at which the playback of T_m completes, i.e., $s(n,t) > p(m,t)$,*

(b) *if $f(n) \leq \sum_{j=m+1}^{n-1} f(j) + \gcd(f(n), f(m))$, then T_n's reception period does not overlap T_m's, i.e., $s(n,t) \geq e(m,t)$.*

PROOF : Suppose that $f(n) \leq \sum_{j=m}^{n-1} f(j) + x$, we have

$$\begin{aligned}
& s(n,t) \qquad (6)\\
&= \left\lfloor \left(p(0,t) + \sum_{j=1}^{n-1} f(j) \right) / f(n) \right\rfloor f(n) \\
&\qquad \text{(from Equation (2))} \\
&= \left\lfloor \left(p(0,t) + \sum_{j=1}^{m} f(j) + \sum_{j=m+1}^{n-1} f(j) \right) / f(n) \right\rfloor f(n) \\
&\geq \left\lfloor \left(p(0,t) + \sum_{j=1}^{m} f(j) + f(n) - x \right) / f(n) \right\rfloor f(n) \\
&= \left\lfloor \left(p(0,t) + \sum_{j=1}^{m} f(j) - x \right) / f(n) + 1 \right\rfloor f(n)
\end{aligned}$$

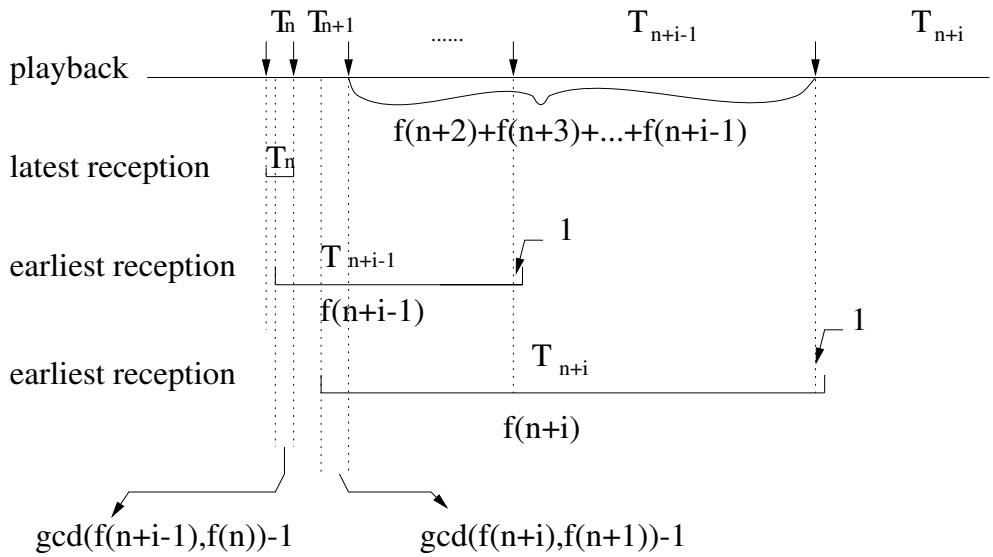


Figure 2: *The reception periods of T_{n+i-1} and T_{n+i}*

$$\geq \left\lceil \left(p(0,t) + \sum_{j=1}^{m} f(j) - x \right) / f(n) \right\rceil f(n) \quad (7)$$

$$\geq p(0,t) + \sum_{j=1}^{m} f(j) - x \quad (8)$$

$$= p(m,t) - x$$

Furthermore, the equality in inequality (7) is true only when $(p(0,t) + \sum_{j=1}^{m} f(j) - x) \bmod f(n)$ is not equal to zero. However, the equality in inequality (8) is true only when $(p(0,t) + \sum_{j=1}^{m} f(j) - x) \bmod f(n)$ is zero. Therefore, we have $s(n,t) > p(m,t) - x$.

Part (a) follows from the case that $x = 0$. Now, we prove part (b). Since $f(n) \leq \sum_{j=m+1}^{n-1} f(j) + \gcd(f(n), f(m))$, we have $s(n,t) > p(m,t) - \gcd(f(n), f(m))$. Since $p(m,t)$ is the finish playback time of T_m and $e(m,t)$ is the finish reception time of T_m, we know that $p(m,t) \geq e(m,t)$. Therefore, $s(n,t) > e(m,t) - \gcd(f(n), f(m))$. Since both $e(m,t) \bmod f(m) = 0$ and $s(n,t) \bmod f(n) = 0$ by the reception schedule, we have that both $e(m,t) \bmod \gcd(f(n), f(m)) = 0$ and $s(n,t) \bmod \gcd(f(n), f(m)) = 0$. Therefore, $s(n,t) \geq e(m,t)$. ∎

PROOF of Theorem 3.2: It is obvious that the client needs a bandwidth of at most b for retrieving a stored segment from the disk for playout at the monitor. We thus prove the theorem by arguing that at most $i - 1$ segments are saved simultaneously at any playback time of a client.

