Performance Management in ATM Networks

Simon Andrew Crosby

St. John's College
University of Cambridge

A dissertation submitted for the degree of
Doctor of Philosophy

May 1995

Abstract

The Asynchronous Transfer Mode (ATM) has been identified as the technology of choice amongst high speed communication networks for its potential to integrate services with disparate resource needs and timing constraints. Before it can successfully deliver integrated services, however, significant problems remain to be solved. They centre around two major issues. First, there is a need for a simple, powerful network service interface capable of meeting the communications needs of new applications. Second, within the network there is a need to dynamically control a mix of diverse traffic types to ensure that they meet their performance criteria.

Addressing the first concern, this dissertation argues that a simple network control interface offers significant advantages over the traditional, heavyweight approach of the telecommunications industry. A network control architecture based on a distributed systems approach is presented which locates both the network control functions and its services outside the network. The network service interface uses the Remote Procedure Call (RPC) paradigm and enables more complicated service offerings to be built from the basic primitives. A formal specification and verification of the user-network signalling protocol is presented. Implementations of the architecture, both on UNIX and the Wanda micro-kernel, used on the Fairisle ATM switch, are described. The implementations demonstrate the feasibility of the architecture, and feature a high degree of experimental flexibility. This is exploited in the balance of the dissertation, which presents the results of a practical study of network performance under a range of dynamic control mechanisms.

Addressing the second concern, results are presented from a study of the cell delay variation suffered by ATM connections when multiplexed with real ATM traffic in an uncontrolled network, and from an investigation of the expansion of bursts of ATM traffic as a result of multiplexing. The results are compared with those of analytical models. Finally, results from a study of the performance delivered to delay sensitive traffic by priority and rate based cell scheduling algorithms, and the loss experienced by different types of traffic under several buffer allocation strategies are presented.

To Bev

Preface

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

This dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university.

No part of this dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

This dissertation does not exceed sixty thousand words, including tables, footnotes and bibliography.

Publications

Some of the work described in this dissertation has been published, and is described in references [Crosby93, Black94a, Crosby94] and [Barnard95].

Trademarks

ARM is a trademark of Advanced RISC Machines.
AVA-200 is a trademark of Fore Systems Inc (USA) and KNET Ltd (UK).
Ethernet is a trademark of the Xerox Corporation.
Ultrix is a trademark of Digital Equipment Corporation.
MIPS is a trademark of MIPS Technologies Inc.
UNIX is a trademark of AT&T.

Acknowledgements

I would first like to thank my supervisor, Ian Leslie, for his inspiration, encouragement and support throughout this work. I am deeply indebted also to Roger Needham and Derek McAuley for many stimulating discussions and much practical advice, and to all of the members of the Systems Research Group at the University of Cambridge, from whom I have learned so much, in particular Paul Barham, Richard Black, Shaw Chuang, Joe Dixon, Mark Hayter, Ian Pratt, Eoin Hyden, Simon Moore, Timothy Roscoe, David Evers, Robin Fairbairns, Subir Biswas, Burkhard Stiller, Daniel Gordon and Kobus van der Merwe.

Frank Kelly, Richard Gibbens and Nigel Bean of the Statistical Laboratory, and Peter Key of British Telecom have frequently given me the benefit of their insight into the performance of complex queueing systems, for which I am extremely grateful.

I owe thanks also to Dieter Barnard of Siemens Corporate Research, Münich who introduced me to the joys of TLT and proved the correctness properties of the UMI protocol, described in Appendix A.

I would like to acknowledge, with thanks, the useful comments on initial drafts of this dissertation made by Simon Moore, Steve Pope, Shaw Chuang, David Evers, Ian Leslie, Burhkard Stiller, Derek McAuley, Kobus van der Merwe and Beverley Keith.

Much of the work in this dissertation has been carried out since I became severely affected by Repetitive Strain Injury. Without the support of the Computer Laboratory, in particular Ian Leslie, Derek McAuley and Roger Needham, I would have been unable to complete my research. The Laboratory purchased a speech recognition system for my use which enabled me to complete my work and dictate this thesis. Nigel Bean, Diana Lucic, Subir Biswas, Noha Adly, NarmHee Lee, Richard Hayton and Ralph Becket made a special effort to keep my spirits up and to keep me at work during the worst months of my RSI, for which I am extremely grateful.

I could never have undertaken this work without the love and support of Beverley Keith, and my family, to whom I will remain eternally grateful.

Finally, I am indebted to the Bradlow Foundation (South Africa), the Foundation for Research Development (South Africa), and the Computer Laboratory for their financial support.

Contents

List of Figures

List of Tables

Glossary

AAL	ATM adaptation layer
ABR	Available Bit Rate
AMO	At Most Once (semantics)
ANS	Alarm Notification Service
ANSA	Advanced Networked Systems Architecture
API	Application Programming Interface
ARM	Advanced RISC Machine
ATM	Asynchronous Transfer Mode
ATDM	Asynchronous Time Division Multiplexing
AVA	Fore Systems AVA-200 Audio Video Adapter
BGT	Background Traffic (in experimental work)
B-ISDN	Broadband Integrated Services Digital Network
BSD	Berkeley Software Distribution
CAC	Connection Admission Control
CBN	Cambridge Backbone Network
CDF	Cumulative Distribution Function
CDV	Cell Delay Variation
CFR	Cambridge Fast Ring
CLP	Cell Loss Priority
CLR	Cell Loss Ratio (or Cell Loss Rate)
CLS	Cell Level Scheduler
CPU	Central Processing Unit
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
DAN	Desk Area Network

DCE	Distributed Computing Environment
DCT	Discrete Cosine Transform
EDD	Earliest Due Date
EXO	EXactly Once (semantics)
FCFS	First Come First Served
FDL	Fairisle Data Link
FIQ	Fast Interrupt Request (on ARM processors)
FPC3	Fairisle Port Controller Revision 3
FPGA	Field Programmable Gate Array
GSP	Generic Signalling Protocol
HEC	Header Error Check (an ATM cell header function)
HP	High Priority
I/O	Input and Output
IDT	Inter-cell Departure Time (distribution)
IB	In Band
IN	Intelligent Network
IP	Internet Protocol
IPC	Inter-process communication
IRQ	Interrupt ReQuest (on ARM processors)
ISDN	Integrated Services Digital Network
ITU-T	International Telecommunications Union, Telecommunications sector
JPEG	Joint Photographic Experts Group (still image compression standard)
MD	Management Domain
MIDDL	Mothy's Interface Definition Language
MMI	Master Master Interface
MSDR	Multi-Service Data Representation
MSNA	Multi-Service Network Architecture
MSNL	Multi-Service Network Layer
MSRPC	Multi-Service Remote Procedure Call
MSSAR	Multi-Service Segmentation and Reassembly
MSN-CMA	Multi-Service Network Connection Management Architecture

OB	Out of Band
OSF	Open Software Foundation
PBS	Partial Buffer Sharing
PCB	Protocol Control Block
PDU	Protocol Data Unit
PSI	Platform Specific Interface
QoS	Quality of Service
RARP	Reverse Address Resolution Protocol
ROM	Read-Only Memory
RB	Rate Based (scheduler)
PTM	Packet Transfer Mode
REF	Reference Connection (in experimental work)
RPC	Remote Procedure Call
SP	Static Priority (scheduler)
SAP	Service Access Point
SCI	Slave Control Interface
SS7	Signalling System Number 7
STM	Synchronous Transfer Mode
TDS	Topology Discovery Service
TINA	Telecommunications Information Networking Architecture
TCP	Transmission Control Protocol
TMN	Telecommunications Management Network
UMI	User Master Interface
VCI	Virtual Circuit Identifier (also Virtual Channel Identifier)

Chapter 1

Introduction

The successful integration of advanced telecommunications services onto a single high speed network hinges on the development of effective network control mechanisms. This dissertation investigates control policies suitable for a multi-service communications environment based around an Asynchronous Transfer Mode (ATM) network, and presents results from experimental implementations.

1.1 Motivation

Modern economies are increasingly dependent on rapid and easy access to information. The decreasing cost of the basic communication technologies, together with the wide availability of cheap processing power, has led to an unprecedented demand for low cost, high bandwidth communication services as the computer, telecommunications and entertainment industries converge.

In response, a large number of new communication technologies has become available, both for the local and wide area. Of these, ATM alone has succeeded in capturing the attention of all sectors of the communications industry because it promises to deliver integrated, universal interconnection, blurring the distinction between local and wide area networks. ATM is unique also in that it appears to offer the best prospects for solving a particularly awkward problem in an integrated communications environment, namely the accommodation of multiple services with widely differing resource and performance demands.

New multimedia applications impose strict constraints on the acceptable level of performance which the network must deliver. These constraints specify their tolerance of loss, and timeliness requirements for the transmission and delivery of data. Multimedia applications are also characterized by resource requirements which may vary over time. Existing telecommunications technologies can either meet the performance constraints, but with an unacceptably low resource utilization, or else make efficient use of the network but fail to deliver the required performance. ATM, in contrast, offers fine-grained sharing of bandwidth, allowing both high utilisation and low access latency.

Before ATM can fulfil its promises, however, significant problems remain to be solved. They centre around two major issues. First, there is a need for a simple, powerful, and flexible network service interface capable of meeting the communications needs of new applications. The approach of the telecommunications industry to date has been to adopt inefficient and inflexible interfaces, in the form of signalling protocols developed from those of earlier telecommunications networks. This approach is unable to keep up with the pace of technological development and stifles the introduction of new services. The selected protocols are also unsuitable for local area and home entertainment networks in which low cost ATM end-systems will be deployed. Second, within the network there is a need to dynamically control the mix of diverse traffic types, ensuring that both they and the network meet their performance criteria. Naïve implementations of new services on a network with inappropriate control mechanisms can lead to poor performance, as shown by early experience with voice and video on the Internet.

Developments from two research areas suggest that both of these problems can be solved. In the first instance, the growing availability of cheap processors has led to the development of sophisticated distributed systems architectures. These simplify the design and implementation of distributed solutions to computing problems, using high level language constructs with well defined communication semantics. Application of the methods of distributed systems to the control and management of telecommunication networks may lead to the solution of the first problem facing ATM.

Addressing the second problem is a vast literature of performance studies of ATM systems which has, largely on the basis of theoretical analysis, proposed a variety of promising dynamic control algorithms for ATM networks. Although this suggests that appropriate solutions may soon be found, the algorithms proposed thus far must be regarded with a degree of caution: almost without exception their analysis has been based on the use of traffic models which at best approximate the behaviour of real applications. Urgent debate on their effectiveness is required in the light of measured performance results using real ATM traffic. Only then will it be possible to select the most appropriate solutions.

1.2 Aims

This dissertation investigates dynamic control mechanisms for ATM networks, which operate on two time scales, and which have very different roles: the first type operates on a relatively long time scale and concerns the establishment and maintenance of connections. The second operates on a short time scale and concerns the dynamic allocation of network resources to ATM traffic. Combined, these functions are responsible for ensuring that the network and its users meet their performance criteria, and are called the *performance management* functions.

An architecture for performance management in a high speed ATM network is presented which applies a distributed systems approach to the control problem. The design aims to demonstrate the advantages of removing complex control functions from the network, locating them instead in a set of distributed programs which may be located outside the network. The architecture also offers a high degree of experimental flexibility. This is exploited to investigate the performance of an ATM network under real traffic conditions, with the aim of determining factors which critically affect the performance delivered to network users under several dynamic control regimes.

1.3 Background

Our current communications environment has developed around multiple technologies: separate voice, data and entertainment networks have each been specialized for their task. The obvious drawbacks, namely the duplication of physical networks, and their inability to accommodate different modes of communication led to the search for a technology which could integrate multiple services.

1.3.1 The Asynchronous Transfer Mode

The Asynchronous Time Division Multiplexing (ATDM) technique, whose early developers included AT&T Bell Laboratories [Fraser93], Cambridge [Hopper78] and CNET [Devault88], has become accepted as the best solution to the problem of integrating communications services. ATDM, now simply called ATM, borrows the best properties from each of two existing multiplexing mechanisms.

From the Synchronous Transfer Mode (STM), ATM borrowed the notion of a fixed size unit of access. In STM small, fixed size portions of each digital signal (circuit) are interleaved in time. The data rate of each source is constant, and the bearer is framed. Transmission is synchronous and the receiver can identify and demultiplex any signal given its position within the frame. This is ideal for Constant Bit Rate (CBR) services, which make full use of the fixed allocation. Synchronous transmission ensures that traffic experiences virtually no jitter[1]. In ATM the unit of access, the cell, was made a fixed size to bound access delay. The standardised size of 53 bytes, of which 5 contain control information, is a compromise between the desire for low access latency and maximum efficiency.

[1]Loosely defined here as the variance in end-to-end delay between successive data units.

From the Packet Transfer Mode (PTM), ATM has borrowed the notion of packets and asynchronous access. The packets are cells, and may be transmitted whenever the source has data to send. One advantage of asynchronous access is that multiple Variable Bit Rate (VBR) sources can be statistically multiplexed onto the transmission link: silence on the part of one source permits another to gain access; this enables a greater number of sources to be multiplexed on the same link than using STM. Combined with the fixed cell size, this permits the network to precisely schedule its resource usage amongst contending sources.

ATM can meet the needs of modern communications services, but, as mentioned previously, its development has brought to light two pressing problems: first, the network will need to support a large number of applications characterised by different connectivity requirements, diverse types and volumes of data, and performance constraints. A rich network interface is required to enable applications to specify their requirements and control the communication. Second, the use of statistical multiplexing permits the over-booking of network resources. Traffic must be buffered at switches and multiplexing points when there is access contention. This makes the provision of performance guarantees more difficult. Delays and jitter can be bounded by limiting the available buffer space but this introduces the possibility of loss when an arriving cell finds the buffer full.

1.3.2 The Network Control Interface

In any network where performance guarantees of some type are required, resources must be "reserved" before communication commences. In traditional telephony and the narrowband Integrated Services Digital Network (ISDN), a circuit represents a static allocation of bandwidth and identifies the route followed by data through the network. In ATM, as in other connection oriented packet networks such as X.25 [Stallings89], circuits are virtual: the network allocates the user a Virtual Circuit Identifier (VCI) which is used to label every cell. The network establishes a mapping between the VCI and state which is used to route cells through the network and allocate them transmission capacity. This contrasts with non-connection oriented networks, such as the IP-based Internet [Comer91], in which no resource reservation is required before transmission. More recently, however, the need to support applications such as audio and video conferencing on the Internet has led to the development of "connection-like" reservation protocols such as ST-II [RFC90] and RSVP [Zhang93b].

1.3.2.1 Signalling

In connection oriented networks the mechanism used to set up a connection is termed signalling, and involves the exchange of messages both between user and network, and within the network to allocate resources at each node on the path. Message-based signalling can be implemented in two ways: in the first, signalling messages are interleaved "in band" with the data. Examples include Dual Tone Multi-Frequency dialling in telephone networks and control packets in X.25. An alternative approach, favoured for its efficiency and reliability, is to exchange signalling on a dedicated "out of band" (virtual) circuit. This increases the data rate at which signals are transmitted and means that signalling messages are not delayed by user data.

The best known out of band signalling protocol is the common channel Signalling System Number 7 (SS7) [Jabbari91] standardised by the International Telecommunications Union Telecommunications Sector (ITU-T) for use in telephony and ISDN. SS7 is a message-based signalling system used within a telecommunications trunk network to transfer signalling messages both for basic connection control and for sophisticated Intelligent Network (IN) services, such as multi-party calls, advanced billing, closed user groups and call transfer. A common, reliable, datagram-based message passing service is used by different User Parts which implement network control functions, including telephony and ISDN signalling, and support access to network services and databases. Separately specified user access protocols permit network users to request communications services from the SS7 signalling system.

1.3.2.2 Services, Applications and Signalling

SS7 was conceived as the best way to provide intelligent services to relatively unintelligent user terminals[2]. The architecture was first released in 1980, and has been progressively developed; the most recent release in 1988 consisted of 38 separate recommendations. Since its adoption, technological advances have led to increases in processing power and memory capacity, physical media capacity (bit rates), and improved media performance (decreasing error rates). Both user terminals and network nodes may now be sophisticated computers, capable of communicating reliably at high data rates. Software technology has also advanced significantly. Since SS7 was first released the familiar OSI 7 layer model of communications [Stallings89] has been developed, encouraging the specification of layered services and protocols. Structured programming, high level languages and distributed systems techniques have also become widely used.

The parallel development of SS7 and the OSI model have followed somewhat divergent paths: although SS7 has a layered structure similar in some ways to the OSI model, it is wholly aimed at telecommunications signalling, whereas the OSI model was intended to facilitate "open" data communications between computing applications. Only recently have the SS7 developers begun to realise

[2]For example, a telephone or videotex terminal.

that the sophisticated services of the IN correspond to applications in the OSI model [ITU-T93c]. Instead of being "open", however, SS7 (like other ITU-T protocols) and the applications it supports, are bound within the trunk network on specialised equipment, inaccessible from the outside: a specific goal of the ISDN approach is to support access to a wide range of services using a limited set of standardised procedures. This has the consequence that all telecommunications services must be located within the service provider network and that new signalling protocol elements must be introduced into the user equipment for every new service.

1.3.2.3 ATM and Broadband ISDN (B-ISDN)

The specifications of many current ITU-T signalling protocols are flawed because they fail to distinguish clearly between the application processes they serve and their fundamental communications requirements. This leads to severe difficulties when attempting to extend or evolve the protocols to new applications or technologies, such as ATM. The favoured approach of the ITU-T for the development of broadband services is to add a B-ISDN User Part (B-ISUP) to SS7 [ITU-T93a], and to modify the ISDN user-network protocol, Q.931, to support B-ISDN access. This approach has already run into difficulties: Q.931 is unable to support virtual circuits, resource reservation, multicast or multi-connection calls, and consequently a new protocol, Q.2931 [ITU-T94a], is being developed. More seriously, this indicates that the telecommunications industry has yet to realise that service integration in broadband (specifically, ATM) networks, occurs at the ATM layer, and not at a higher "service layer". The B-ISDN as a service concept is thus unlikely to materialise because:

- It is difficult to anticipate all future services and their needs, hence any signalling protocol which tries to do so will need to be frequently modified.
- ATM is inherently "open": once users have access to an ATM bearer they may well implement their own, sophisticated services on top of it, irrespective of the wishes of the network operator.
- Deregulation of the telecommunications market will force network operators to provide access to third party service providers from outside the network.