During T_1's playback period, only $T_1, T_2, T_3, ...T_i$ can be received. This is because, for any segment T_j, where $j > i$, we know $f(j) \leq \sum_{m=j-i+2}^{j-1} f(m) + \gcd(f(j), f(j-i+1)) \leq \sum_{m=j-i+1}^{j-1} f(m)$. According to Lemma 3.1(a), $s(j,t) > p(j-i,t)$. Since $j > i$, we have $p(j-i,t) > p(1,t)$. We conclude that $s(j,t) > p(1,t)$, i.e., T_j is received after T_1's finish playback time. Therefore, only $T_1, T_2, T_3, ...T_i$ can be received during T_1's playback period. Since we never save segment T_1 to the disk, at most $(i-1)$ segments are saved during this period.

Now, consider T_n's playback period for any $n > 1$. We claim that no segment other than $T_n, T_{n+1}, T_{n+2}, ..., T_{n+i-1}$ are received during this period. We prove the claim as follows. Obviously, for any $j < n$, T_j is received before T_n's playback period. For any $j > n + i - 1$, we have $f(j) \leq \sum_{m=j-i+2}^{j-1} f(m) + \gcd(f(j), f(j-i+1)) \leq \sum_{m=j-i+1}^{j-1} f(m) \leq \sum_{m=n+1}^{j-1} f(m)$. From Lemma 3.1(a), we conclude that T_j is received after T_n's finish playback time. Furthermore, we see that T_{n+i-1}'s reception period does not overlap that of T_n from Lemma 3.1(b). Therefore, at most $i - 1$ segments are received simultaneously at any time during T_n's playback period. ∎

3.3 The Client Storage Requirement

We analyze the client storage requirement of a disk-conserving broadcasting scheme as follows. For a client arriving at time t, the earliest time that the client receives T_j is $p(j-1,t) - f(j) + 1$ according to

Equation (4). Therefore, we can bound the total storage space required for segment T_j at any time. Figure 3 shows the maximum time units of data saved for each segment at the xth time unit of T_n's playback period, i.e., at time $p(n-1,t)+x$. Note that the figure shows only the case that $x > 1$, and that the client receives at most $f(n) - x$ time units of data from T_n because the client saves at most $f(n)$ time units of data and consumes x time units of data from T_n. We can derive a similar bound for the case that $x \leq 1$. Summing the storage space required for all segments, we conclude that the total amount of data saved is no greater than $f(m) - 1$ time units of data. Note that T_m is the last segment whose data is received before time $p(n-1,t)+x$. Therefore, we derive the following bound for the amount of client storage required.

Theorem 3.3
A disk-conserving broadcasting scheme with partition function $f(n)$ requires a client storage space that can store $(\max_{1\leq n\leq K} f(n) - 1)$ time units of data, i.e., $(\max_{1\leq n\leq K} f(n) - 1)Db/\sum_{n=1}^{K} f(n)$ data.

First, we state a lemma that will be useful in analyzing of the storage requirement.

Lemma 3.2 *For any $x_1, x_2, ..., x_n \geq 0$, $x_1 + \sum_{j=2}^{n}[x_j - \sum_{m=1}^{j-1} x_m]^+ \leq \max_{1\leq j\leq n} x_j$, where $[x]^+ \stackrel{\text{def}}{=} \max\{x, 0\}$.*

PROOF : We prove the lemma by induction on n. It is obviously true when $n = 1$. Now, consider two cases.
Case 1. $x_n > \sum_{m=1}^{n-1} x_m$.
We have

$$\begin{aligned} & x_1 + \sum_{j=2}^{n}[x_j - \sum_{m=1}^{j-1} x_m]^+ \\ \leq\ & x_1 + \sum_{j=2}^{n-1}[x_j - \sum_{m=1}^{j-1} x_m]^+ + (x_n - \sum_{m=1}^{n-1} x_m) \\ \leq\ & \max_{1\leq j\leq n-1} x_j + (x_n - \sum_{m=1}^{n-1} x_m) \\ \leq\ & x_n \\ \leq\ & \max_{1\leq j\leq n} x_j \end{aligned}$$

by the induction hypothesis.
Case 2. $f(n) \leq \sum_{m=1}^{n-1} f(m)$.
We have

$$\begin{aligned} & x_1 + \sum_{j=2}^{n}[x_j - \sum_{m=1}^{j-1} x_m]^+ \\ \leq\ & x_1 + \sum_{j=2}^{n-1}[x_j - \sum_{m=1}^{j-1} x_m]^+ \\ \leq\ & \max_{1\leq j\leq n-1} x_j \\ \leq\ & \max_{1\leq j\leq n} x_j \end{aligned}$$

by the induction hypothesis. ∎

PROOF of Theorem 3.3: We prove the theorem by analyzing the storage space needed during T_n's playback period. Let t denote the arrival time of the client. Consider the x th time unit during T_n's playback, i.e., time $p(n-1,t)+x$. First, we give a bound on the total data saved for segment T_n. If $x \leq 1$, the client saves at most $f(n) - 1$ time units of data from segment T_n. If $x > 1$, the client saves at most $f(n) - x$ time units of data from segment T_n, since $f(n)$ is the maximum time units of data saved and x is the consumed data. Therefore, the client saves at most $f(n) - 1 - [x-1]^+$ time units of data. Now we derive a bound on the storage space requirement of segment T_{n+i} at time $p(n-1,t)+x$ for any $1 \leq i \leq K-n$. We know that the client saves at most

$$\begin{aligned} & [p(n-1,t) + x - s(n+i,t)]^+ \\ \leq\ & [p(n-1,t) + x - (p(n+i-1,t) - f(n+i) - 1)]^+ \\ & \qquad \text{(from inequality (4))} \\ =\ & [f(n+i) - 1 - \sum_{j=n+1}^{n+i-1} f(j) - (f(n) - x)]^+ \end{aligned}$$

time units of data from segment T_{n+i} at time $p(n-1,t)+x$. In other words, the client saves at most $[f(n+1) - 1 - (f(n) - x)]^+$ time units of data from segment T_{n+1} at time $p(n-1,t)+x$. The client saves at most $[f(n+2) - 1 - f(n+1) - (f(n) - t)]^+$ time units of data from segment T_{n+2} at time $p(n-1,t)+x$. .... and so on. The client saves at most $[f(K) - 1 - \sum_{j=n+1}^{K-1} f(j) - (f(n) - t)]^+$ time units of data from segment T_K at time $p(n-1,t)+x$. Therefore, a client saves at most

$$\begin{aligned} & f(n) - 1 - [x-1]^+ \\ +\ & [f(n+1) - 1 - (f(n) - x)]^+ \\ +\ & [f(n+2) - 1 - f(n+1) - (f(n) - x)]^+ \\ +\ & ... \\ +\ & [f(K) - 1 - \sum_{j=n+1}^{K-1} f(j) - (f(n) - x)]^+ \\ \leq\ & f(n) - 1 - [x-1]^+ - (f(n) - x) \\ +\ & f(n) - x \\ +\ & [f(n+1) - 1 - (f(n) - x)]^+ \end{aligned}$$

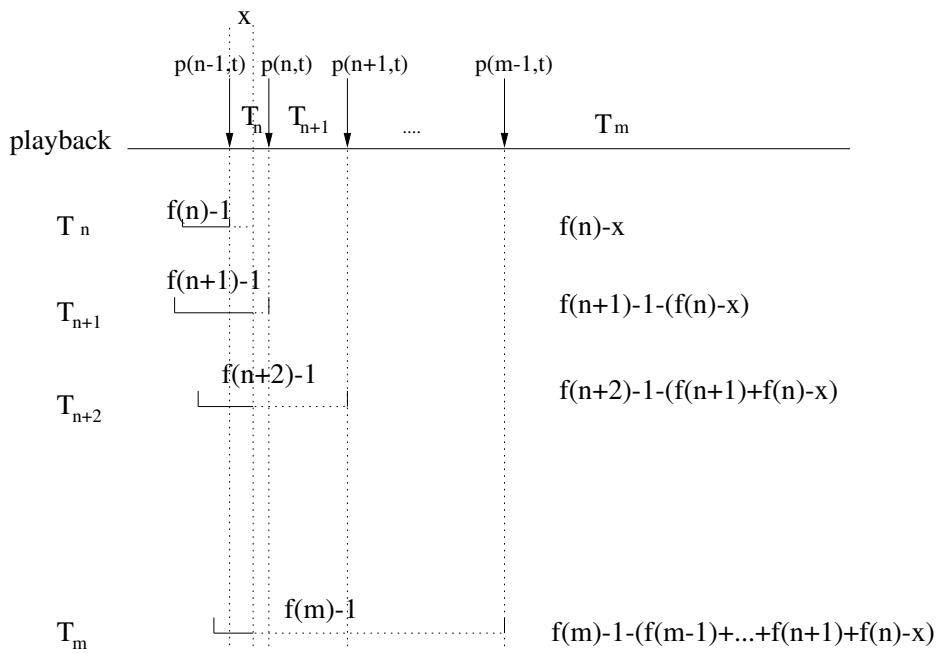


Figure 3: *The maximum time units of data saved at* $p(n-1,t)+x$

$$
\begin{aligned}
&+ \quad [f(n+2)-1-f(n+1)-(f(n)-x)]^+ \\
&+ \quad ... \\
&+ \quad [f(K)-1-\sum_{j=n+1}^{K-1} f(j)-(f(n)-x)]^+ \\
&\leq \quad f(n)-1-[x-1]^+-(f(n)-x) \\
&+ \quad \max\{\max_{n+1\leq j\leq K} f(j)-1,(f(n)-x)\} \qquad (9) \\
&\leq \quad \max_{n\leq j\leq K} f(j)-1 \\
&\leq \quad \max_{1\leq j\leq K} f(j)-1
\end{aligned}
$$