Although the ITU-T has moved to integrate its signalling work for the IN with the OSI model, the current approach is still based on message passing and has significant drawbacks: each protocol is required to indicate its backward compatibility with older protocol versions, and must be able to deal intelligently with future, unknown message types and parameters. The application (service) interfaces themselves are not typed: so-called "hooks" to IN service logic are embedded within the basic call signalling process in the form of signalling protocol information elements which can be directed to service access points. The encoding of these protocol elements is service-specific, and even though an abstract syntax and encoding scheme such as ASN.1 [Rose91] may be used, the syntax of the elements used by an application must be identified on a per-interaction basis.

The problem here is that signalling for all services is multiplexed onto a single access protocol which must be continually enhanced as applications develop. Tennenhouse [Tennenhouse89] identified the need to avoid multiplexing in the data path because it is inefficient and leads to poor performance. This dissertation argues that multiplexing multiple signalling tasks (for each service) onto a single control protocol complicates it unnecessarily, leaving the protocol vulnerable to failure and hampering the development of new services. It is important that the development of broadband communications does not follow this route.

1.3.2.4 The Distributed Systems Approach

Many of the problems which the ITU-T standards have attempted to address have already been solved by advanced distributed systems architectures, such as ANSA [ANSA89, ANSA92] and others. In such architectures service interfaces, the procedures they export, and the data types they use are strongly typed. Each service instance exports a unique interface reference representing its service offer, and a federated "trading" environment allows prospective clients to obtain service offers which match their requirements. The process of binding to a service offer ensures that both client and server conform to the same interface type, and the use of remote procedure calls (RPC) simplifies communication. Importantly, the binding represents a dedicated communication path between client and server which can be used to control communication, making this approach attractive for signalling between network users and services in the communications environment.

1.3.2.5 Support for Multimedia Applications

The use of RPC is ideal for control purposes but poorly suited to continuous media communications. New services incorporating multiple media must be integrated into the distributed systems environment. This problem has been addressed by several authors. Nicolaou [Nicolaou91], for example, defined an architecture named IMAC for real-time multimedia communication, integrating control and synchronisation support for typed continuous media streams and their performance constraints into the ANSA framework. Sreenan [Sreenan92] developed a synchronization service for continuous media applications, also using ANSA. Two important needs can be identified: first, distributed multimedia communications services, such as a synchronisation service, may act as *third party* agents on behalf of network clients, and need to be able to manage communication components on their behalf [Heijenk94]. Second, the control of remote multimedia devices, such as cameras and video stores, requires access to traditionally hidden network control interfaces. There is no obvious way to integrate such systems into traditional network control architectures based on the ISDN approach.

Support for multimedia applications has been addressed by extending the network signalling interface [Bubenick91, Minzer91, Heijenk94], providing sophisticated call and connection management signalling services. These approaches go some way towards addressing the problems of network control, but result in extremely complicated interfaces which are very difficult to specify and verify. Their support for third party service provision is also awkward.

1.3.2.6 Contribution

The distinction between network service and user is becoming blurred; to support applications using architectures such as IMAC, the network interface should be open in two ways: first, applications should have access to an open signalling interface which allows them to construct sophisticated multimedia calls, and second, the network should be able to expose its internal control interfaces to privileged applications which implement new services.

This dissertation proposes a novel approach to the control of communication networks, using distributed systems techniques. Chapter 3 presents an architecture for distributed control of an ATM network which implements both network control functions and services as a set of distributed applications which may be located outside the network. This

- simplifies the network interface, reducing the possibility of errors and enhancing its efficiency and flexibility,
- encourages development of sophisticated services,
- permits the compositional specification and verification of network control and service interfaces,
- facilitates the incorporation of both intelligent and dumb network elements,
- reduces the cost of integrating new network equipment, maximising re-use of software components, and
- permits network control software to be developed in a powerful environment, with high level support for testing and debugging.

Aspects of network control may be integrated into sophisticated multimedia service architectures, such as IMAC, giving rise to a truly integrated network capable of meeting the demands of future applications. Crucial to the success of the architecture is the correct operation of its distributed control protocols. Chapter 4 describes the formal specification and verification of the signalling protocol used, and contrasts the architecture of Chapter 3 with related work, described in the literature. An implementation of the architecture is presented in Chapter 5, demonstrating the viability of the approach.

1.3.3 Traffic Control in ATM Networks

By integrating all services onto a single bearer, ATM places new demands on the network control system. It is unique among statistical multiplexing technologies in allowing users to associate Quality of Service (QoS) constraints with traffic streams. To enable the network to guarantee these constraints, new *in band* control mechanisms are required to schedule the network resources between contending users.

1.3.3.1 Quality of Service Issues

The QoS constraints for each class of applications will differ – data applications are typically tolerant of delay but intolerant of loss, whilst audio and video have strict requirements on the timeliness of delivery, but may tolerate some loss. It is crucial that the QoS specification describes performance criteria which the network can *deliver* and which are *measurable*. In this sense it forms a service *contract* between user and network: the network agrees to deliver the QoS subject to the user adhering to its declared traffic parameters. Because of the dynamic and unpredictable nature of ATM traffic, however, the guarantee made by the network must ultimately be probabilistic; a loss constraint might stipulate that fewer than 1 in 10^6 cells be lost, for example. Both the QoS and traffic parameters are negotiated when the connection is set up and possibly also during the call.

1.3.3.2 Traffic Parameters

The traffic parameters are used by the network for resource reservation and traffic control. A fundamental requirement is that they capture the dynamic aspects of ATM traffic: computer communication is bursty, and calls may last seconds or days. Data and compressed video traffic have been shown to exhibit signs of long range dependence [Leland93, Garrett94], and both client-server (distributed) and distributive communication can give rise to highly correlated traffic.

Section 1.3.2 argued that in an open multi-service network it will be impossible to characterise all services because they, too, are network users. The traffic parameters are thus of key importance. They must provide the network with a concise statement of the information it requires to *allocate resources*, including bandwidth and buffer space, and enable it to dynamically *manage traffic* from the source. They must also be useful for *admission control*, permitting the network to protect itself from congestion, and they should not be excessively complex or difficult to determine. Finally, they should permit *resource accounting*, and should be *flexible* enough to accommodate future services.

This is a tall order, and consequently a large amount of research has been carried out to identify parameters which critically affect network performance, largely using analytical models. Two major problems are associated with this approach: first, to accurately reflect the behaviour of real ATM sources the models require a large number of parameters which must be fitted to measured data – this is difficult or impossible in practice, and second, the large number of parameters makes solution of the performance models difficult. In spite of this, much of the current understanding of ATM systems has resulted from these models, and they have been used to guide the ITU-T standardisation process. There is an urgent need for results from practical investigations of performance issues, both to improve the accuracy of analytical models and to guide the development of improved dynamic admission control policies.

1.3.3.3 Admission Control

Traffic control in ATM networks essentially involves admission control mechanisms operating on multiple time scales. Hui [Hui88], for example, identified the need for controls operating at the call, burst and cell scales, including:

- Connection Admission Control (CAC), and parameter negotiation (call time scale). Given a particular traffic mix, the CAC function must determine whether accepting a new connection will cause the QoS constraints of any of the connections to be violated, either on the basis of pre-computed [Kelly91] or measured data [Courcoubetis91, Crosby95], or both [Gibbens95].

- Flow and congestion controls, (burst and cell scales). The network limits resource use to its available values by providing feedback to sources (flow control) [Kung94, Ozveren94], or alternatively discarding traffic when resources are scarce (congestion control) [Lucantoni90, de Cleene94].

- Renegotiation mechanisms (burst scale). Sources negotiate with the network on a per-burst basis for transmission capacity [Boyer90, Crosby93].

- Policing, Buffer allocation policies and cell scheduling (cell scale). These are discussed in more detail in Section 1.3.3.4.

A detailed study of mechanisms on each time scale is beyond the scope of this dissertation. Instead, it aims to provide new insight into the effects of multiplexing in a real ATM network on the performance delivered to ATM connections under a variety of cell scale control regimes.

1.3.3.4 Cell Scale Admission Control

During periods of short-term overload cells must be buffered, causing delay. For some classes of application both the end-to-end delay and the jitter which result may affect their performance, requiring large playout buffers at the receiver. For video conferencing, for example, a maximum delay of approximately 20ms (half a frame time) is frequently cited [Frater94]. Furthermore, queueing delays can cause cells from a connection to become grouped together, artificially increasing the peak rate and making the traffic more bursty, violating its traffic parameters, and leading to buffer overflow. The task of determining whether the user has deliberately exceeded its traffic parameters and of protecting the network from abuse is called policing.

Policing

The policing function must decide whether or not to admit an arriving cell, based on observation of the traffic process and the agreed resource contract. Key to the success of a policing algorithm is its transparency: a compliant traffic stream should experience no loss and only traffic in excess of the stated parameters should be discarded. It is widely recognised, however, that it is difficult or impossible to accurately police moments of the source distribution other than the peak, based on a short observation period [Tutufor90]. The ITU-T has selected a simplified leaky bucket algorithm [ITU-T93b] for policing the peak rate of a connection. Under this algorithm arriving cells are placed in a (virtual) buffer of size one, which is drained at the peak rate. Cells arriving in excess of this rate are either dropped or marked as discard candidates using the Cell Loss Priority (CLP) bit in the cell header.

In spite of extensive analytical studies of the effectiveness of policing algorithms, for example [Rathgeb90, Roberts92], it is still not known how significant the effects of multiplexing in ATM networks are likely to be, what sorts of controls should be applied by the network, and where they should be located. Several questions must be answered: first, what degree of loss will a well behaved connection suffer as a result of multiplexing as its traffic traverses the network? This is addressed in Chapter 6, which studies the performance delivered to CBR ATM connections when multiplexed with bursty background traffic at multiple switches. Second, if sources are not policed at the network edges, what are the effects of multiplexing on bursts of traffic as they proceed through the network? Chapter 7 investigates a phenomenon known as burst expansion, which suggests that bursty traffic should become better behaved as a result of multiplexing.

Scheduling, Shaping and Buffer Allocation

When temporary overload occurs, traffic must be buffered at switch nodes, requiring the solution of two problems. First, how big should the buffers be? Queueing models can help to identify a size which guarantees a low probability of buffer overflow, but may not realistically reflect the queueing dynamics of real ATM traffic. Recent approaches based on dynamic measurement [Duffield95, Crosby95] are promising in this regard. Second, how should the buffer space be multiplexed between different connections? Static allocation is inefficient, but dynamic sharing risks domination by one class. When an arriving cell finds the buffer full, either it or a cell in the buffer must be discarded. To solve these problems a buffer allocation mechanism must be implemented which attempts to meet the loss and delay constraints of the different connections. Delay considerations suggest that a small buffer is more appropriate, however if the buffer is too small or the traffic too bursty then excessive loss will result.

Irrespective of the buffer allocation policy, queued traffic must be scheduled for transmission. Scheduling algorithms may attempt to accommodate multiple delay priorities, or deliver guaranteed rates of transmission to different connections. In addition, scheduling can be used to shape bursty traffic at network entry points, to reduce its impact on other sources. Recent work in this area has identified a number of promising policies, but, once again, these have been proposed on the basis of highly restrictive analytical models.

Chapter 8 studies several cell scale policies in detail, on the basis of experimental implementation on the Fairisle network. The investigation reveals several interesting aspects of network control and traffic dynamics which are seldom incorporated into analytical models, and finds that several algorithms proposed in the literature on the basis of analytical work fail to perform adequately in a real ATM network. The results of Chapters 6, 7 and 8 are based on experimental observation of multiplexed traffic in the Fairisle ATM network at Cambridge.

1.4 Overview

This chapter has provided a brief overview of several problems which must be solved in the ATM environment, and outlined the contributions made by this dissertation. The work is structured as follows:

Chapter 2 introduces the experimental environment in which the research described was undertaken, and describes the multi-service network architecture on which it was based. Chapter 3 describes an architecture which facilitates distributed control and signalling in a multi-service network, and Chapter 4 presents details of the specification and verification of the signalling protocol used, stressing the need for its robustness in the face of errors. Implementation of the architecture both on UNIX and the Fairisle ATM network is described in Chapter 5.

Use of the architecture is described in the balance of the dissertation: Chapters 6 and 7 investigate the problem of cell delay variation and burst expansion in an uncontrolled ATM network and Chapter 8 investigates the performance of the network under a variety of cell-scale control regimes.

Chapter 2

Research Context

This chapter describes the context in which the research described in later chapters was undertaken. Owing to the practical nature of much of the work, its description involves technical details of the experimental equipment used.

2.1 The Fairisle ATM Network

Building on earlier ATM experience at Cambridge, including the Cambridge Fast Ring (CFR) [Hopper86], the Cambridge Backbone Network (CBN) [Greaves88] and the Cambridge Fast Packet Switch [Newman89], the Fairisle Project developed an experimental ATM platform to permit research into performance, architectural and management issues in ATM networks [Leslie91, Black94b]. Fairisle is based on a network architecture composed of ATM switches interconnected to form a general topology LAN; the Fairisle switch is a descendant of the Cambridge Fast Packet Switch. The switch is described below in some detail because it is the platform upon which much of the research described in this dissertation was based. Further details can be found in [SRG94].

2.1.1 The Fairisle Switch Fabric

The Fairisle switch fabric is based on a simple, 4×4, input buffered, self-routeing crossbar element. Switches of degree greater than 4 are constructed using multiple 4×4 crossbars in a 2-stage delta interconnection network. The 4×4 crossbar is internally non-blocking, but the delta configuration is internally blocking, at the output of the first stage of crossbar elements. The fabric architecture is pictured in Figure 2.1. Blocking occurs when two or more inputs simultaneously attempt to route a cell to the same output of a single crossbar. A priority bit allows high priority traffic to preferentially succeed in contention, and round-robin arbitration is used to select between contending inputs with the same priority.

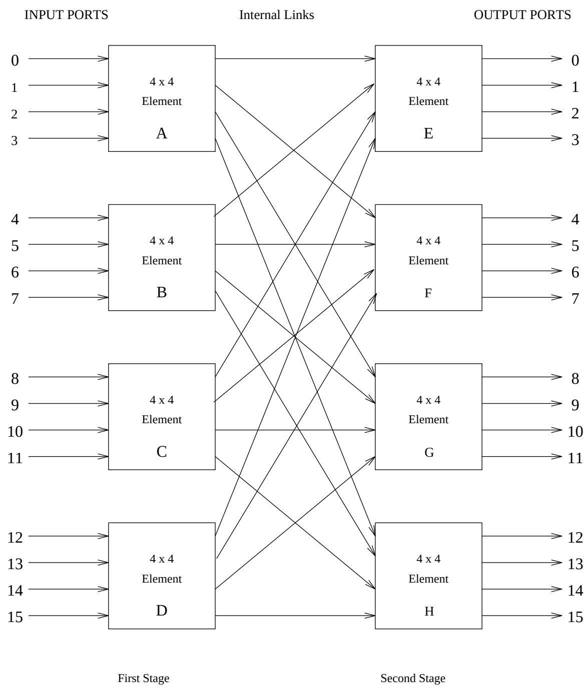


Figure 2.1: The Fairisle switch fabric design

Transmission across the fabric interface is cell synchronous. At the start of each slot all input ports which wish to transmit launch their cells into the fabric, where routeing takes place. In the event of a collision, one cell proceeds through the fabric and the remainder are rejected. An input port is informed by means of a negative acknowledgment (NACK) signal when its cell fails arbitration.

Fabric blocking reduces the effective throughput of the switch because rejected cells must be retransmitted. Two features improve the performance of the Fairisle switch: first, the fabric is built from crossbar elements of degree 4 rather than 2, and second, the fabric operates faster than the input and output ports. In the current implementation the fabric operates at 124 Mb/s, and the transmission links at 100 Mb/s. A small FIFO buffer performs rate adaptation between the fabric and output ports.

2.1.2 The Fairisle Port Controller

The implementation of traffic control policies in Fairisle is entirely soft. Each input-output port pair is managed by a single line card, or port controller. The output of the switch is passive and all queueing and policy issues are handled at the inputs. Each input is equipped with 2K cell buffers, 4 MB of RAM and an ARM RISC processor. The design of the Fairisle port controller has been improved over 3 revisions; the current design, named the FPC3, was used for the experimental work described in Chapters 5, 6, 7 and 8.

The hardware design of the Fairisle port controller permits a high degree of experimental flexibility owing to the use of field programmable gate arrays (FPGA) which can be re-programmed to implement facilities required for experimental work. A Xilinx FPGA arbitrates access to the cell buffers between the processor, reception from the ATM transmission link and injection into the switch fabric.

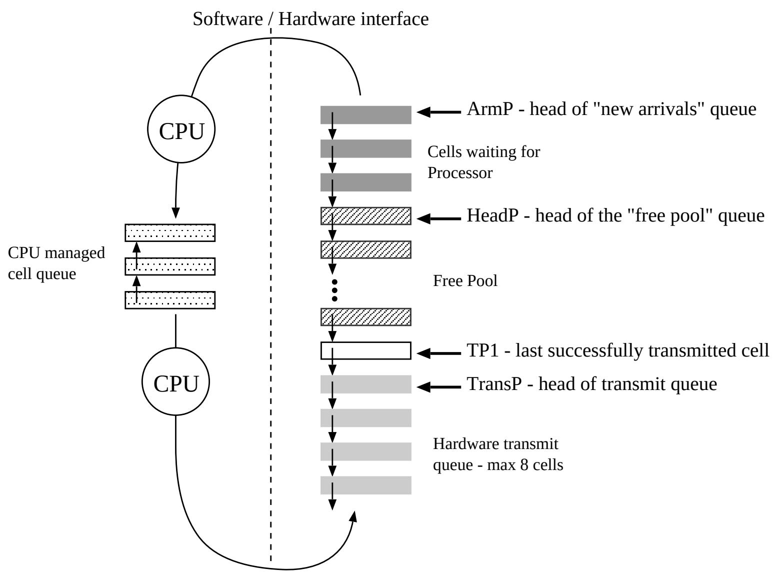


Figure 2.2: Buffer management on the FPC3 Xilinx

The Xilinx also manages cell reception and fabric transmission, and the pool of free buffers; Figure 2.2 illustrates the mechanism. An arriving cell is inserted into the buffer at the head of the free pool (pointed to by *HeadP*), and joins a queue of cells waiting for CPU processing. If the queue (pointed to by *ArmP*) is non-empty then the processor is interrupted. The CPU removes buffers from the arrivals queue and inserts them into a processor-managed queue for forwarding into the Fairisle fabric. Cells destined for the local system are removed at this point. Any intelligent processing of the cell stream, including scheduling and buffering, must be implemented by the processor. When space is available the CPU may add cells to a hardware managed transmit queue (with a maximum size of 8 cells). Each cell is given a “hardware retry count” (with a maximum value of 7) which determines the maximum number of times the Xilinx will automatically retransmit the cell into the fabric after a NACK; the processor is notified if this fails and may retry or discard the cell. Buffers which have been transmitted automatically rejoin the free pool.