time units of data. Inequality (9) is derived from Lemma 3.2 by having $x_1 = f(n) - x \geq 0$ and $x_i = f(n+i) - 1 \geq 0$ for $i > 1$. ∎

4 Greedy Disk-Conserving Broadcasting Scheme GDB(i)

In this section we introduce a greedy disk-conserving broadcast scheme, GDB(i), for a given client I/O bandwidth of ib where i is an integer and $i > 3$. GDB(i) selects its partition function "greedily" so as to minimize the number of server channels needed to guarantee a given service latency. Corollary 1 tells us to select the partition function of GDB(i) to "grow" as fast as possible starting at $f(1) = 1$. To constrain the client bandwidth to i channels, we greedily choose $f(n)$ to be the largest number that satisfies the sufficient condition given in Theorem 3.2 and the non-workahead requirement given in Equation (5). Specifically, for $n \leq i$, $f(n)$ is the largest number that satisfies $f(n) \leq \sum_{j=1}^{n-1} f(j) + 1$. For the case that $n > i$, $f(n)$ is selected to be the maximum number that satisfies $f(n) \leq \sum_{m=n-i+2}^{n-1} f(m) + \gcd(f(n), f(n-i+1))$. Therefore, we have the following partition function, $f_{GDB(i)}(n)$ that guarantees a client I/O bandwidth of ib,

$$
f_{GDB(i)}(n) = \begin{cases} 2^{n-1}, & \text{if } n \leq i \\ \lfloor (\sum_{j=n-i+1}^{n-1} f_{GDB(i)}(j)) \\ / f_{GDB(i)}(n-i+1) \rfloor f_{GDB(i)}(n-i+1), \\ & \text{if } n > i \end{cases}
$$

The resulting partition series is

$$1, 2, 4, 8, 16, ..., 2^{i-1}, 2^i - 2, 2^{i+1} - 8, ...$$

For example, when $i = 4$, the partition series is

$$1, 2, 4, 8, 14, 24, 40, 70, ...$$

The partition function is further constrained by the disk space requirement. We see from Theorem 3.3 that the storage requirement of the GDB(i) scheme is determined by the largest segment size. Therefore, if C is the number of time units of data in the largest segment, we have the following partition function $f^C_{GDB(i)}(n)$ determined by C,

$$f^C_{GDB(i)}(n) = \begin{cases} f_{GDB(i)}(n), & \text{if } f_{GDB(i)}(n) < C \\ L, & \text{if } f_{GDB(i)}(n) \geq C \\ & \quad > f_{GDB(i)}(n-1) \\ C, & \text{if } f_{GDB(i)}(n-1) \geq C \end{cases}$$

where L is the largest number that satisfies $L \leq C$ and $L \leq \sum_{m=n-i+2}^{n-1} f_{GDB(i)}(m) + gcd(L, f_{GDB(i)}(n-i+1))$. Note that we limit $f^C_{GDB(i)}(n)$ by L for the case that $f_{GDB(i)}(n) \geq C > f_{GDB(i)}(n-1)$ in order to satisfy the sufficient condition (in Theorem 3.2) for the client bandwidth being at most ib for any C. For example, when $C = 65$, the partition series for GDB(4) is

$$1, 2, 4, 8, 14, 24, 40, 65, 65, 65, ...$$

When $C = 67$, the partition series for GDB(4) is

$$1, 2, 4, 8, 14, 24, 40, 66, 67, 67, 67, ...$$

4.1 The Client Resource Requirements

We can establish that the partition function $f^C_{GDB(i)}(n)$ satisfies the sufficient condition that a client's I/O bandwidth is at most ib. Furthermore, the sizes of the segments are monotonically increasing. Now, from Theorem 3.2 and 3.3, we have the client resource requirement of GDB(i).