2.1.2.1 Support for Experimental Work

The FPC3 Xilinx implements a number of features which support experimental work, including:

- a 16 bit free-running clock which increments at the cell slot rate ($3.556\mu s$) and can be used as an accurate absolute time reference,
- a 16 bit timer clock which decrements at the cell slot rate and which can be programmed to interrupt the processor at a specified slot time, and
- hardware support for automatic delay stamping of cells – a non-zero value of the high order bits of the cell VCI is used as an offset into the cell payload where the cell is stamped with the value of the free running clock on its arrival and departure. Special signalling support is required to make use of this feature.

The processor section of the port controller is similar to an Acorn personal computer, and is equipped with an I/O bus which enables it to be attached to Ethernet and other devices. The ARM processor was chosen because it supports a low latency Fast Interrupt reQuest (FIQ) mode which is used by the Xilinx to notify the CPU when:

- a cell has arrived on the reception or loopback interface,
- space is available on the hardware transmit queue,
- the free buffer pool is exhausted or a cell has exceeded its hardware retransmission count, or
- the timer clock has expired.

The Null Port Controller

The Null Port Controller is a simple port controller which implements only basic VCI remapping functions and FCFS queueing. It is controlled by a Fairisle port controller via a separate control interface, and enables an inexpensive switch with limited intelligence to be built.

2.1.3 Multicast in Fairisle

The FPC3 implements a simple, serial multicast mechanism: when a cell emerges from the switch fabric, it can optionally be copied both to the transmission module and back to the input port controlling that output. The input port can then retransmit the cell in a similar fashion. Special management of the VCI space is required to use this feature, which replaces an earlier multicast fabric based on a slotted ring, which could be attached to the input side of each port controller at a switch [Doar93].

2.1.4 The Desk Area Network

Using the Fairisle switch fabric as a workstation interconnect, Hayter designed and built the Desk Area Network (DAN), a workstation intended to support multimedia applications [Hayter93]. The DAN uses an ATM fabric within the workstation to interconnect the CPU, memory nodes, an ATM-based frame store, and workstation devices and the ATM network. Internally the ATM fabric is reliable, and provides frequent and predictable preemption points, allowing the CPU to schedule the interconnect to ensure that all multimedia streams within the workstation meet their requirements. Multimedia streams are routed directly to the devices which consume them, freeing the CPU and improving performance.

2.2 The Multi-Service Network Architecture

The ATM protocol stack implemented at the Computer Laboratory is called the Multi-Service Network Architecture (MSNA) [McAuley90]. Key to the efficiency of MSNA is its use of *lightweight* Virtual Circuits (VCs) which can be quickly set up and torn down, and which optimise for the common case, namely error-free communication [Leslie83]. Layering, multiplexing and error control are minimised; MSNA provides guarantees only on the integrity of data actually delivered end-to-end. In MSNA an ATM connection (called a liaison) is uniquely identified by the two ATM end-point addresses, called Service Access Points (SAPs), which it connects. A liaison represents a full-duplex, end-to-end communication path.

MSNA implements four functional layers, namely the Multi-Service Access layer (MS-Access), the Multi-Service Data Link layer (MSDL), the Multi-Service Network Layer (MSNL), and the MS Segmentation and Reassembly (MSSAR) layer. Figure 2.3 shows how the functional layers of MSNA relate to the well known B-ISDN protocol reference model adopted by the ITU-T. Two sets of procedures are identifiable in each layer, in the control and user planes respectively. The control plane functions are those used to set up, manage and tear down

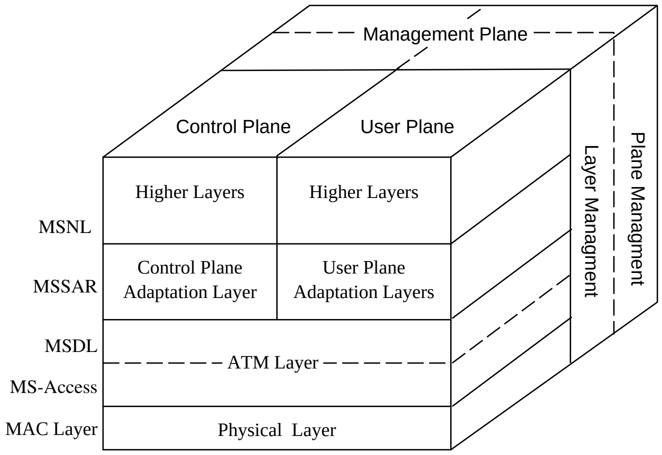


Figure 2.3: MSNA and the B-ISDN protocol reference model

liaisons, whereas the user plane permits applications to transfer data. The management plane is responsible for long-timescale control operations, such as fault detection and reporting, and configuration and topology management.

2.2.1 MS-Access

MS-Access provides MSDL with a consistent and uniform access mechanism to the underlying Media ACcess (MAC) layers of different networks. It supports simple "request" and "indication" primitives for transmission and reception of cells, and is responsible for the delivery of resources to meet the QoS constraints of different liaisons. Depending on the MAC layer, multiple MS-Access instances may be defined, each delivering a different QoS.

2.2.2 MSDL

MSDL is concerned with cell-based communication on a hop-by-hop basis, using communication paths called *associations*. It is a data link layer protocol, and is responsible for dealing with addressing (VCI format) and other ATM layer functions. Although it is a data link protocol, MSDL does not offer any error correction facilities. Its implementation is network architecture dependent: definitions exist for Fairisle, for earlier ATM networks such as the CFR and CBN, and for the Ethernet. In the latter, cells belonging to a single VC are packed into an Ethernet frame to form a "fat cell", making the Ethernet appear to higher layers like a conventional ATM network. In Fairisle, MSDL defines the structure of the ATM cell header and is called FDL. In practice the functions of MSDL and MS-Access are generally combined into a single functional layer.

2.2.3 MSSAR

MSSAR is responsible for segmentation and reassembly of user data, and for error detection. As such MSSAR is equivalent to the ATM adaptation layer of the ITU-T model. Early definitions of FDL pre-dated the current standards compliant cell format, implementing a block based reassembly protocol for MSSAR. A standards compliant ATM Adaptation Layer 5 (AAL5) is now also supported. A Protocol Data Unit (PDU) which fails reassembly (due to cell loss or corruption) is discarded without notifying MSNL.

2.2.4 MSNL

An MSNL liaison is formed by concatenating one or more associations, forming a single path linking two MSNL SAPs. MSNL adds no additional functions to the basic data transmission service offered by MSSAR. MSNL is an internetworking layer. An address in MSNL, also called an MSNL Service Access Point (SAP) uniquely identifies both the interface to the end system through which the connection is routed, and the MSNL user within the end system. An MSNL SAP may be advertised as an access point for a service. A liaison is uniquely identified by a pair of SAPs which it interconnects. It is an end-to-end unit of communication and represents a full duplex communication path. A one-to-one mapping of liaisons to MSDL associations ensures that multiplexing occurs only at MS-Access, streamlining the data path.

In MSNA no error control functions are provided. Protocol processing at the non-endpoint stations of a liaison is minimal: MSDL and MS-Access perform cell forwarding where possible. QoS criteria and routeing information are associated with each VCI at each node. For liaisons which cross network boundaries MSNL routeing is required. The mechanism used to set up and tear down connections in MSNA is based on a simple metasignalling protocol which is described in more detail in Chapter 3.

2.2.5 Software Support for MSNA

Support for MSNA is implemented in various operating systems at the Computer Laboratory, including DEC Ultrix 4.3, Wanda and various versions of the Pegasus operating system. Under Ultrix, access to a raw MSNL protocol layer is provided via the BSD 4.3 socket interface [Leffler89], which permits applications to set up connections and to send and receive data. The implementation available at the start of this research incorporated all MSNA functionality within the kernel. As part of the practical work described in Chapter 5, this was replaced.

2.2.5.1 Wanda

The Wanda micro-kernel, developed initially by Dixon [Dixon92], and extended by other members of Systems Research Group, including the author, was designed to enable systems research using high speed networks, both in end systems and within the network. Wanda is a single address space system, and supports lightweight threads, both within the kernel and in user space. The kernel itself is multi-threaded, a feature which is used to implement the Inter-Process Communication (IPC) mechanism, which is based around MSNA.

Both control and user plane MSNA functions are implemented within the kernel. An application interface to a raw MSNL service is provided, and although provision was made for the support of different QoS parameters for each connection, no specific QoS mechanisms were implemented. The user interface to the IPC system is similar to the BSD socket interface, with two notable exceptions. First, it permits an active connector to specify the QoS for each direction of communication; this may be modified by nodes on the path and the passive, "listening" application which accepts the connection. Second, it extends the semantics of connection acceptance: a connection request to a passive listener will not be accepted by its operating system until the application itself has accepted it.

In Fairisle, Wanda runs on each port controller of a switch. All port controllers operate independently, making management of the switch bandwidth difficult. This failing is addressed by the communication architecture presented in the following chapter. On each port controller, the Fairisle interface is represented within the local Wanda system as two logical devices, namely a point-to-point transmission interface, and a fabric interface which enables the port to communicate with other port controllers on the same switch.

2.2.6 Higher Protocol Layers

MSNL provides a raw, unreliable data service to higher protocol layers. Implementations of TCP/IP on MSNL are also available, both for Unix and Wanda. A typical requirement at the application level is for a distributed computing environment. This is provided by the Multi-Service Remote Procedure Call mechanism, version 2 (MSRPC2).

2.2.6.1 MSRPC2

MSRPC2 is a simple and efficient RPC mechanism for distributed systems programming, specifically designed for use with MSNA [Roscoe94]. The MSRPC2 runtime was developed by the author and the user interface by Timothy Roscoe, the author, and Richard Hayton of the Computer Laboratory. It is based on aspects of the ANSA [ANSA89] Computational Model and the DEC SRC Network Objects system [Birrell94]. It differs fundamentally from them at the low level in that it is extremely lightweight: bindings are implicitly single-threaded and have a one-to-one correspondence with MSNL liaisons. MSRPC2 also incorporates transparent optimisations for calls within a single address space which map invocations onto single procedure calls.

The user interface represents the MSRPC2 computational model: each service is specified in a typed interface definition which supports inheritance and subtyping. Each interface describes the data types which traverse it, operations which are exported by the server and their arguments, and exceptions which may be raised by the operations. The Interface Description Language (IDL) used is called MIDDL and is a minor extension of ANSA IDL.

RPC Stubs and Marshalling Code

The stub code is generated from the interface specification by the stub compiler, and is specific to the interface type. It consists of type definitions for use by client and server, and stubs which make remote invocations and exceptions look like their local equivalents.

Marshalling code is also partially generated by the stub compiler; for each data type defined in an interface it provides C functions or macros for marshalling and unmarshalling an instance of the type, and freeing memory allocated to hold variable size data. The wire representation used by MSRPC2 for encoding data types, MSDR, is based on the procedural mechanism of XDR [Sun87], except that the byte ordering used by a client is specified by the server.

The MSRPC2 Runtime

The MSRPC2 runtime is implemented over the POSIX Pthreads interface [IEEE92], and is portable apart from a layer which is used to hide the platform specific IPC primitives. MSRPC2 libraries exist for both Wanda and DEC Ultrix.

MSRPC2 exploits the reliability of the basic MSNL transport: by default it does not implement a reliable transport protocol on top of MSNL[1]. Errors arising at the network level are merely reported to the user, and the semantics of the RPC mechanism reflect this. State associated with the binding between a client and server is maintained at the connection endpoints. The use of MSNL has the following consequences:

- Operation invocations and replies may (infrequently) be lost. MSRPC2 thus cannot guarantee that a remote operation will be executed. As a

[1] A reliable transport using TCP is also available.

result the only semantics supported when using an MSNL transport are At Most Once (AMO) [Birrell84] semantics. Clients are notified of operation failures and may retry them.

- By keeping per RPC overhead to a minimum (8 octets), MSRPC2 permits a typical RPC call and its arguments to fit into a single ATM cell.

- The MSRPC2 interface reference permits a service to specify its QoS requirements, so that data transferred across its interface will be properly delivered by the network.

2.3 Operating Systems Research

Recently the Pegasus project [Leslie93], a joint effort involving the Computer Laboratory and the University of Twente, has designed a novel operating system which supports fine grained sharing of resources and QoS guarantees, in essence carrying the ATM concept into the heart of the operating system. Application domains receive explicit notification of all system activity which affects their QoS, and may dynamically negotiate for resources. This enables them to behave intelligently in response to the received resource allocation. Three prototype implementations of the architecture have been implemented. Hyden [Hyden94] implemented a prototype system called NEMO, defining a QoS paradigm which enables the operating system to support continuous media. Black [Black95] used the Fairisle port controller as a platform for the development of *Fawn*, an operating system with support for explicit scheduling of network activity, and support for optimized I/O. Roscoe [Roscoe95] implemented the Nemesis operating system which extends Hyden's work, integrating traditional workstation activities and multimedia but preventing "QoS crosstalk" between applications.

2.4 Summary

This chapter has described the environment in which the research described in the following chapters was undertaken. Further details of the Fairisle platform, and the MSNA protocol suite will be introduced where required.

The systems environment at the Computer Laboratory provides an ideal vehicle for research into performance management in ATM networks. The Fairisle network and the MSNA protocol suite provide the experimental flexibility required to support the investigation both of novel network signalling and control mechanisms and the effects of dynamic admission control policies. MSRPC2 provides general purpose distributed computing facilities which may be used to assist these efforts. Chapter 3 presents and architecture for distributed control of an ATM network which is ideally suited to managing such an environment, but which is also applicable to any multi-service network.

Chapter 3

A Connection Management Architecture for MSNA

High speed networks offer enormous opportunities for the development of new applications and communication services, but to fully exploit them, improved network control mechanisms and service interfaces are required. This chapter describes an experimental control architecture for the MSNA protocol suite. The architecture aims to simplify the management and control of a multi-service network and facilitate the development of sophisticated services, whilst retaining the flexibility required to permit research into traffic control mechanisms for ATM networks.

3.1 Motivation

New multimedia applications require a rich, flexible communications interface which permits them to establish and manage connections carrying a wide range of traffic types which have synchronisation and other performance requirements. Calls involving multiple connections, multiple media, and multiple participants are expected to become the norm. For its part, the network must provide the sophisticated communications service interface which these applications require.

What is meant by the term "service"? Traditional telecommunications services are ill suited to an era in which network nodes and end-systems have become powerful computers. It is necessary to distinguish between basic communications services such as point-to-point or multicast connections, and more sophisticated, "value added" services, such as synchronisation, which may be implemented by computing applications. The network service interface is represented by the signalling protocol which allows users (applications) to request the appropriate facilities.

3.1.1 The Location of Communication Services

One way to support increasingly complex service demands is to locate all services within the network – the so-called Intelligent Network approach, discussed in some detail in Section 1.3.2. This has the major drawback that the addition of each new service requires the network nodes to be modified to incorporate "hooks" for the new service features, and the addition of new signalling facilities to the user access protocols. Not only does this increase the complexity of the network control software, but it also increases the risk of "feature interaction" and catastrophic failure. It is also unattractive because, historically, network operators have been poor at developing new services. Deregulation of the telecommunications market should encourage third-party service providers to develop and sell new services implemented *on top* of a single underlying network.

It is thus important to distinguish between the service interface which should be offered by the network, and the communication services which are offered by advanced network applications. This distinction has not been necessary until recently – traditional telecommunications services, for example telephony and X.25, define both the service itself and the procedures for accessing it. ISDN, on the other hand, provides a relatively basic connection service but allows users to access (unspecified) sophisticated network services by including service-specific elements in the signalling protocol. The advent of ATM, and the notion of a single underlying technology which can support all communications services, raised the spectre of a "grand unifying service interface" to the B-ISDN, in the form of a signalling protocol which includes service-specific features for all applications, present and future. Clearly this is not only undesirable, but also impossible to achieve.

ISDN confuses two concepts which must be treated separately:

- how users should signal to services, and
- where the services should be located.

By requiring all service-specific signalling elements to be expressed using a single protocol it forces those services to be located within the network. An alternative, which forms the central aim of the architecture described in this chapter, is to keep the network interface as simple as possible, implementing complex services on top of a carefully designed network interface which offers only basic communication facilities. This approach has numerous advantages:

- The simpler network interface can be more easily specified, yielding greater confidence that it will operate correctly.
- Problems of feature interaction are eliminated because the network remains consistent internally.

- As network equipment changes, and standards evolve, the mechanisms used internally to implement the basic service primitives can change without affecting users.
- The use of a simple, open interface will encourage the development of sophisticated services outside the network.

3.1.2 The Network Service Interface

How should the open service interface be defined, and what primitives should it incorporate? Ultimately users should be able to signal their requirements across the network interface using a protocol which is rich enough to support all their communications needs. The process of determining the set of connection primitives is far simpler than defining the set of network services.

It is now widely accepted that the signalling procedures should be located in an Out of Band (OB) control plane, for reasons of performance and efficiency. Current OB signalling protocols, including draft signalling standards for ATM such as those defined and standardized by the ITU-T [ITU-T94a] and ATM Forum [ATMF94], are complex and inflexible. This results from their origins in existing message-based ISDN signalling protocols which assume that all services are embedded within the network, under control of a single provider. Unfortunately they have failed to adapt to the different circumstances in modern networks where end systems include both sophisticated computers and "dumb" devices incapable of signalling.

The connection management architecture described in this chapter is designed using a distributed systems approach: the network exports a well defined service interface based on a procedural abstraction. Network users are its clients, and invoke operations on the service interface. Signalling and control functions both between user and network, and within the network are achieved using an RPC mechanism, and network control functions are located in a set of distributed programs. This approach offers a clear distinction between different aspects of network control. It also enables communication to be managed procedurally via a set of well defined, typed interfaces, as opposed to the monolithic approach of message passing. Use of a distributed computing paradigm allows communications services to be incorporated naturally into the distributed computing environment, and presents users with a high level, procedural abstraction to naming, location, and invocation of services.

3.1.3 Control of the Network

The network service interface is but one component of the network control system. Within the network, control functions must be exercised to build and parameterise connections. New applications have demanding resource and performance constraints which require a flexible approach to network control. The architecture described in this chapter takes an innovative approach to this problem which facilitates management of disparate types of network and equipment. Rather than require each network node and end-system to incorporate sufficient intelligence to implement signalling, management and other control functions, these are located in a set of distributed control services which need not reside on the node itself. The services are responsible for ATM address management, signalling, routeing, CAC, and traditionally low-level device control functions, including parametarisation of In Band (IB) control mechanisms which ensure that each connection meets its performance constraints. This approach is ideally suited to the support of multimedia communication architectures such as IMAC [Nicolaou91] which implement specialized control mechanisms for multimedia devices and streams.

Each node surrenders control of communication to a separate master service. A single master may be responsible for many nodes of different types; where necessary it implements a protocol conversion which enables it to communicate with different equipment types. Vesting control in a separate master permits the support of nodes with limited processing power and allows the network control functions to be implemented independent of architectural features of the equipment. This is particularly important in the ATM LAN environment, where the success of ATM hinges on development of low cost ATM systems which may not be sufficiently intelligent to implement the full control plane. On a practical note, it also permits the control services to be directly ported to different operating platforms, reducing implementation effort, errors and incompatibilities. By removing control from network nodes and locating it instead on intelligent computers surrounding the network, it becomes possible to manage simple devices such as TV set-top boxes, and ATM cameras and displays. Flexibility of the IB control functions is also essential, both because of the fluidity of ATM standards and because suitable control mechanisms have yet to be developed. The architecture presents a flexible model of policy implementation which can be modified by control functions operating on different time scales, and which facilitates experimentation with different traffic controls.

The architecture is described in the remainder of this chapter. It is presented in the context of the MSNA protocol suite, described in Chapter 2, which is used in all ATM equipment at the Computer Laboratory; as a result it is named the Multi-Service Network Connection Management Architecture (MSN-CMA). At the time of writing, the MSNA network incorporates two generations of the

Fairisle ATM switch and the Ethernet. Third party ATM switches, end systems and hosts are also attached; the MSN-CMA aims to enable their integration into a single coherent network. It should be stressed that the service interface of the architecture does not attempt to include all desirable communication primitives. Rather, it demonstrates the applicability of the distributed systems-based approach to network control. The approach has led to the definition of a comprehensive research project named DCAN (Distributed Control of ATM Networks) at the Computer Laboratory, which is described in more detail in Chapter 4.

The remainder of this chapter is structured as follows: Section 3.2 describes the control mechanisms defined for the original implementation of MSNA at the Computer Laboratory, highlighting several shortcomings of the definition. Section 3.3 then introduces the structure of the MSN-CMA, defining terms used throughout the description of the architecture. The control plane functions of the MSN-CMA are discussed in Section 3.4, including user and network signalling, and IB control mechanisms. Management plane issues are dealt with in Section 3.5, and Section 3.6 deals with higher level, "value added" services which must be supported by the network. References to some related work are given in the text; however a full review is deferred to the end of Chapter 4, in which details of the call control procedures supported by the MSN-CMA are described.

3.2 Existing Control Plane Functions in MSNA

Before describing the design of the MSN-CMA, it is necessary to review the control functions provided by MSNA, both as described by McAuley [McAuley90], and as implemented at the Computer Laboratory. MSNA was described in some detail in Section 2.2. Key to its success is its use of lightweight connections – in current implementations the control plane is similarly lightweight, with unfortunate consequences, as explained in the following subsection.

3.2.1 Out of Band Control

The OB control plane functions in MSNA are used to set up and tear down liaisons and are based on a simple metasignalling protocol. Metasignalling messages fit into a single cell, and are transmitted on a well known VCI. Only three primitives are supported for connection control, namely CONNECT, ACCEPT and KILL. Figure 3.1 shows the steps involved in connection setup. Initially a server application on host b requests an MSNL SAP on which it can receive incoming connections (step 0). The host allocates it a SAP, B, which the application then "offers" for incoming connection requests (1). SAP B is called a "passive" SAP, and may only be used to receive connection requests.

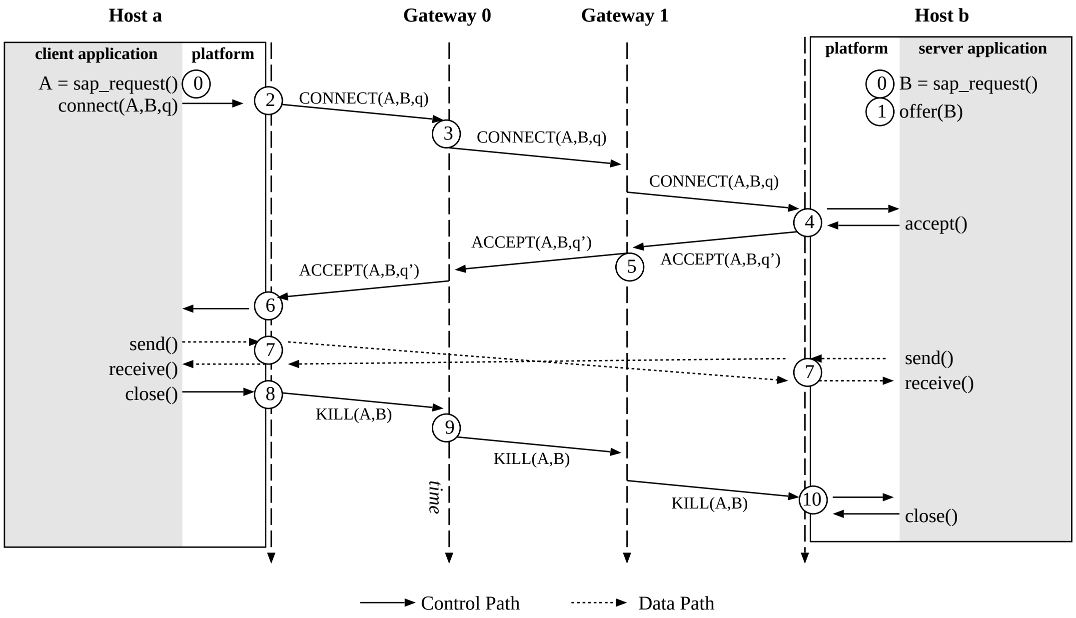


Figure 3.1: Liaison setup using the simple metasignalling protocol

On host a, in a similar procedure, the client application requests and is allocated an "active" SAP, A. It then requests the host to connect to passive SAP B (whose identity it must ascertain by some other means), on host b (2). Host a then broadcasts its CONNECT request on a link selected by its routeing algorithm, passing as parameters the source and destination addresses (A, B) and a QoS specification q for both directions of communication.

The metasignalling protocol permits the setup of simple associations. Higher level functions are required to build liaisons. Historically this function has been provided by gateways running the Wanda micro-kernel: Gateway 0 receives the incoming connection request on a so-called non-specific "wildcard" MSNL SAP (3). Realizing (using its routeing tables) that it is able to route the request on towards the requested destination address, gateway 0 attempts to place its own connection to the remote address. This process continues at additional gateways until the destination is reached. Host b presents the incoming connection request to the application which is listening on address B (4). If the connection is accepted then host b returns an ACCEPT message identifying the liaison (A, B) and a modified specification q', indicating the allocated QoS. At each gateway on the path, receipt of the ACCEPT message completes the association for that hop (5). The gateway immediately accepts the original request which it received and then "joins" the two associations which it spans. Depending on the network types joined[1], the forwarding mechanism may be implemented at different levels. On Fairisle, for example, the join entails setting up the cell-header remapping tables to permit efficient cell forwarding in hardware. For joins between networks of different types, such as Fairisle and the Ethernet, data forwarding must be done at the MSNL level, by the gateway.

When the ACCEPT message is received by the source host a (6), informs the application that communication may commence (7). At the end of communication, either party may issue a KILL message to destroy the liaison (8). At each gateway the joined associations are broken, destroying the communication path (9). The other party is also informed that the connection has been broken, and notifies its user (10).

3.2.1.1 Drawbacks of the Metasignalling Approach

The metasignalling protocol uses simple cell-based message exchange. Unlike the ITU-T and ATM Forum signalling protocols, it is unreliable because signalling cells may be lost. To overcome this a simple timeout and retry mechanism is used for connection setup messages only. A connecting host will continually re-broadcast its CONNECT request until either the user withdraws its request or a reply is received. The metasignalling protocol is not well defined, and the following errors and problems may occur:

- If any component on the path of a liaison should fail, none of the others will be informed and resources will not be reclaimed. This includes end systems, whose applications may then "hang" forever, waiting for data.
- Connections may be left open permanently, irrespective of whether they are in use, causing resource depletion.
- A node may find, on re-booting, that it is receiving data on existing connections which were routed through it before it crashed[2].
- No reliable mechanism exists to terminate connections – a KILL message may simply be lost and will only be retransmitted if the node continues to receive data.
- A poor choice of the retransmit timeout can cause difficulty when attempting to set up long liaisons.
- A passive end-system which receives a duplicate connection request for a connection which it has already accepted, has no way of knowing whether its reply was lost or the active user crashed and has restarted.

[1] Recall that MSNA is not ATM-specific.

[2] A special KILL mechanism was implemented to avoid this.

- An application or host which does not wish to accept a connection request has no mechanism for informing the connector of the decision or its cause, so the connector will retry until it times out.

To avoid these problems, a reliable signalling protocol is necessary. In the MSN-CMA this requirement is of key importance, and is dealt with in detail in Chapter 4.

A further difficulty with the MSNA control plane is that it assumes that each node in the network, including gateways, is capable of terminating the control plane functions, that is, that every node can signal to request connection setup and teardown. Moreover, the control path through the network is implicitly the same as the data path. This is unnecessarily restrictive because, increasingly, dumb devices are being attached to the network. An example is the Fore Systems AVA-200 ATM Camera[3]. The camera is capable of transmitting video directly into the network or the Fairisle switch fabric, but is incapable of signalling – it implements a simple control protocol which is used to inform it of the video parameters and VCIs to use for transmission. In order to successfully incorporate such devices into a multi-service network, proxy signalling is required whereby some other node signals on behalf of the device; the control path must be able to differ from the data path.

Previous implementations of MSNA [Dixon92, McAuley90] have combined the functions user and control planes into a single functional unit, normally bound within the operating system kernel of a host or network gateway in a system specific manner (following the example of UNIX [Leffler89] and Mach [Accetta86]). This makes the management of multiprocessor systems such as the Fairisle switch difficult, because it assumes that each processor (kernel) is responsible for its own management. It is impossible to manage shared resources, such as switch bandwidth, without managing the functionality of a switch as a single object with multiple network interfaces (possibly in a distributed fashion). The MSN-CMA thus argues for a wide separation of user and control planes, both within the network and in its attached end systems.

3.2.2 In Band Control

In band control in MSNA is achieved at the MS-Access interface, which presents MSDL with a number of access points to the underlying MAC layer, each parameterised to deliver a particular QoS. Although MSNA defines the control point for IB functions, namely MS-Access, in practice it has ever been implemented, and the mechanisms for setting up and parameterising IB control points were never defined.

[3]Based on the ATM camera described by Pratt in [SRG94].

3.3 Structure of the MSN-CMA

The motivation of the introduction, combined with lessons learned from the failings of the simple metasignalling protocol, lead to a statement of the main architectural requirements of the MSN-CMA, which include:

- A rich set of features for use by multimedia applications which demand a range of QoS parameters and modes of connection.

- Low latency, reliable, out of band user-network and intra-network signalling for connection setup and teardown, with distinct control, management and data paths.

- Support for the control of diverse equipment types using a common set of interfaces.

- Location of the control and management plane functions outside the network nodes and possibly outside the network.

The following subsections describe the basic components of the MSN-CMA, introducing the concepts which will be used in the detailed discussion of the network control functions which follows.

3.3.1 Network Entities and Management Domains

A *network entity* is defined to be an object which has one or more network interfaces, and which implements the *user plane* of the MSNA protocol stack for each of those interfaces. Using this definition a single host is an entity, as is a gateway bridging two networks. It is necessary to take into account the physical structure of network devices when applying the definition: the set of ports on a Fairisle switch is most conveniently considered to be a single entity (a switch) rather than a collection of independent entities connected by a network with special properties (the fabric), in spite of the fact that multiple processors are involved. This makes sense especially when considering third party switches which typically use a single processor to control multiple ports – such devices can only be regarded as a single entity. Of course, it is also possible to view a Fairisle switch as a set of entities, if this is more convenient. In this case one or more port controllers will support the services implementing the MSN-CMA, and each switch will be responsible for its own connection management.

A collection of network entities grouped together for the purposes of management and control form a *Management Domain* (MD)[4]. Rather than require each

[4]This definition is not related to the ISO [ISO91] definition of the same name.

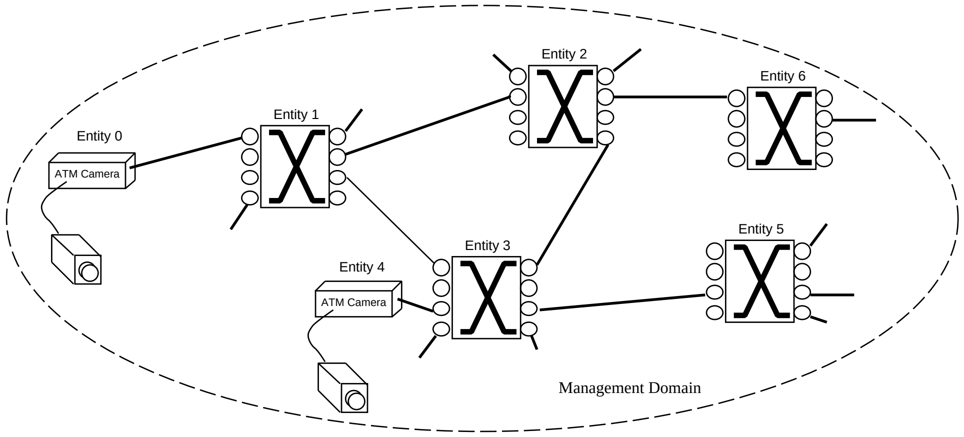


Figure 3.2: An Example MSNL Management Domain

entity to implement the control and management planes, each entity within an MD surrenders all control and management plane functions of MSNA to a set of cooperating, distributed services which implement the required functions on its behalf. Although the grouping of entities into a domain may be arbitrary, in practice an MD will generally reflect a logical grouping of entities, such as a single switch (in which case the entities are its ports) or an entire ATM network. Within an MD the boundaries between entities are preserved – this is required because each entity is still capable of independent communication.

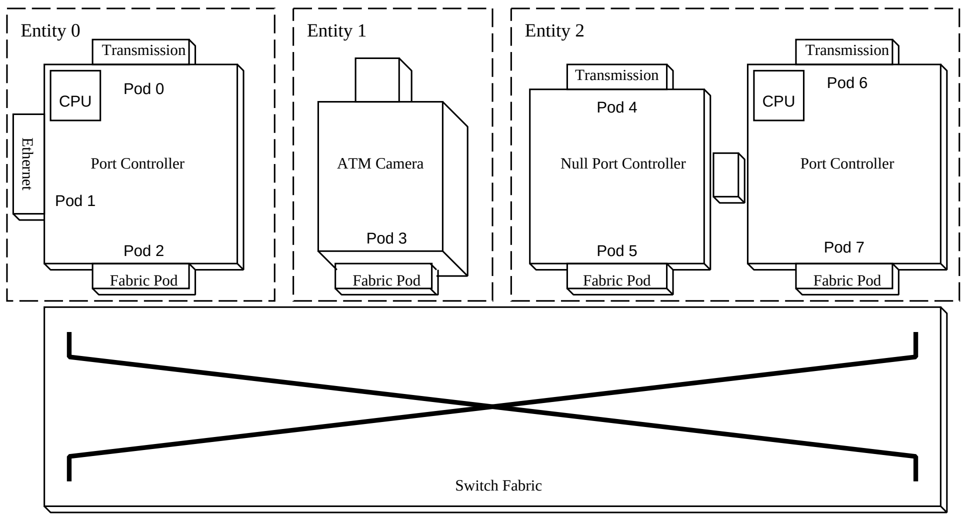


Figure 3.3: A Fairisle switch Management Domain

Figures 3.2 and 3.3 illustrate two examples of management domains; the first is an ATM network with attached third party devices, and contains six entities: two ATM Cameras and four switches. The second is a single Fairisle switch composed of 3 entities. Entity 0 is a Fairisle port controller with an Ethernet interface. Entity 1 is an ATM Camera, and entity 2 is a Fairisle port controller and an attached Null Port Controller[5]. This entity has 4 interfaces.