Theorem 4.1 *GDB(i) requires a client disk bandwidth of ib and a client storage space of $(f^C_{GDB(i)}(K) - 1)Db / \sum_{n=1}^{K} f^C_{GDB(i)}(n)$.*

4.2 Comparing the Resource Requirements of GDB(i) and SB

We compare the resource requirement of GDB(i) with the recently proposed broadcasting scheme, SB. The SB scheme is a disk-conserving broadcasting scheme with the partition function

$$f_{SB}(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2, & \text{if } n = 2, 3 \\ 2f_{SB}(n-1) + 1, & \text{if } n > 3, n \bmod 4 = 0 \\ 2f_{SB}(n-1) + 2, & \text{if } n > 3, n \bmod 4 = 2 \\ f_{SB}(n-1), & \text{if } n > 3, n \text{ odd} \end{cases}$$

We can generalize the partition function for any client storage space.

$$f^C_{SB}(n) = \min\{f_{SB}(n), C\}$$

We show that GDB(i) requires no more resources than SB does. It can be easily verified that $f^C_{GDB(i)} \succ f^C_{SB}$. From Corollary 1 and Theorem 3.3, we conclude that GDB(i) requires no more server channels and no more storage space than SB for guaranteeing the same service latency.

Figure 4 illustrates the client storage and the server channel requirement for SB, GDB(4), GDB(5), GDB(6). We assume that the video is displayed at a rate of 1.5 Mb/s, the video length D is 100 minutes and the guaranteed service latency is 6 seconds. We select the range 100–750 MBytes for the client storage space and 10–20 for the number of server dedicated channels. From the figure, we can see that when the server dedicates 16 channels, GDB(5) needs only 110MByte storage space while SB needs 220 MBytes, which is a 50% savings on the storage space. Even GDB(4) requires only 117 MByte storage space, which is close to a 50% savings. In addition, if the client storage space is 220 MBytes, GDB(5) needs only 11 dedicated broadcast channels while SB requires 16, which is a 31% savings on the server bandwidth. Furthermore, SB requires at least 16 dedicated broadcast channels regardless of the amount of client storage, while GDB(5) needs only 11 dedicated broadcast channels when the client has only 330MByte of storage space.

The curve labeled GDB(4) in Figure 4 plots the number of channels required verus client storage space when the client I/O bandwidth is $4b$. The GDB(5) curve shows how the channel requirements are decreased when the I/O bandwidth is increased to $5b$. However, we can see that the curves for GDB(5) and GDB(6) are very close to each other. In fact, the curves for GDB(5) and GDB(6) overlap when the client storage space is greater than 330 MBytes. Therefore, we see that most of the performance gains that can be realized by increasing the client I/O bandwidth are achieved by the time the I/O bandwidth is equal to 4 or 5 times the video channel bandwidth.

5 Broadcasting Scheme GDB3

In this section, we present a broadcasting scheme (referred to as GDB3) for the case that the client I/O bandwidth is $3b$. GDB3 is a disk-conserving scheme that is different from GDB(3). Recall that GDB(3) scheme uses a linear growing partition function

$$f_{GDB(3)}(n) = \begin{cases} 2^{n-1}, & \text{if } n \leq 3 \\ f_{GDB(3)}(n-1) + 2, & \text{if } n > 3 \end{cases}$$

Since it can be easily verified that $f_{SB} \succ f_{GDB(3)}$, the disk-conserving broadcasting scheme with $f(n)$ as