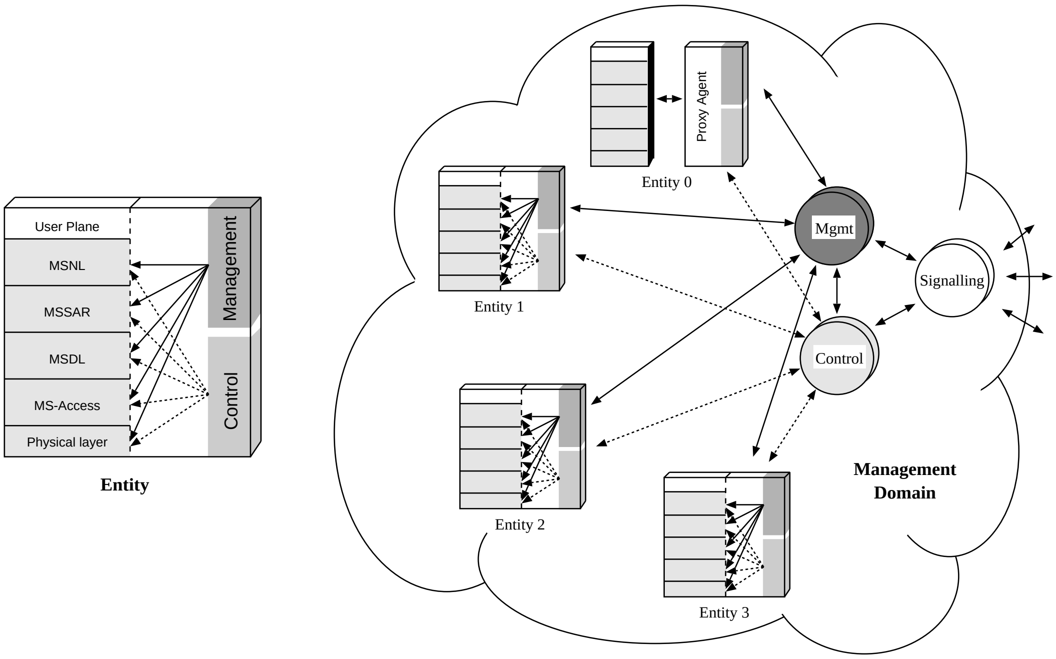


Figure 3.4: A network entity exports interfaces in the control and management planes; multiple entities are grouped into a Management Domain

Within the MD each entity is abstracted as an object which exports a set of interfaces which provide access to MSNL, MSSAR, MSDL, and MS-Access within the entity. The interfaces are used by the control and management services within the MD to construct, parameterise and destroy liaisons on behalf of the entity. For example, at the MS-Access layer an entity may export a management interface which permits the MD's management function to detect the error rate for the interface, and a control interface which permits the MD to parameterise a policing algorithm for a connection. The control interfaces are described in Section 3.4.

Not all entities within the network will conform to the interface requirements of the control functions within the MD. A third party ATM switch, for example, or an ATM Camera, may use a proprietary protocol to control communication. Since the management and control services expect each entity to export the appropriate

[5]These components were described in detail in Chapter 2.

interfaces, however, they must be implemented by a *proxy agent* which exports on behalf of the entity the appropriate set of control interfaces, whilst using the proprietary protocol internally to manage the device. Figure 3.4 shows the basic structure of an MD: each entity exports control interfaces which are used by the control and management plane functions to facilitate communication for all entities in the MD. The diagram also shows a proxy agent signalling on behalf of a primitive entity.

An MD is regarded as a single functional unit insofar as management and control are concerned. When an entity in an MD fails, then the MD is considered to have failed only if any of its essential control and management services fail. In particular, if the MD contains the whole MSNA network, then it is essential to ensure that the MD remains functional and that only the entity is restarted.

3.3.2 Networks and Subnetworks

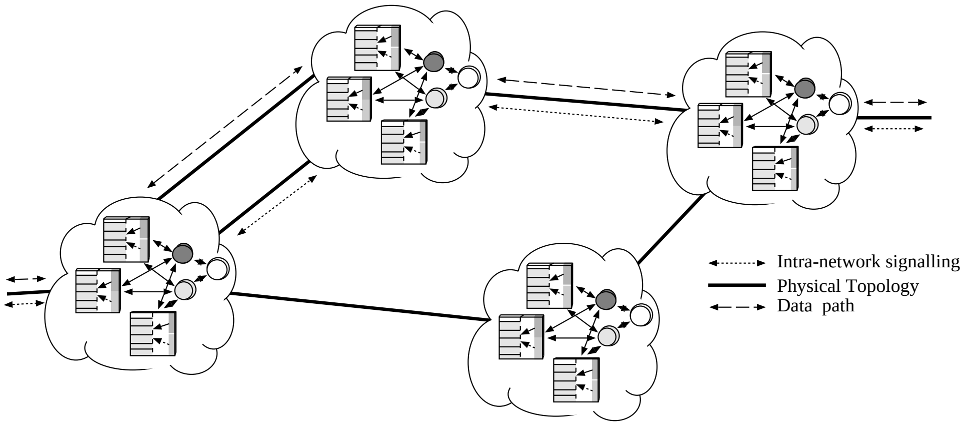


Figure 3.5: A network of management domains forms an MSNA network

A set of management domains together form an MSNA network. MDs are peers in the MSN-CMA and communicate using intra-network signalling. The set of MDs is overlayed on the physical topology of the underlying network. Each MD exports a procedural signalling interface. It invokes operations on the signalling interfaces of the MDs adjacent to it in order to request connection setup and teardown on behalf of both its entities, and entities located in other MDs whose connections are routed through it. Figure 3.5 depicts the structure of the MSNA network which results, showing the path followed by intra-network signalling and the user data (these need not be the same) for a connection traversing the network.

MSNA differs from many network architectures in that it supports multiple network *types*, such as ATM and the Ethernet. Each type is characterized by its MAC layer properties, such as the underlying frame size, the media access mechanism used and its reliability. For example the Ethernet supports variable sized frames, and uses distributed access control, whilst an ATM network supports 53 byte cells and point-to-point links. Each type has different performance limits of which the control plane functions must be aware. In MSNA the network type-specific features are encapsulated by MS-Access, which provides to MSDL a consistent interface for the transmission of cells.

MSNA is an internetworking protocol – liaisons may span multiple network types. Addressing and routeing issues are managed by MSNL, and require the identification of MSNL subnetworks. A subnetwork is typically defined by the boundary of a network type, but in the case of point-to-point networks the set of links terminating at an entity defines a subnetwork. Subnetworks may be hierarchically composed. Each network interface (also referred to as a pod) attached to an entity is assigned a unique interface address. Each entity is assigned a primary address which is the address of one of its interfaces, using its other addresses as aliases. Pod and entity addresses define the topology of the network and are used for routeing.

Where an administrative boundary exists between two subnetworks, the management and control functions for the network entity bridging the two subnetworks may be split between two MDs. In this case each MD implements management and control functions for the interface(s) and associated protocol layers which lie within its administrative boundary.

The control and management plane functions are implemented by a set of co-operating services within each MD. Section 3.4 and 3.5 describe in detail the implementation of these functions by the services in each domain.

3.4 The Control Plane

The control plane functions of MSNA are those concerned with connection setup and teardown. This section describes three interfaces which are required to implement the architecture, at different layers of the MSNA protocol stack (Figure 2.3 in Chapter 2 displays them graphically). At the MSNL level an interface is used by network users to request connection services from the MD; this is discussed in Section 3.4.1. A second interface at the MSNL layer, described in Section 3.4.2, is used by peer MDs to achieve connection management throughout the network. Finally, Section 3.4.3 describes an interface exported by each entity which allows the control plane to manage the layers below MSNL.

3.4.1 The User Master Interface

The MSNA protocol suite is supported by many operating platforms, each of which provides users with a different abstraction of (and interface to) the communication facilities offered by MSNA. Any such system is by definition an entity in some MD. The MSNA control functions are located in a set of cooperating services which preserve the layer boundaries within MSNA. In a conventional communication architecture, these functions are presented to the user (application) via a control interface which is platform specific; they may, for example, be represented by the BSD socket interface [Leffler89] or the Wanda IPC interface. Beneath this interface the platform implements the control plane functions to build the appropriate connection components for the user. Most existing application programming interfaces (APIs) are insufficiently descriptive to allow proper control of the communication needs of multi-service applications. This is discussed in more detail in Chapter 5, however the development of an ATM API is beyond the scope of this dissertation.

In the MSN-CMA the MSNL control plane functions are implemented by the MSNL Master service. To request network layer control functions, the user must therefore signal its requirements to the MSNL Master. The MSNL Master exports a procedural interface for this purpose, called the User Master Interface (UMI). The UMI is thus the interface between a user wishing to use MSNL, and the MSNL control plane. Each operating platform must translate calls across its specific communication interface (called the Platform Specific Interface, PSI) into calls across the UMI, on behalf of each of its MSNL users. The UMI defines the signalling protocol with which the user controls all aspects of the communication. This includes the allocation of MSNL SAPs and the set up and teardown of MSNL connections. On the user side of the UMI a state machine represents the user's view of the current state of a connection. Within the MSNL Master a similar state machine is used to control the connection.

The MSNL Master maintains the definitive state for every call within the MD, and is responsible for management of the MSNL address space, in the form of MSNL SAPs, for each entity which it controls. The UMI protocol specifies the valid state transitions for a call which can occur in response to user requests and network events. The UMI presents the user with an MSNL SAP, which uniquely identifies the user service access point in the MSNL network. A pair of SAPs uniquely identifies a call, and is called a liaison.

Consider the steps involved in connection set up between a user and a remote SAP. First, the user requests a SAP for its end of the connection, via the PSI. The platform supporting the user then signals the request across the UMI. The MSNL Master responds by returning a SAP which is bound to a call reference whose value is unique within the MD, and which is a required argument in each request across the UMI.

Having been allocated a SAP the user can signal a connection request across the UMI (via the PSI). In response the MSNL Master takes steps to establish a communication path between the two SAPs. In general this will involve control actions on other interfaces, described below. Once the connection has been set up, the MSNL Master informs the user, and data transfer can commence. On termination of the call the user frees its SAP by disconnecting the call. Section 3.4.8 describes the steps involved in connection setup in more detail.

Note that the definition of an MD does not require the MSNL Master or any of its associated services to reside physically on an entity or entities within the MD. It would be perfectly feasible to locate many MSNL Masters for different MDs on different subnetworks on a single host, possibly within a single process, if required.

Performance Issues

An entity signalling on behalf of several applications simultaneously may maintain more than one UMI binding to the MSNL Master. Since each UMI operation takes as argument the call reference, interleaving requests for different connections is quite acceptable. Further, the MSNL Master functions may be split between several processors, or located on a single, powerful processor to ensure that the control plane performance is satisfactory. Chapter 5 comments on performance in detail.

3.4.2 The Master Master Interface

If a user attempts to connect to a remote MSNL SAP (that is, a SAP which is not managed by the user's MD), the MSNL Master must signal to the remote MSNL Master which is responsible for that SAP, requesting a connection. The problem of locating the remote MD is called the routeing problem, and is discussed in Section 3.4.7.2. Typically there will be several MDs between the source and destination. Each of these is involved in setting up the communication path through the entities under its control. In this case inter-MD signalling must take place for every "hop" which the connection will traverse. To communicate its request to the MSNL Master within the MD of the next hop, the MSNL Master invokes operations on the Master Master Interface (MMI). The MMI is similar to the UMI, in the sense that its primitives are MSNL layer primitives, however it need not be concerned with SAP allocation, since this is a private matter within each MD. The definitions of the MMI and UMI are independent, however the MMI must support at least the basic primitives required by the UMI. MDs are peers within the MSN-CMA and thus the MMI signalling interface is symmetrical. Across the MMI each call is uniquely identified by the pair of SAPs which it connects; here the call reference is the globally unique liaison identifier.

Compatibility with Other Systems

Both the UMI and MMI involve signalling between network elements. Equipment which supports, for example, standards-based signalling may also be incorporated into the network. In this case a protocol conversion is required between the MMI and the protocol implemented by the equipment. This can be achieved via a proxy-MMI which performs the conversion.

3.4.3 The Slave Control Interface

Within the MSNA control plane, functions are required at layers below MSNL to construct the data path within each entity through which the liaison passes, and to parameterise and monitor the IB control mechanisms required to deliver the requested QoS to the liaison and the network as a whole. Each entity must therefore export a control interface which the MSNL Master can use to direct the construction and destruction of data paths within it. This interface is called the Slave Control Interface (SCI). The SCI incorporates functions from both the control and management planes, allowing the MSNL Master to:

- build associations within MSDL,
- determine resource availability within the entity (to aid admission control),
- assign an association to an MS-Access instance which will deliver it the appropriate QoS,
- assign an association a MAC layer VCI for transmission, and a VCI on which to expect arriving data, or specify a VCI mapping at a switch,
- parameterise IB control mechanisms, for example setting the leak rate on a leaky bucket policing mechanism, or selecting the CLP bit to be set in the cell header,
- parameterise alarms which are used to notify the MSNL Master of exception conditions on the interface, and
- retrieve and reset MSDL and MS-Access performance statistics.

The notification mechanism for alarms takes the form of an interface exported by the MSNL Master which is used by the management plane functions within the entity to report alarm conditions, such as network interface errors and performance exceptions.

3.4.4 Specification of MSN-CMA Interfaces

The interfaces exported by the control and management plane services are procedural, and are defined in an Interface Definition Language (IDL). A prospective client binds to an instance of the service interface and is presented with an object-oriented abstraction of the service which enables it to invoke operations on the service interface. The procedural abstraction is supported by a lightweight RPC mechanism, namely MSRPC2, which was described in Chapter 2. The MSRPC2 IDL, MIDDL, permits a procedural specification of the control interfaces in the network, including the syntax of the communication, the operations in each interface, their arguments and return values, and possible exceptions which may be raised. Most IDLs used in current distributed systems are unable to express the semantics of the interface, a problem which is addressed in Chapter 4, which investigates methods which allow formal specification of the interfaces.

In most telecommunication networks signalling is message based. In this paradigm the users and network communicate by exchanging messages whose format is described in the specification. Message passing permits both user and network to initiate communication, whereas the client-server paradigm assumes that all communication is initiated by the client. To facilitate initiation by both parties, the MSN-CMA implements a service interface at each. The user invokes operations on the MSNL Master, whilst the MSNL Master invokes operations on the complementary interface at the user. Without this there would be no way, for example, that the network could inform the user of an incoming connection request. Thus, although the UMI is described as a single interface, it comprises two complementary separate service interface definitions, one each for user and MSNL Master. The requirement for two service interfaces is a small price to pay for the advantages of the RPC-based approach, which greatly simplifies the interface definition and dynamic aspects of the communication. This is discussed in detail in Chapter 4.

3.4.5 Construction of the Data Path

The MSNL Master instructs each entity within its MD to build the elements of the data path required to link the SAPs involved in a liaison. There are three types of liaisons, namely *local, remote* and *forwarding*. Local liaisons are those between two users on the same entity (platform). In this case the platform will in general provide an efficient and optimised mechanism for user–user communication. In BSD UNIX, for example, local communication at the socket layer involves simply inserting a buffer on a socket data structure and rescheduling the receiving process. Remote liaisons are those which connect two users on different entities. The remote entity may be within or external to the MD. If it is within

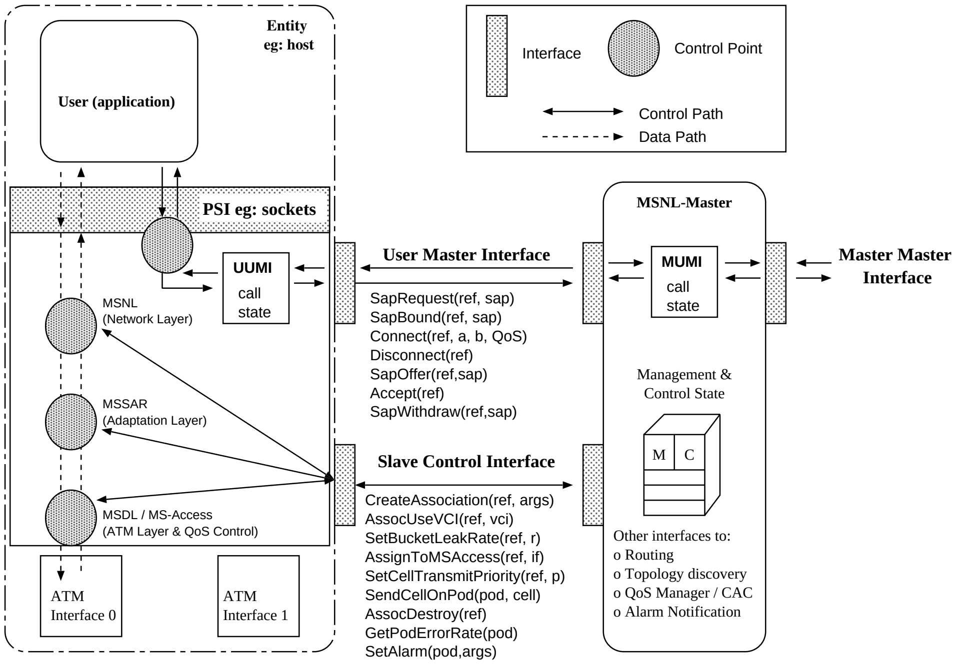


Figure 3.6: The MSNL Master and its interfaces to entities and the network

the MD then the MSNL Master for the MD is responsible for the setup of the entire data path. If the remote user is located on an entity external to the MD, then the MMI will be used to set up the elements of the data path which traverse other MDs. When a MD is traversed by a liaison which is not terminated in one of its entities then (for that MD) the liaison is a forwarding liaison. Such liaisons are implemented by building a data path (using the SCI) which ensures that data transmitted over the liaison will be forwarded transparently through the entities in the MD. The MSNL Master is free to optimise the process of building connections, for example by caching liaison segments or using virtual paths.

Figure 3.6 depicts a single entity and its MSNL Master. The interfaces exported by the entity and the master are shown, together with examples of typical operations on each interface (although the directions are not shown). User requests (from the application) on the PSI are translated into UMI signals. On the user side of the UMI (UUMI) and master side (MUMI) the state of each call is maintained. Not shown are interfaces to other services used by the MSNL Master, which are described in Section 3.4.7.

3.4.6 In Band Control

Critical to the success of a multi-service network is its ability to deliver guaranteed QoS to connections. To do so requires IB control mechanisms which, in MSNA, are located in the MS-Access layer. During connection setup the MSNL Master assigns each association to an MS-Access instance which implements the appropriate IB mechanisms. As an example, a node at which a connection enters the MD may be instructed to implement a leaky bucket policer to protect the MD from badly behaved users. Depending on the QoS required by a connection, it will also need to be given a service allocation and buffer constraint at each node.

A wide range of IB control mechanisms has been identified in the literature, and a brief introduction to the issues was given in Section 1.3.3. One of the main aims of the MSN-CMA is to support experimental work on performance of multi-service networks, requiring flexibility in the control plane. OB control mechanisms are located external to each entity, but IB controls must be located in the data path, within each entity. A flexible mechanism is required to enable experimentation with a range of control policies, whilst providing a consistent MS-Access interface to MSDL. IB control policies are not static, since they must react dynamically to changing network load on the burst and cell time scales. They must also evolve on a slower time scale as connections are set up and cleared.

In the MSN-CMA, the functions of MS-Access are abstracted by a single object, named the Cell Level Scheduler (CLS) which implements all IB controls at a given pod. The CLS ensures that all associations routed through a pod are allocated sufficient resources both for reception and transmission, and has access points in the control, management and user planes. In the user plane the CLS presents two primitives to MSDL: `request()`, to send a cell, and `indication()`, to indicate reception of a cell. These may be partially or wholly implemented in hardware, for example the automatic enqueueing, VCI translation and transmission of cells forwarded through a switch. In the control plane the CLS permits measurement of performance and resource usage, and dynamic control on two time scales. Whilst the CLS described here aims to be flexible, all ATM equipment will offer a low-level control interface which "narrows" the definition of the CLS, pictured in Figure 3.7 and described below.

Each CLS incorporates a number of named IB control *policies*, any one of which may be operational at a time. Each policy implements a queueing mechanism for cells routed through it, and a scheduling mechanism for selecting which cell to transmit next. These are expressed in the form of a set of named *methods* which implement the policy. Since the requirements of policies will differ widely; the CLS definition is as general as possible, however several features are included

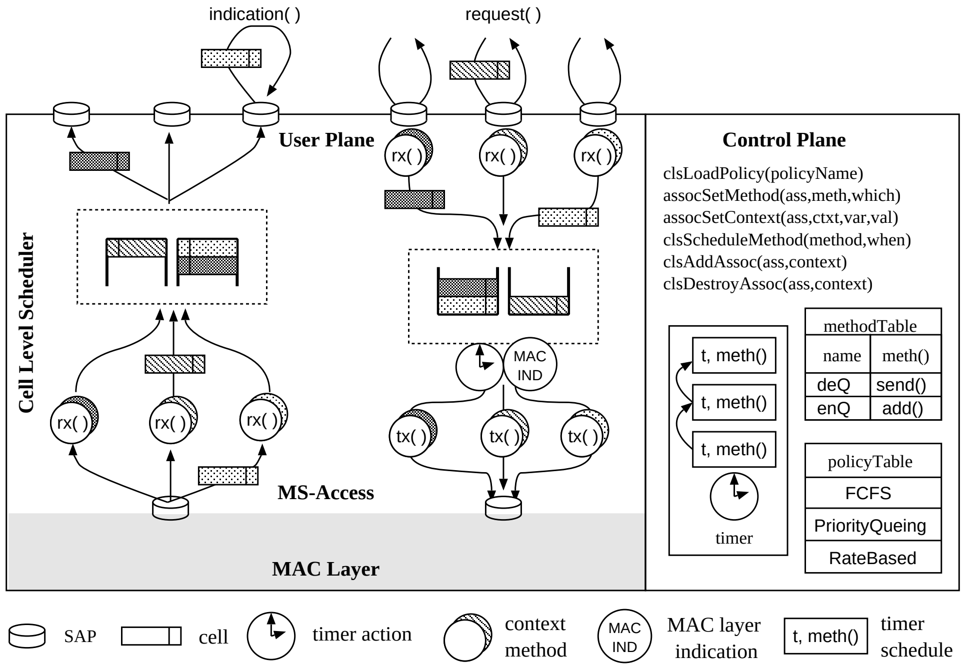


Figure 3.7: The cell level scheduler, with 3 MSDL users

to assist policy implementation. The high resolution timer is an object which ticks at some pre-defined rate. It can be set up to execute a named method from its execution schedule at a specified future time. The timer may be used to schedule transmission from the CLS, or for monitoring and statistical sampling.

The CLS maintains a context for each direction of transmission for each association routed through it. The context can be accessed using the VCI allocated to the association, and stores association-specific requirements used by the policy. Examples of context information include the cell loss priority for the association and its buffer allocation. In addition the context specifies two methods, `rx()` and `tx()`, which represent the IB control points within MS-Access for each association. Figure 3.7 shows a CLS serving 3 associations, each of whose cells and methods are shaded differently. The `rx()` method is invoked when a cell from the association is received by the CLS as a result of the user (or MAC layer) invoking its `request()` primitive, and `tx()` is invoked when a cell is transmitted by the CLS.

Each policy defines a subtype of the context which may support any number of additional methods and associated state, specific to the policy. Examples include `queue_overflow()`, invoked when an association exceeds its buffer limit or `rate_violation()` which implements the appropriate control function when the transmission rate is exceeded. The set of methods supported by a policy is named in its interface definition, allowing the MSNL Master to appropriately control the CLS. These methods may be used to refine the `rx()`, `tx()` and other methods, and may be scheduled on the CLS timer.

Contexts (and the policy state) may be refined dynamically on two time scales: in response to the instantaneous load and performance variations of the CLS, for example to begin cell discard when the buffer is full, and on a slower time scale, typically the call time scale, as contexts are created and destroyed by the MSNL Master for each connection. In the management plane each policy exports information which allows the MSNL Master to implement dynamic control on the call time scale, such as CAC functions. This takes the form of event notification and polled updates. The CLS design is similar to, but more sophisticated than the proposed approach of [Pacifici95] to dynamic control of multimedia networks, which uses a "legislator-executor" feedback-based model for dynamic control operating on a single, slow time scale.

Resource Management Flows

Draft ATM standards provide Resource Management (RM) flows for dynamic exchange of parameters for IB controls, such as rate and credit based flow control, between network elements and users. The MSNL Master is responsible for setting up these flows (for which code points are standardized) and parameterising their contexts in the CLS. For example an explicit rate specification may be communicated to the next switch on the path of a connection by a CLS context which updates the rate specification of an RM cell and forwards it to the next switch. Receipt of an RM cell may cause the context state and methods to be updated.

3.4.7 Other Control Plane Services

The previous subsections have described the mechanisms whereby the MSNL Master constructs communication paths for its entities. To do so properly, several services are required within the MD, which may be implemented independently of the MSNL Master. They include:

- the resource allocation and monitoring function which manages resources in each entity and within the MD as a whole
- the routeing function which determines which route to use for the data path connecting two MSNL SAPs, and
- the topology determination management function which determines the topology of the networks to which the MD is attached.

Since many of these management and control services will be common to all MDs, an instance of each service is not required for each MD; a mechanism by which elements in each MD can access the required services will suffice.

3.4.7.1 Resource Control

The resource control service (named the CAC daemon) is responsible for connection admission and for monitoring the service delivered to connections. The CAC mechanism takes as parameters a link and a QoS specification, and decides whether or not a new call with the given QoS specification can be accommodated on the link, on the basis of its traffic parameters and QoS requirements. Many candidate CAC algorithms have been proposed in the literature, for example [Ahmadi90, Kelly91, Gibbens95]. Details of the CAC algorithm are beyond the scope of this dissertation; some initial work on CAC using Fairisle is reported in [Crosby95]. In addition to making the CAC decision, resource allocation decisions must be made at lower layers. Specific allocations of transmission resource and buffer space must be made within each CLS, according to the policy it implements. Correspondingly, policy-specific resource allocation functions must be implemented by the MSNL Master. These use the SCI to translate the allocations into specific mechanisms within each CLS.

3.4.7.2 Routeing

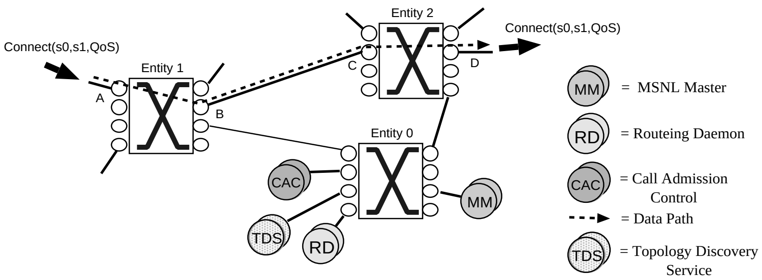


Figure 3.8: Routeing within the MD

In order to route requests between MDs, a routeing function is required which maps the logical topology of management domains onto the physical topology of the network. The Routeing Daemon (RD) must be aware of the structure (physical topology) of each MD which it serves, and obtains this information from the Topology Discovery Service (TDS), described in Section 3.5.2. The function of the RD is to take a tuple (s_0, s_1, QoS), where the s_i are SAPs, and return a route through the MD which joins them and which satisfies the QoS specification.

Figure 3.8 shows an MD and several of its services, including the RD. An incoming MMI connect request at port A on entity 1 must be translated into the physical route (A, B, C, D) through the MD. The RD makes use of the services of the CAC daemon to determine whether a candidate route under consideration will meet the QoS specification. The following subsection describes the steps involved in connection setup, in detail.

3.4.8 Connection Setup

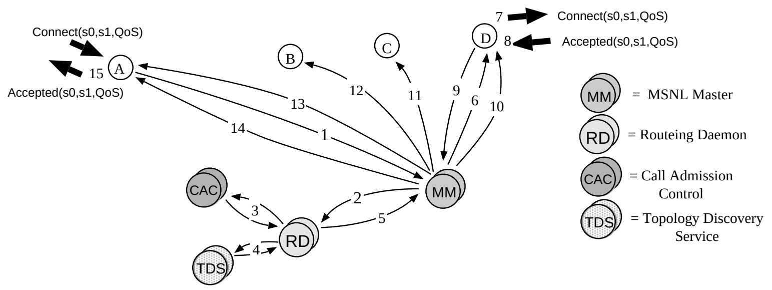


Figure 3.9: Steps involved in connection setup

Consider the topology shown in Figure 3.8. This section traces the steps involved in connection setup using the MSN-CMA, and should be compared with the description of the metasignalling mechanism described in Section 3.2 above. A full description of the UMI is given in Chapter 4.

Consider an incoming MMI connect request with arguments (s_0, s_1, QoS) at node A. The steps required for connection setup are shown in Figure 3.9. The entity controlling pod A terminates the MMI because it is at the edge of the MD, and forwards the MMI request to the MSNL Master (step 1). The MSNL Master requests the services of the RD (2) to route the request through the MD. The RD, in turn, uses the CAC daemon (3), and TDS (4), which stores the current network topology, returning to the MSNL Master the selected route (A, B, C, D) through the MD (5). The MSNL Master instructs node D to forward the MMI request (6) from its network interface (7), and reserves the requested resources (i.e. bandwidth and buffer space) for the connection temporarily. Some time later, if the connection is accepted by the destination $s1$, an acknowledgement is returned (8), which node D forwards back to the MSNL Master (9). At this point the MSNL Master confirms the resource reservation on the route (A, B, C, D), using the SCI to build the appropriate components in the user plane for the liaison, including the necessary VCI mappings (10, 11, 12, 13) in the nodes on the data path. Finally (14), the MSNL Master instructs node A to return an MMI acknowledgment to indicate that the connection has been accepted (15).

3.5 The Management Plane

The management plane implements control functions which operate over a time scale greater than that required for connection setup. This includes configuration management and the logging of errors and failures.

3.5.1 Alarm Notification

Entities in the network will occasionally detect error conditions on their links, or will fail to meet the QoS requirements of the connections routed through them. A mechanism is needed to inform the MSNL Master of these conditions, to permit appropriate control plane actions to be undertaken. These alarms are implemented by the Alarm Notification Service (ANS). The SCI includes operations which allow the MSNL Master to parameterise alarm reporting functions within each entity. When an alarm is set up, the MSNL Master provides the entity with an interface on which to report any alarms which are generated. This is the interface of the ANS. As an example, the MSNL Master might require notification of cell loss, which can be fed to the CAC daemon for use in acceptance decisions.

3.5.2 Topology Discovery

Several services, such as the routeing function, need to know the topology of the network and the MD. Determining the topology of the network is the task of the TDS. In order to perform its task the TDS requires management information from each entity. Each pod implements error detection capabilities which determine the status of its link. Periodically the TDS queries each entity to determine the status of its links, and to detect whether or not the entity itself has failed. If any element of the MD has failed then the TDS triggers a topology update. The topology discovery algorithm used in the Autonet [Schroeder90] could be adapted for use in this way. The TDS can also make use of the ANS for notification of link errors. The TDS feeds information to the MSNL Master via the latter's exception interface, to notify it of topology changes which require connections to be aborted.

3.5.3 Other Services

To fully address the communication needs of a multi-service environment, other services will also be required, for example in and out of band security features. Whilst these are not addressed in the current work, it should be possible to extend the architecture to include them. Two services which address an issue of practical importance, namely the procedure by which management domains are constructed and initialised, are described below.

3.5.3.1 Address Resolution

Initially, when the entities in the MD are powered up, no mechanism exists to enable them to communicate. One of the first things an entity must do is discover its own identity and then attempt to boot. The process of identity mapping is called the Reverse Address Resolution Protocol (RARP) [Comer91], and in MSNA is achieved using the metasignalling protocol. Thus, at the start-of-day, each entity must be able to metasignal[6]. A service not mentioned previously which is implemented by the MD is the RARP daemon, to which all RARP requests are forwarded. The RARP daemon maps the unique entity identifier into an MSNL address, which it returns to the entity (by metasignalling), together with the address of a boot service to which boot requests should be directed.

3.5.3.2 The Boot Service

The process of booting an MD is incremental. The MD is defined to contain all entities which have successfully booted and bound to the MSNL Master. An entity which receives a RARP reply connects to the boot service, which responds by returning a boot image and configuration information which includes an interface reference for its MSNL Master. The image also contains an application which imports the local SCI interface to allow it to metasignal. Using metasignalling the entity binds to its master, and permits the master to bind to its control interfaces. At this point the entity is considered to have joined the MD. The configuration of an MD may thus change dynamically; the MSNL Master consults a database which provides it with information about specific capabilities of each entity.

3.6 User and Network Services

This chapter has argued for the separation of service-specific aspects of control and the basic facilities required to implement communication. Several network services have been described, but little mention has been made of advanced services such as synchronisation and other desirable communication controls. It should be clear that the favoured approach is to locate these services outside the network. Some of them may, however, require privileged access to network control interfaces, for example, to control remote devices. Once the security issues have been addressed, the MSN-CMA offers a suitable way of incorporating new services: they are simply distributed programs and prospective clients

[6]The metasignalling protocol may be regarded as simply a primitive form of the MMI.

may bind to them in the usual way. Thus, support for architectures such as IMAC [Nicolaou91] is implicit in the MSN-CMA.

Third party services, such as a synchronisation service or other "provider" acting on behalf of a client, may need to be able to set up communications paths and perform manipulation both of data and control interfaces before presenting the data to the client. Such services are easily incorporated in the MSN-CMA: they may have privileged access to network control interfaces, and can "call back" a multimedia application awaiting the processed data on a SAP specified in the service request. This can also be used to provide sophisticated group management functions for multicast.

Finally, with regard to service standardisation, the ITU-T has long stressed the need for standardised interfaces to communications services. This chapter has already argued that this should not be achieved by forcing service-specific features into the network signalling protocols. Instead, if services are to be standardised then their procedural IDL *interfaces* should be standardised, together with a distributed name service such as the trader, which allows applications to find service offers which match their needs.

3.7 Summary

Based on a lightweight connection model, distributed control and RPC-based signalling, the MSN-CMA offers a simple management model for the multi-service network, with several advantages.

- Network control is located in a set of distributed programs which need not be located in the network.
- Only a simple user-network interface is supported. Sophisticated services may be built on top of the network.
- The distributed computing approach drastically simplifies service access: interface trading hides service location, and ensures that client and server are compatible. The use of RPC, with its well defined communication semantics, reduces implementation effort and errors.
- The simplified network interface is more easily specified, giving greater confidence of its correct operation.
- Multiplexed signalling is avoided, and different functional components of the architecture can be independently developed and tested.
- Both sophisticated and dumb network entities may be managed, and new equipment may be rapidly incorporated into the network.

The following chapter describes the MSN-CMA service interface, represented by the UMI, in more detail. A thorough specification of the interface is essential to guarantee its robust operation.

Chapter 4

The Specification and Verification of the User Master Interface

The architectural requirements of the MSN-CMA describe several important interfaces, including the UMI, MMI and SCI. Proper operation of the protocols defined by these interfaces is vital to the correct operation of the MSN-CMA. This chapter identifies the requirements for a full specification of the interfaces, using the UMI as an example. It also contrasts the MSN-CMA and UMI protocol with related control systems for communication networks, reported in the literature.

Although the MMI and SCI are not described in detail, their design follows the same principles as described below for the UMI. The UMI was chosen as an example because it represents the service interface of the MSN-CMA, and can be compared with other user-network protocols. The MMI may be implemented as a simple subset of the UMI, whilst the SCI is discussed in more detail in Chapter 5 which describes its implementation.

4.1 A Model of the UMI

The UMI defines the network service interface for the MSN-CMA, providing primitives for connection setup and teardown. The user of the UMI is a network entity, which signals requests on behalf of multiple applications (its users).