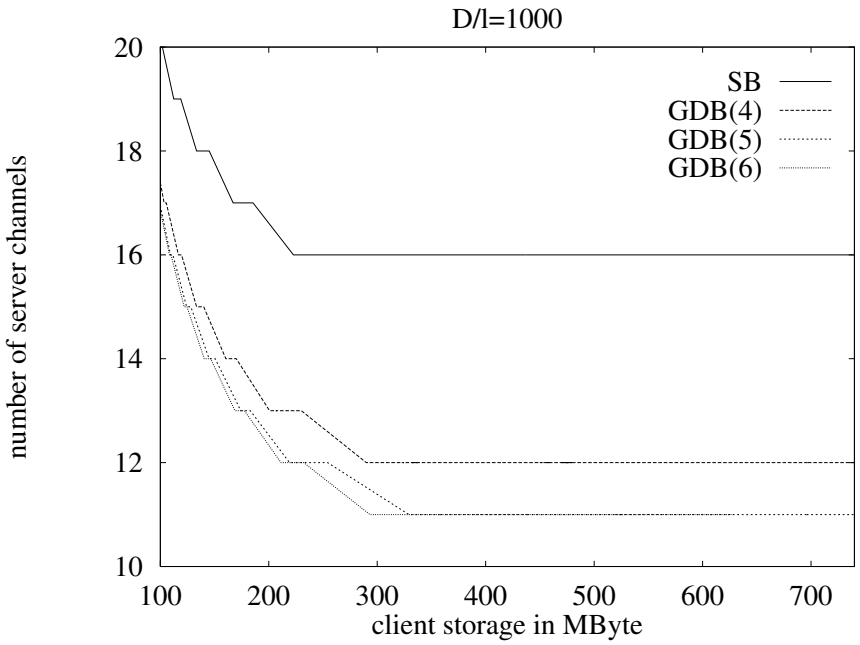


Figure 4: *The tradeoff exhibited by SB, GDB(4), GDB(5) and GDB(6)*

the partition function requires more resources than SB does. Therefore, we need to select an alternative partition function $f_{GDB3}(n)$ for the case that the client I/O bandwidth is $3b$. The intuition behind our choice of the partition function $f_{GDB3}(n)$ is to conserve the client I/O bandwidth by having two consecutively equal-length segments since the two equal-length segments can be received consecutively. Therefore, the segment size can grow exponentially without additional client I/O bandwidth. We have the following partition function for GDB3

$$f_{GDB3}(n) = \begin{cases} 2^{n-1}, & \text{if } n \leq 3 \\ 4, & \text{if } n = 4 \\ 10, & \text{if } n = 5, 6 \\ 24, & \text{if } n = 7, 8 \\ 5f_{GDB3}(n-4), & \text{if } n > 8 \end{cases}$$

The partition series generated by this function is

$$1, 2, 4, 4, 10, 10, 24, 24, 50, 50, 120, 120, ...$$

This partition function $f_{GDB3}(n)$ ensures that no more than two segments are simultaneously received. Therefore, the GDB3 scheme requires a client I/O bandwidth of $3b$.

As with the GDB(i) scheme, we can limit the size of the largest segment by parameter C, expressed in time units of data. More precisely, we have a disk-conserving broadcasting schemes that uses the partition function $f^C_{GDB3}(n)$ as follows.

$$f^C_{GDB3}(n) = \begin{cases} f_{GDB3}(n), \\ \quad \text{if } f_{GDB3}(n) < C \text{ and } n \text{ is odd} \\ L, \\ \quad \text{if } f_{GDB3}(n) \geq C > f_{GDB3}(n-2) \\ \quad \text{and } n \text{ is odd} \\ C, \\ \quad \text{if } C \leq f_{GDB3}(n-2) \text{ and } n \text{ is odd} \\ f^C_{GDB3}(n-1), \\ \quad \text{if } n \text{ is even} \end{cases}$$

where L is the largest number that satisfies $L \leq C$ and $L \leq 2f_{GDB3}(n-2) + \gcd(f_{GDB3}(n-3), L)$. Note that we limit $f^C_{GDB3}(n)$ by L for the case that $f_{GDB3}(n) \geq C > f_{GDB3}(n-2)$ in order to satisfy the sufficient condition that the client bandwidth is at most $3b$ for any C.

5.1 The Client Resource Requirement

In this section, we analyze the client disk bandwidth and storage space required by GDB3. First, we see that the partition function $f^C_{GDB3}(n)$ satisfies the condition for ensuring that the client disk bandwidth required is at most $3b$. Therefore, the theorem follows.

Theorem 5.1 *GDB3 requires a client I/O bandwidth of $3b$ and a client storage space of $(f^C_{GDB3}(K) - 1)Db / \sum_{n=1}^{K} f^C_{GDB3}(n)$.*

5.2 Comparing the Resource Requirements of GDB3 and SB

We prove that GDB3 requires no more resources than SB does. It can be easily verified that $f^C_{GDB3} \succ f^C_{SB}$. From Corollary 1 and Theorem 3.3, we thus conclude that GDB3 requires no more server channels and no more storage space than SB does for guaranteeing the same service latency.

Figure 5 compares the resource requirements of SB and GDB3. We assume that the video is displayed at a rate of 1.5 Mb/s, the video length D is 100 minutes, and the guaranteed service latency is 6 seconds. The figure shows that GDB3 decreases the client storage space requirement and the number of dedicated server channels without additional client I/O bandwidth. For example, when the server dedicates 16 channels, GDB3 requires only 160 MBytes storage space while SB requires 230 MBytes, which is 30% reduction. Furthermore, when the client has 230 MBytes of storage space, GDB3 needs to dedicate only 15 server channels while SB requires 16 server channels, which results one channel saving per video.