Figure 4.1 identifies two sides of the UMI, the User side (UUMI) and the Master side (MUMI). The UUMI responds to "internal" requests on the PSI from its attached users and to requests from the MUMI. In response to these requests it generates "internal" responses to the users on the PSI and makes requests to the MUMI. The MUMI responds to requests from the UUMI, from the network

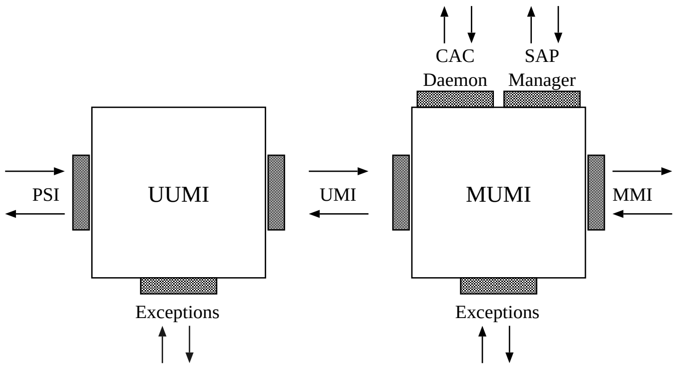


Figure 4.1: A model of the UMI

signalling interface represented by the MMI, and also to an interface which notifies the MSNL Master of exceptions conditions which require its attention. A similar interface is required by the UUMI. The MSNL Master also makes use of additional services to set up each connection. In the specification of the UMI the interaction of all of these interfaces is important, since actions on any of the interfaces may require action on the UMI.

Section 3.2 identified several failings of the MSNA metasignalling protocol resulting from inadequacies in its specification. The inadequacies result from a failure to take into account the complexity of the environment in which the protocol operates. Holtzmann [Holtzmann91] identifies five key aspects in the specification of a protocol, namely:

1. the definition of the *service* to be provided by the protocol,

2. the *assumptions* made about its environment,

3. the *encoding* of the messages used to implement the protocol,

4. the *vocabulary* of the messages, and

5. the *procedure rules* guarding the consistency of message exchanges.

The decision to use an RPC mechanism for signalling simplifies the description of the service interface and encoding of the vocabulary of the protocol, and directly influences the environment in which it operates. It is unable to assist the description of the procedures, for which an alternative notation is required. This makes it difficult to delimit cleanly the requirements identified by Holtzmann, however the description of the UMI protocol in following sections (with respect to the model of Figure 4.1) follows his approach.

The UMI is defined by the descriptions of the *service interfaces* exported by the user and MSNL Master, and by a state machine at each side of the interface. The operations in each service interface, together with their arguments, form the *vocabulary* of the protocol, and a set of transition rules associated with each state machine describe the protocol *procedures*. In the subsections which follow, elements of the vocabulary are denoted in `non-proportional` font, and refer to the notation used in Appendices A and B which present the UMI specification in detail. Each element in the vocabulary is given a type indicating its origin, shown by the first letter in its name: `e` for exceptions, `u` for UMI requests, and `m` for MMI requests.

4.2 Operational Environment

UMI requests are exchanged by means of RPCs. The RPC mechanism used is MSRPC2 with At Most Once (AMO) semantics, described in Section 2.2.6. When the MSNL Master resides on the same network entity as the user side of the UMI, MSRPC2 offers Exactly Once (EXO) semantics, and the procedures are a simple subset of the procedures for the AMO case.

In keeping with the goal of lightweight operation, MSNL is used for the RPC transport, with the consequence that requests or responses may be lost. The following causes of RPC failures are to be expected:

1. The request may be lost.
2. The response may be lost.
3. The server dealing with the request may be slow in responding to the request, particularly if it is overloaded.
4. The server may crash, losing all state.
5. A client may crash after issuing a request.

Causes 1, 2 and 3 require that all requests be encoded in an idempotent fashion so that they can be retransmitted. Detecting transmission failure requires that a requester maintain a timeout for every outstanding request. When the timeout expires, if response has not been received, the client can re-issue its request. Old and out-of-sequence responses are automatically detected and discarded by MSRPC2. Causes 4 and 5 additionally require a mechanism to detect failed components and procedures which ensure correct operation in the face of failure.

4.2.1 The Exception Interface

The exception interfaces in Figure 4.1 are used to deliver all error notifications to the UUMI and MUMI. A timeout raises the (`e_time_out`) exception, allowing a client to retransmit its request. More complicated faults may also occur at any time, including:

- receipt of data by an entity for a nonexistent association,
- failure of nodes or links on the data path, or of the entity itself,
- failure to deliver the requested QoS to a liaison.

The last condition is detected using the alarm notification service and falls into the management plane. The first two are control plane issues and are reported by delivering exceptions at the MUMI exception interface. In the first case, the entity raises an exception (`e_u_data`) via the ANS identifying the connection for which data is being received. The MSNL Master must take action to abort the rogue liaison. Failure of network components in the MD is detected in the management plane by the TDS which raises the (`e_u_failed`) and (`e_n_failed`) exceptions respectively, indicating failures in the `u`ser or `n`etwork links. The MUMI is also informed by the (`e_u_recovered`) and (`e_n_recovered`) exceptions when the components recover. One final condition exists: if a connection becomes idle due to component failure or simply because it has fallen out of use, then after some time, the (`e_idle`) exception is raised, allowing the MSNL Master to destroy the liaison if required, ensuring that its resources are reclaimed by the network.

4.2.2 Encoding of Requests

An advantage of using RPC is that it simplifies the encoding of the protocol vocabulary. Each operation in the interface corresponds to a "command" in the protocol. Each operation takes arguments which are encoded automatically. In MSRPC2 the encoding rules for MIDDL basic types are specified by the procedural mechanism of MSDR. Compound data types described in the interface are marshalled by the MSRPC2 stubs generated by the interface compiler. The mechanism is described in more detail in Section 2.2.6.

4.3 Requirements of the Service Interface

Many of the requirements of the MSN-CMA service interface are common to all high performance networks; a few are determined by the necessity to support experimental work, and the experimental nature of the equipment used. One of the main goals was to keep the service interface as lightweight as possible.

The UMI permits users and the MSNL Master to control calls. The *call reference* uniquely identifies each call and its associated state on both sides of the interface, and is passed as an argument to every RPC invocation. On the MMI the globally unique liaison identifier is used as the call reference.

4.3.1 The PSI

The PSI allows multiple applications supported by the entity to request MSNL services. The definition of the PSI is necessarily platform specific, for example the BSD socket interface. The platform translates calls across the PSI into UMI requests internally, and may use any useful value as the call reference, for example a reference to the protocol control block (PCB) in the kernel.

4.3.2 MSNL SAP Management

SAP management is implemented by the MUMI in the form of a SAP manager, to which all SAP requests must be directed (this service is typically tightly bound with the MSNL Master). The initiator of an MSNL connection is termed the active user; a user which offers a SAP for incoming connection requests is termed the passive user. Associated with the active user is an active SAP, whilst the SAP assigned to the passive user is termed the passive SAP. A SAP is not typed *per se* but becomes so when it is used for communication. A SAP which has been marked as passive cannot be used to initiate a connection request, and a single passive SAP may serve as the endpoint of multiple liaisons. A user may request a SAP by name (corresponding to a well known address) or may be content with any SAP. The SAP manager refuses to allocate a SAP if it is already in use or if the address space is exhausted.

4.3.2.1 Vocabulary

The UMI permits the user to request a SAP (`u_sap_req`), possibly by name. If available the MSNL Master returns) the sap, bound to the call reference (`u_sap_bound`), if unavailable it rejects the request (`u_sap_denied`). A passive user may offer its SAP for incoming connection requests (`u_sap_offer`). If successful the MSNL Master acknowledges the offer (`u_sap_offered`). The passive user may withdraw the offered SAP at any time (`u_sap_withdraw`). This is acknowledged with (`u_sap_bound`) as the SAP moves back to the bound state. A SAP remains bound to its call reference until the call reference is eventually destroyed. The procedures are shown in Figure 4.2. Having been granted a SAP the active user can also use it to connect to a passive SAP. This is described below.

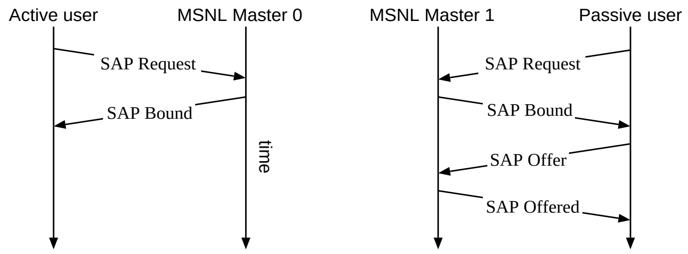


Figure 4.2: Procedures for SAP allocation

4.3.3 Connection Services

A range of modes of communication have been proposed in the literature, including point-to-point, point-to-multipoint, and multipoint-to-multipoint communication. Support for point-to-point connections, incorporating support for full duplex communication between two endpoints, is a basic minimum.

4.3.3.1 Multicast

Multicast, in the form of multi-party communication, is necessary for distributive services, but the best way to implement it is not clear. Nor, indeed, are the semantics of the communication – if all parties are connected to one another via a single connection, how does a receiver distinguish messages received from two different senders? Multicast connections are complicated to set up, administer and control because they involve the determination and maintenance of the multicast routeing tree [Doar93], membership eligibility tests, and require primitives to support solicited and unsolicited addition of new parties to the communication [Bubenick92].

4.3.3.2 Fast Reservation

Another possible feature is fast reservation, which allows a source to request resources for each traffic burst, waiting for permission from the network, and delaying transmission by a round-trip time [Doshi90]. Alternatively the source may not wait for acknowledgement; the network simply discards a burst for which it has insufficient capacity [Roberts92, Boyer90]. A study by the author [Crosby93], found that the gain in link utilisation resulting from renegotiation is insufficient to warrant its implementation.

4.3.3.3 A Simple Approach

Other authors have addressed the grouping of related connections into calls and the composition of basic call components into sophisticated objects [Minzer91, Heijenk94]. The MSN-CMA recognises the need for sophisticated services, but adopts a simplified approach, providing only a point-to-point connection service, whose basic element is a liaison, uniquely defined by a pair of SAPs and a QoS specification. More sophisticated connection services can be built on top of the basic call component, by implementing additional connection management services.

In order to support external support for point-to-multipoint multicast (by a multicast service)[1], the notion of a multicast group is introduced. A liaison may optionally be assigned to a multicast group; all liaisons in the same group share the *same* multicast group identifier, which identifies the owner of the group. Higher layer protocols must be used to publicise the identifier and to allow users to subscribe to the group. The multicast service within the MD manages requests for addition and deletion of liaisons to and from a group. By definition, a liaison assigned to a multicast group is simplex (i.e. distributive), with communication originating at the root and propagating to the leaves.

4.3.4 Resource Allocation

The metasignalling protocol described in Section 3.2 is a two-phase protocol: the active user issues a CONNECT message, and the passive user returns an ACCEPT message, moving the connection into an active state. On the forward path the gateways involved construct association records and record all state required by the connection. This has the advantage that the passive user can commence transmission immediately it accepts the connection, but it also has two associated drawbacks: first, if the accept message is lost then the network must deal with the rogue connection and its data. Second, if the passive user rejects the connection then the allocated resources must be freed again. The time consuming allocation process on the forward path could have been avoided, and the passive user's decision could be determined more quickly if the resource allocation were performed after the user accepted the call. Both to enable this, and for enhanced reliability, the UMI uses a (modified) three-phase signalling mechanism for connection setup. Other interactions, including SAP management, are two-phase.

Stallings [Stallings89] identifies the need for a three-way handshake protocol when unreliability of the signalling network can cause confusion as to the state of

[1] The design of the multicast service is due to Shaw Chuang of the Computer Laboratory, and will be described in detail in [Chuang95].

a connection. In this case the use of RPC to implement the connection set up is unreliable because the MSNL Master has no way of knowing that the active user received its reply to the connect request. In order to guarantee consistent state at the MSNL Master (and passive user) a third phase is used. On the forward path (the connect request) each MSNL Master reserves resources but does not create any association state. As the acceptance returns back across the network, each MSNL Master builds the appropriate data structures in the entities on the path. When the accept response reaches the active user, it may commence data transfer immediately, but an additional acknowledgement is also returned across the UMI to the MSNL Master (out of band) to indicate that the connection is complete. At the passive user either the receipt of data or of the final acknowledgement will suffice to complete the connection.

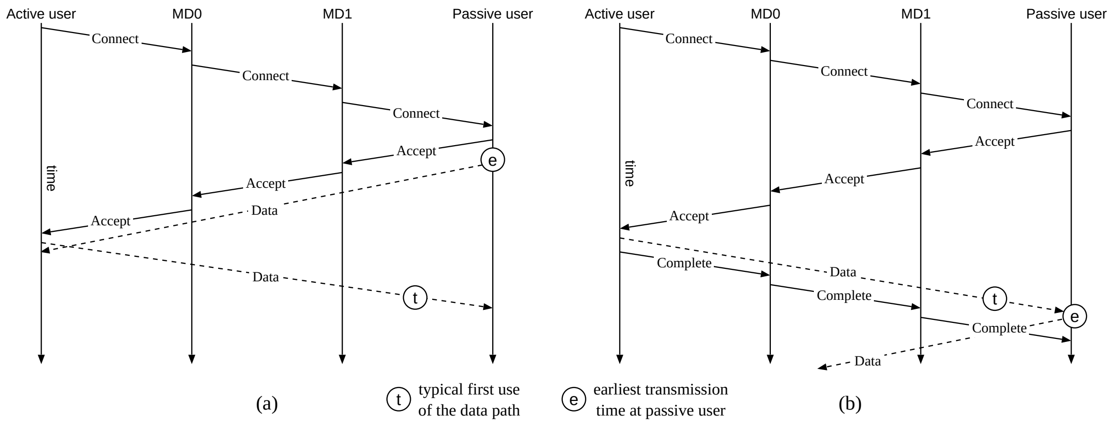


Figure 4.3: (a) Two-phase versus (b) modified three-phase connection setup. The UMI uses approach (b)

The passive user may not commence transmission until it knows the data path has been built. This restriction is not considered important because the bulk of passive users will be servers which, after accepting a connection, await service requests from the client rather than immediately commencing transmission. The advantages of the approach are its greater reliability and consistency. Figure 4.3 shows connection setup using both mechanisms, with the time of the typical first use of the data path in a client-server paradigm, and the earliest time at which the passive user may transmit. Note that in mechanism (b), used by the UMI, either the receipt of the final complete message or receipt of data moves the passive user into a fully connected state. Connection setup latency for the typical path should not be degraded.

4.3.4.1 Vocabulary

An active user may request the MSNL Master to connect (`u_connect`) it to a remote SAP. Arguments include the active and passive SAPs and the requested QoS. If the MSNL Master is able to make the connection, the (`u_connect`) request will be presented to the passive user. Both the user platform and the application (PSI) end user are given the opportunity to accept (`u_accepted`) or reject (`u_rejected`) the request. In the first case, a modified QoS specification is returned to the active user indicating the allocated QoS. If the connection is rejected by the passive user or the MSNL Master, the returned arguments include the reason and a modified QoS specification indicating the available QoS.

The active user, on receipt of the acceptance, may commence transmission, and issues the (`u_complete`) notification in the control plane. The passive user may commence transmission either on receipt of data or the complete message. At any time, either user may issue a (`u_disconnect`) request to end communication. When the connection has been destroyed, the MSNL Master returns a (`u_idle`) response. When the user application finally releases its call reference, the user platform issues the (`u_final`) request; the MUMI destroys the call reference and returns a (`u_final`) response.

4.3.5 User to User Signalling

The UMI and MMI permit users to signal to one another out of band using user-user signalling requests across the interface using a (`u_data`) request. The signalling mechanism is, in the spirit of MSNA, unreliable, and a message is only delivered to its intended recipient if the liaison to which the message refers is in a fully connected state.

4.3.6 Errors and Failures

Apart from the exceptions resulting from the unreliable communication mechanism, several error conditions must be taken into account in the UMI itself. The MSNL Master may unilaterally decide to (`u_abort`) a connection at any time. The platform must inform its user and respond with a (`u_aborted`) reply, immediately destroying the call reference. Similarly, either the active or passive user *application* (i.e. the PSI user) may be halted at any time by the platform (for example due to a bus fault). The platform supporting the user discards the call reference with a (`u_final`) request, to which the MUMI responds with a (`u_final`) response.

4.4 Specification of the Call Procedures

The previous sections have described the UMI with regard to the first four of Holtzmann's requirements for a protocol specification. The *call procedures* identify all valid sequences of state transitions at the UUMI and MUMI in response to inputs from the interface and environment.

Interface description languages such as MIDDL, ANSA IDL [ANSA89], CORBA IDL [OMG91] and the OSF DCE [OSF92] permit the definition of data types which cross the interface, operations which are offered by the server, and exceptions which may be raised. Although use of an IDL simplifies the protocol definition, all IDLs suffer the drawback that they do not permit the specification of the *operational semantics* of the interface. The operational semantics describe the set of valid state transitions on either side of the interface and their detailed specification is vital to ensure robust and correct operation of the protocol.

4.4.1 Formal Description Techniques

Initially two state tables were used to specify the UMI, one for each side of the interface. Each table entry specified an input token, current state, next state and output token. When the exception interfaces, PSI and MMI were included, the state tables became too unwieldy to comprehend, and a more concise notation was sought.

A number of formal description techniques (FDTs) have been developed which allow the specification (and testing) of protocols. The ITU-T has standardized the Specification and Description Language (SDL) [ITU-T88] for specification of signalling protocols. This has an associated graphical notation which may be used to illustrate protocol operation. Although it has been used to specify several large protocols, SDL permits the specification of infinite systems, which renders the verification of the specification formally undecidable in some cases. Few case studies of protocol verification using SDL have been reported. Conversion of SDL specifications to working code is awkward, with the result that its use is impractical. Two other formal description techniques, Lotos and Estelle, have also been standardised by the ISO, and are discussed in [Holtzmann91].

Holzmann's Promela [Holtzmann91] permits the description of so-called validation models of concurrent systems. Promela provides a notation, in the form of a programming language, with constructs to specify processes, their interaction via message channels and their modification of state variables. A Promela program may be executed (validated) using the SPIN validator to prove properties about its temporal correctness. The notation is aimed at validation and deliberately abstracts away the encoding, vocabulary and communication mechanisms used by the protocol. It is not aimed at assisting implementation.