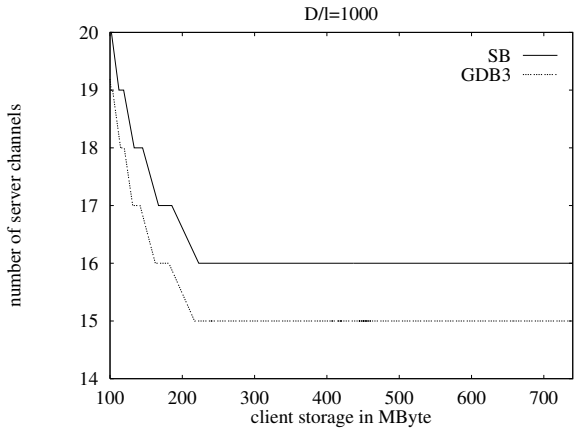


Figure 5: *The tradeoff exhibited by GDB3 and SB.*

6 Greedy Disk-Conserving Broadcasting Scheme GDB(K)

In Section 4, we saw that as the client I/O bandwidth grows, the client storage space and the number of server channels required are reduced. However, the client I/O bandwidth requirement never exceeds Kb in a disk-conserving scheme, since the client does not receive data from more than $K-1$ channels. Now, we present a disk-conserving broadcast scheme (referred to as GDB(K)) that assumes the client I/O bandwidth is at least Kb. Since the only constraint on the partition function is the non-workahead condition, the partition function for GDB(K) is

$$f_{GDB(K)}(n) = 2^{n-1}.$$

Limiting the size of the largest segment, we have a family of disk-conserving broadcasting schemes each of which depends on parameter C.

$$f^C_{GDB(K)}(n) = \max\{f_{GDB(K)}(n), C\}$$

In fact, GDB(K) is an optimal disk-conserving scheme. That is, given the client storage size, GDB(K) requires the smallest number of server channels among all disk-conserving schemes. This is because its partition function is the fastest growing partition function that satisfies the non-workahead condition and the storage size constraint. It follows from Theorem 3.2 that GDB(K) requires a client I/O bandwidth of at most $b(\lceil \log_2 C \rceil + 1)$. Therefore, GDB(K) requires a client I/O bandwidth of $b(\min\{K, \lceil \log_2 C \rceil + 1\})$, and storage space of $(f^C_{GDB(K)}(K) - 1)Db / \sum_{n=1}^{K} f^C_{GDB(K)}(n)$.

Figure 6 illustrates the client storage requirement and the number of server channels for schemes SB, GDB3, GDB(4), GDB(5), GDB(6), GDB(K). From the figure, we can see that by increasing the client I/O bandwidth by b, GDB(4) reduces both the client storage space and the number of dedicated channels significantly compared to GDB(3). For example, when the number of dedicated channels is 15, GDB(4) needs only 60% of the client storage space required by GDB(3). When the client storage space is 290 MBytes, GDB(4) needs to dedicate 3 fewer server channels than GDB(3) does. Furthermore, GDB(5)'s resource requirement is close to GDB(K)'s, which means that we can achieve close to optimal performance with a client I/O bandwidth of only $5b$.

6.1 The Minimum Number of Server Channels Required by GDB(K)

We see from the previous section that GDB(K) requires the minimum number of server channels when the client storage is only 550 MBytes. One natural question to ask is whether the minimum number of server channels required by GDB(K) is close to the optimal achievable by any broadcasting scheme. In this section, we show that the minimum number of server channels required by GDB(K) is only 1.44 times the minimum number of server channels required by *any* broadcasting scheme. We first determine the minimum server bandwidth required to achieve a given

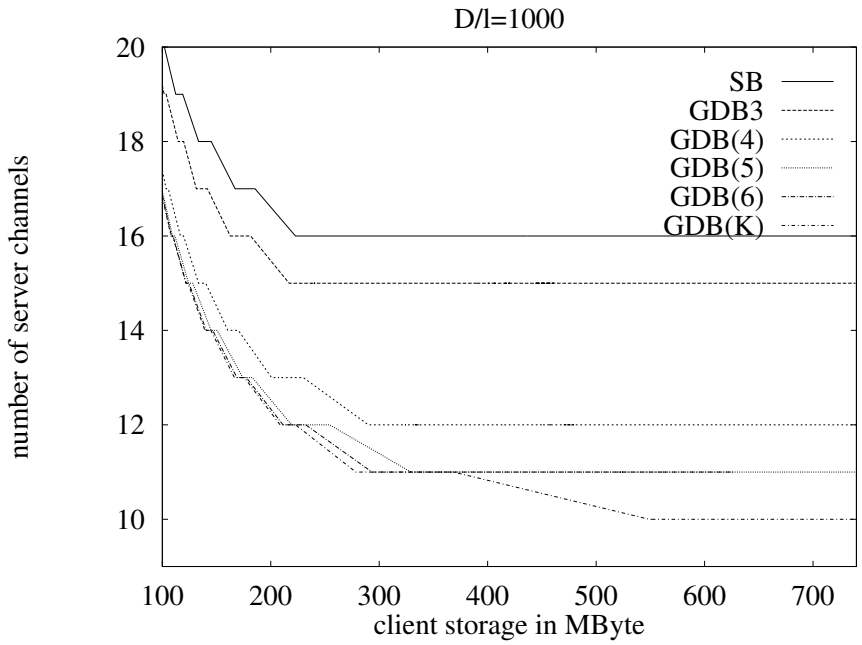


Figure 6: *The tradeoff exhibited by GDB(i) and GDB(K)*

latency for any broadcasting scheme. Using an argument on the broadcast frequency of a video data for any non-workahead broadcasting scheme, we show that the server needs to dedicate at least $\ln((D+l)/l)$ channels, for a D-second video.

Theorem 6.1 *Any broadcasting scheme needs to dedicate at least* $\ln((D+l)/l)$ *server channels in order to guarantee a service latency of* l *for a* D*-second video.*

PROOF : Consider the video data between the xth and $(x+dx)$th seconds of the video. A client arriving at time t has to receive the data between time t and $t+l+x$ in order to ensure the continuous playback, since the client has to begin to play out this data at time $t+l+x$. Therefore, the server needs to broadcast the data at least once between time t and $t+l+x$. Moreover, since a client can arrive at any time, the server needs to broadcast the data at least once between time t and $t+l+x$, for any t. We conclude that the data has to be broadcast every $(l+x)$ seconds.

To broadcast the data of length dx every $(l+x)$ seconds, the server needs a bandwidth of at least $bdx/(x+l)$. Since the server has to broadcast data between the xth and $x+dx$th second for all $x < D$, the total bandwidth required to broadcast the whole video is

$$\int_0^D bdx/(x+l) = b\ln((D+l)/l).$$

■

The minimum number of server channels required by GDB(K) is $\log_2(D/l+1)$ for achieving a service latency l. This is because in GDB(K), to achieve a service latency l, the first segment is of length at least l, the second segment is of length at least $2l$, ... Therefore, we need at least $\log_2(D/l+1)$ segments or server channels for a D-second video. From the above theorem, the minimum number of channel required by GDB(K) is only 1.44 times of the minimum number of server channels required by *any* broadcasting scheme. From the previous section, we see that GDB(5)'s resource requirement is close to GDB(K). Therefore, with a reasonable client I/O bandwidth of only $5b$ or 0.94MBytes/s, the minimum number of server channels required by GDB is close to 1.44 times the minimum server resource requirement of *any* broadcasting scheme.

7 Summary

In this paper, we have considered the problem of broadcasting popular videos from a video-on-demand server to a large number of clients across a high-speed network. We proposed a formal framework for studying broadcasting schemes and designing a family of schemes, Greedy Disk-conserving Broadcasting (GDB). GDB gives a partition strategy, broadcasting

schedule, and reception schedule according to the size of the client resources (including client I/O bandwidth and client storage space). We show analytically that GDB requires less resources than the most recently proposed scheme, *Skyscraper Broadcasting* (SB), while guaranteeing the same service latency. Furthermore, our performance study illustrates that GDB significantly reduces the resource requirements. Finally, we show that with reasonable client I/O bandwidth, both the server resource and the client storage space required by GDB is close to the minimum achievable by *any* disk-conserving broadcasting scheme.

Acknowledgments. The first author would like to thank Sneha Kumar Kasera and Zhi-Li Zhang for their helpful comments on the paper.

References

[1] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "A Permutation-Based Pyramid Broadcasting Scheme for Video-on-Demand Systems", *Proc. of the IEEE Int'l Conf. on Multimedia Systems*, June 1996.

[2] C.C. Aggarwal, J.L. Wolf, and P.S. Yu. "On Optimal Batching Policies for Video-on-Demand Storage Server", *Proc. of the IEEE Int'l Conf. on Multimedia Systems*, June 1996.

[3] P.W. Agnew, and A.S. Kellerman, *Distributed Multimedia*, Addison Wesley, ACM Press.

[4] K.C. Almeroth, and M.H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service", *IEEE Journal on Selected Areas in Communications*, 14(6):1110-1122, August 1996.

[5] D. Comer, *Internetworking with TCP/IP*, Prentice Hall, 1995, pages 289-300.

[6] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching", *Proc. of ACM Multimedia,* Oct. 1994, pages 15-23.

[7] A. Dan, D. Sitaram, and P. Shahabuddin. "Dynamic Batching Policies for an On-Demand Video Server", *Multimedia Systems,* 4(3):112-121, Jun. 1996.

[8] A. Dan, P. Shahabuddin, Dinkar Sitaram, and D. Towsley, "Channel allocation under batching and VCR control in movie-on-demand servers", *IBM Research Report*, 1994.

[9] S. Deering, RFC 1112.

[10] K.A. Hua, and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems", *ACM SIGCOMM*. Sept. 1997.

[11] S. Viswanathan, and T. Imielinski, "Metropolitan area video-on-demand service using Pyramid Broadcasting", *IEEE Multimedia Systems*. 4:197-208, 1996.