In the case of the MSN-CMA protocols, which use an RPC mechanism with well understood semantics, it is desirable to specify the vocabulary and state changes using a formal notation which can be *automatically translated* into the programming language representation of the RPC system. Furthermore, just as distributed network services are *composed* to deliver the full functionality of the MSN-CMA, the ideal FDT would permit the composition of separately specified interfaces. This functionality is provided by the FDT of choice, namely the *Temporal Logic of Transitions* (TLT), a general framework for the design and verification of distributed systems, developed at Siemens [Cuéllar94a].

4.4.2 TLT

The design of TLT is centred around three views: a *user* view (called U-TLT), consisting of a guarded-command programming language for the specification of concurrent systems and their environments, a *logical* view (A-TLT), consisting of a temporal logic into which U-TLT programs are translated and in which their properties are formulated, and a *semantical* view (C-TLT), providing a suitable semantic interpretation for A-TLT formulas. Using these views, system development techniques such as refinement and composition are supported formally, and verification methods such as theorem proving and model checking are available.

The U-TLT program represents a clear and unambiguous specification of the protocol being modelled. The language provides a full programming interface in which predicate calculus operators are used to specify the *guard* of each *instruction*. The guard specifies conditions under which the instruction may be enabled. The instruction modifies program variables and a so-called primed state variable which becomes the input to the next iteration. At each stage every guarded instruction is considered and the choice between enabled instructions is arbitrary. Instructions may be coded using relational, conditional and arithmetic operators. An example is given in Section 4.4.3 below.

A useful feature of U-TLT is that the programming language permits arbitrary user defined, typed state variables to be used. Furthermore, given the specification of the protocol in U-TLT, a translation of the vocabulary into an RPC-based, programming language interface is reasonably straightforward, and can be achieved automatically. This is has recently been demonstrated by the TLT developers at Siemens. The principal reason for the choice of TLT, however, is that it permits compositional verification of large, modular systems. In the MSN-CMA a large number of sub-components, each with their own interfaces, are composed to form a single control system. TLT permits the independent verification of each subsystem. The subsystems can then be composed and the full system verified as a result. No other verification systems of which the author is aware possess this facility.

```
% Call state = STATE_U_FINAL (no call) and UMI input = U_SAP_REQUEST
```

$[]\quad \mathit{call} = \mathit{state\_u\_final} \wedge \mathit{rx\_umi} = \mathit{u\_sap\_req} \longrightarrow$

$\quad$ (IF ($\mathit{sap\_manager} = \mathit{ok}$) THEN

$\quad\quad$ ($\mathit{call'} = \mathit{state\_u\_sap\_bound} \parallel \mathit{rx\_umi'} = \mathit{u\_null} \parallel \mathit{tx\_umi'} = \mathit{u\_sap\_bound}$

$\quad\quad \parallel \mathit{sap'} = \mathit{bound} \parallel \mathit{tx\_exc'} = \mathit{e\_set\_timer} \parallel \mathit{time\_out\_ctr'} = \mathit{0}$)

$\quad$ ELSE

$\quad\quad$ ($\mathit{call'} = \mathit{state\_u\_final} \parallel \mathit{rx\_umi'} = \mathit{u\_null} \parallel \mathit{tx\_umi'} = \mathit{u\_sap\_denied}$))

Figure 4.4: The TLT instruction for SAP allocation at the MUMI

4.4.3 TLT Specification of the UMI

Each connection is assigned a unique call reference, which identifies all call state on both sides of the UMI. This means that it is sufficient to model only a single user and call because calls are independent. The TLT specification of the UMI can thus be broken down into separate user (UUMI) and master (MUMI) programs with appropriate environments. To be robust, the specification must include a guard for all possible inputs to every state of the state machine. In TLT the environment of each component (UUMI and MUMI) generates every possible input at each step. This exhaustive approach includes "normal" call sequences such as those described above for SAP acquisition, call setup and teardown, and also all "erroneous" sequences, resulting from the AMO RPC semantics.

The TLT design of the UMI consists of two U-TLT *programs*, the UUMI and MUMI, and reflects closely the model of Figure 4.1. Each program has an associated environment, and three dedicated interfaces. The UUMI has interfaces to the user platform (PSI), an exception interface (EXC), and the master (UMI). The MUMI has interfaces to other masters (MMI), the user (UMI) and an exception interface.

The state of the SAP associated with a call is modelled by a state variable which defines the current type of the SAP attached to a call. The valid SAP status values are drawn from the set { `null, bound, active, passive` }. Both the SAP manager and CAC daemon are modelled by state variables which choose their values nondeterministically at each step from the set { `ok, not_ok` } modelling all possible outcomes to requests for SAPs and resources respectively[2]. Figure 4.4 illustrates the SAP allocation process, showing a single instruction from the MUMI program. The initial call state is *state_u_final* (no state); an arriving SAP request is granted if the SAP manager allows it, and the call state *call'* moves into the state *state_u_sap_bound*. Simultaneously (||) the `u_sap_bound`

[2]This neatly avoids the problem of a potentially infinite state space.

response is returned to the user, the SAP becomes bound, and the timer is reset with an infinite timeout (0). If the request is rejected, the `u_sap_denied` response is returned, and the call state is not initialised.

Each interface contains input and output buffers which model RPC operations with asynchronous message passing. The UUMI and MUMI programs are composed to model the whole protocol using a *connector*. Full details of the model are given in [Cuéllar94b], and a related paper [Barnard95]. A full TLT specification of the MUMI program is given in Appendix B.

4.4.4 Verification of the Design

TLT includes a toolbox which enables rapid verification of TLT programs. The TLT specification of the UMI was verified using these tools by Dieter Barnard of Siemens Corporate Research and Development. Details of the TLT design and a proof of the correct operation of the UMI are presented in Appendix A. The proof showed that several important properties hold:

- Livelock free: This is the most important property, and shows that the interface does not contain any non-progress cycles. All valid cyclic behaviour was identified, and more importantly, two non-progress cycles[3] were found and corrected.

- Link failure: if a transmission link fails during connection setup, an exception is raised, and the call state is cleared. Additional calls are rejected until the link status returns to normal.

- Correct SAP management: The MSNL Master will always grant a SAP if there are SAPs available. An active user never uses a passive SAP, and when data transfer takes place, the user's SAP must be either passive or active, and not just bound, or even worse, null. All SAPs are eventually freed (there is no resource leakage).

- All calls which are initiated are eventually cleared. All calls which are initiated are either successfully connected or rejected.

It should be noted that the initial FDT used for the UMI, namely state tables, was unable to succinctly express the multiple inputs into the system and the nondeterminism of the environment.

[3] Bugs in the specification.

4.4.5 Concluding Remarks

The choice of a simple service interface for the MSN-CMA has numerous advantages, including the ease of specification and verification of the interface. This has also been termed the property of *finite completeness* [Miller92b]: because the protocol service interface offers a finite number of connection possibilities, the task of writing the protocol itself is finite. In contrast, ITU-T protocols such as Q.2931 (see below) may never completely stabilize.

4.5 Related Work

The MSN-CMA, described in Chapter 3 facilitates distributed control of a multiservice network. Its service interface is represented by the UMI protocol defined in the preceding sections. This section reviews in some detail several alternative architectures for control of communication networks. Some define only a signalling protocol rather than a full control and management architecture. Where appropriate they are contrasted with the MSN-CMA.

4.5.1 ITU-T and ATM Forum Approaches

Many ITU-T signalling protocols have developed from an era when the customer premises equipment was far less intelligent than the network elements. The desire for a simple user-network interface led to the requirement for a limited set of protocols which express all service features. This assumption no longer holds but unfortunately the ITU-T approach to signalling for the B-ISDN has not fully recognised this. Section 1.3.2 described their approach in some detail. The ITU-T ATM UNI protocol was initially based on Recommendation Q.931, also known as the Digital Subscriber Signalling System 1 (DSS 1) used for ISDN signalling [ITU-T94b]. The call procedures in this protocol are inadequate for ATM networks, however, with the result that Recommendation Q.2931 [ITU-T94a], and extensions, also known as DSS 2, are being developed. The ATM Forum UNI [ATMF94] is a similar protocol aligned as far as possible with DSS 2, but with some simplified procedures, and a broader set of connection services. Details of the differences between the two can be found in [Stiller95]. Both are based on message exchange on top of a reliable signalling adaptation layer, Q.SAAL. Messages contain commands, and compulsory and optional, variable length information elements. An example of a compulsory information element in a CONNECT message is the call reference. An optional information element might include AAL parameters or a QoS specification.

The ITU-T approach to standardisation for B-ISDN is based on the successive development of three so-called Capability Sets, each of which defines a set of services which the protocols implement. The first set supports only simple point to point connections; resources are allocated using the connection peak rate. Capability Set 2 will support point-to-multipoint connections and virtual paths, multi-connection calls, bandwidth negotiation and possibly renegotiation, and VBR services. In Capability Set 3, additional support will be provided for broadcast connections and other services which have yet to be identified.

The encoding rules for signalling messages are specified individually for each ITU-T protocol, rather than using a set of standardised procedural rules, such as ASN.1 [Rose91] or MSDR. The encoding is often inefficient, requiring bit-wise operations on arbitrary sub-parts of data words. Problems are also caused by the facility for including optional service-specific information elements within each message – the protocol behaviour under each combination of elements should be specified but frequently is not, causing ambiguity[4]. Further drawbacks include the preservation of archaic features from earlier protocols. The reliance on message passing precludes the use of client-server based approaches to signalling and network services, and requires that each network element is sufficiently intelligent to parse and act upon each type of message. In summary, the approach fails to capitalise on advances in high level service architectures.

The network management protocols standardized by the ITU-T are based on a layered network architecture, known as the Telecommunications Management Network (TMN) [ITU-T92b, ITU-T92c], which overlays a management network on top of the underlying transmission network topology. Management protocols are based on the use of CMIP/CMIS [Rose91] (the OSI management protocols), which are crude in the sense that they permit only basic query and response type interaction with managed objects. The ATM Forum approach to management follows the use of the Simple Network Management Protocol (SNMP), which is also a primitive query-response protocol which interacts with ATM equipment via a standardized Management Information Base (MIB). In contrast the MSN-CMA permits the definition of procedural management functions using RPC.

4.5.2 The Generic Signalling Protocol

Recognising the flaws in current telecommunication network control architectures, Miller and Turcu have proposed the Generic Signalling Protocol (GSP) [Miller92a, Miller92b]. GSP, like the MSN-CMA, is simple and allows the construction of network services from basic elements. It is object based, and aims to incorporate a diverse set of equipment and network types. It defines four functional elements of a communication network, namely access, mediator, switching and transport entities, which are pictured in Figure 4.5. Transport entities are responsible for

[4] A SETUP message, for example, contains 4 mandatory and up to 18 optional elements

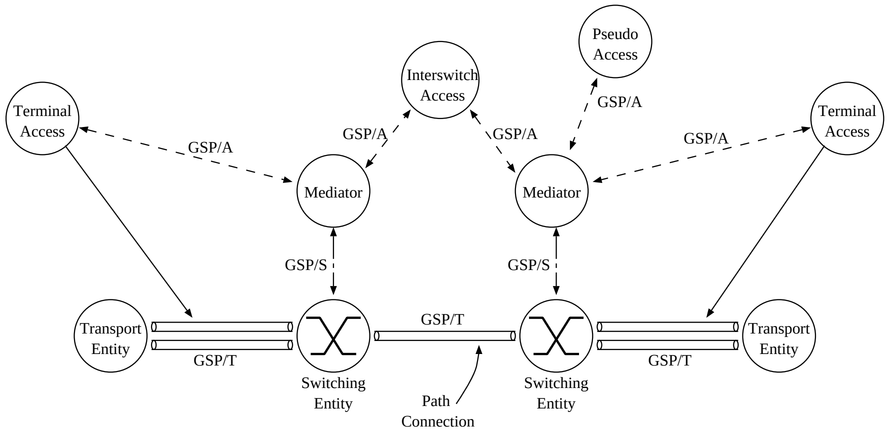


Figure 4.5: The GSP Architecture

terminating path connections at a switching entity. Path connections are used to transfer data. A switching entity connects together path connections using the GSP for transport (GSP/T) protocol subset. A user signals its requests for path connections to a mediator, using the GSP for access (GSP/A) protocol subset. The mediator uses the GSP for switching (GSP/S) protocol to instruct a switching entity under its control to set up the requested communication paths. Inter-switch access is also managed using GSP/A, with an inter-switch access entity relaying signals. Proxy signalling is supported via the pseudo-access mechanism.

GSP/A provides only seven basic connection facilities: primitives include OFFER, ACCEPT, CLEAR, ATTACH, ALSO_ATTACH, SUSPEND, RESUME, and user-to-user out of band data signalling. Several features of GSP are similar to those of the MSN-CMA: it offers simple control functions, proxy signalling, and the notion of a mediator (similar to the MSNL Master). It differs though, in several important aspects.

- GSP assumes a reliable transport protocol for the transmission of signalling messages, whereas the MSN-CMA uses an unreliable RPC mechanism.

- GSP supports SUSPEND and RESUME operations on connections – this is counter to the philosophy of lightweight connections which is central to MSNA and which favours instead support for rapid establishment and teardown of connections.

- Multipoint communication is supported using the ALSO_ATTACH primitive, whereas in the MSN-CMA this is supported by building the multicast service on top of the lightweight connection service.

- Each switching entity is managed by a single mediator, whereas the concept of a management domain in the MSN-CMA permits multiple switching entities to be managed by a single master.

- The connection setup mechanism requires the exchange of four messages across the interface before data transmission can commence, whereas the UMI uses a modified three-phase protocol.

- The signalling mechanism in GSP is based on command-response principles, but does not use RPC. The mechanism used to signal multiple requests simultaneously is to batch them together into a single multi-request message. The MSN-CMA permits multiple RPC requests to be simultaneously outstanding on a single interface.

An important similarity to the MSN-CMA is that both exhibit the property of finite completeness: since the protocol service interface offers a finite number of connection possibilities, the task of writing the protocol itself is finite. In contrast ITU-T protocols such as Q.2931 may never completely stabilize – each new service requires an additional protocol element.

4.5.3 The Telecommunications Information Networking Architecture

Recently the telecommunications industry, particularly carriers, has begun to realize that a more flexible approach is required for network management and control. The TINA Consortium (TINA-C) was set up with the aim of defining and validating an "open" architecture for telecommunications services in a broadband environment, whilst preserving inter-operability with installed equipment [Nilsson95].

The TINA architecture is based on well understood principles of distributed computing, and has built on the CORBA [OMG91], ANSA [ANSA89] and ODP [ITU-T92a] principles, whilst preserving compatibility with ITU-T standards. The architecture is broken down into four components: the *computing, service, management* and *network architectures.*

The *computing architecture* uses the OSF Distributed Computing Environment (DCE) [OSF92] to define an object-based model for computation, communications services and management. The architecture hides from applications the underlying details of network technology and distribution concerns. Communication is RPC based, and services such as trading, federation and notification are supported.

The *service architecture* provides a mechanism for specifying, analyzing, designing and operating service-related telecommunications software components.

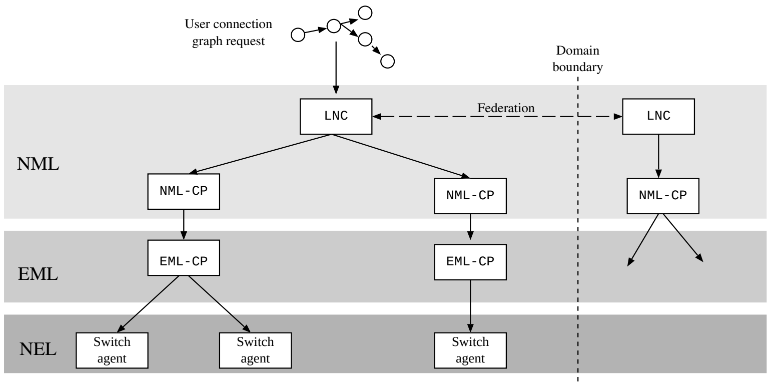


Figure 4.6: The TINA Connection Management Architecture

It defines the software components used by all services, providing modules of re-usable code for user and terminal agents, service sessions and session managers. User agents represent users in the network, and may move with them.

The *management architecture* provides a set of management principles for each of the other architectures. The architecture distinguishes between the management of computing resources and telecommunications management, which is responsible for management of network elements, and uses ITU-T principles (and protocols).

The *network architecture* is of most interest in this context. It provides a set of generic components for network resources, such as connections, used by service applications. It is based on the principles of ITU-T Recommendation G.803, which identifies layered components of a transport network according to their functions, abstracting away technology specific aspects, and defining technology independent control and management functions. The user's view of a network is encapsulated in a *connection graph* which links vertices representing ports, using edges representing connectivity. Services request a *communication session manager* to build a physical connection graph, given a logical connection graph defining the needs of the service. The *communication session manager* interacts with the connection management functions to establish and monitor the physical connections.

The TINA-C connection management approach is hierarchical, as shown in Figure 4.6: a user presents a connection request, in the form of a connection graph, to a Layer Network Coordinator (LNC), which is responsible for setting up the path within a particular layer network. Federation of LNCs permits a layer network to span multiple domains, and allows connection establishment in parallel in different domains. Each LNC delegates responsibility for sections of the path to Network Management Layer Connection Producers (NML-CPs), one per (sub)network, which in turn instruct Element Management Layer CPs (EML-CPs) to build connection components within the elements they control, at the Network Element Layer (NEL). Any sub-part of the hierarchical structure may be located on a single network element, or the entire hierarchy may be centralized.

The TINA architecture is similar to the MSN-CMA in that it removes complex control functions from network elements. The hierarchical control structure allows either centralised or distributed management of the network, according to requirements. It also follows a distributed systems approach. The TINA service architecture is far more feature-rich than the MSN-CMA, and facilitates mobility, which is not incorporated in the MSN-CMA. At a low level TINA remains compatible with existing control and management standards. This is unfortunate, since it is precisely the problems which they cause that it is designed to overcome. Also TINA-C has yet to address the problem of interaction with "dumb" multimedia devices, which are implicitly supported by the MSN-CMA.

4.5.4 Connection Management Access Protocol

The CMAP proposal [Bubenick91, Bubenick92] is an ATM user-network signalling protocol featuring a rich set of signalling primitives: point-to-point, point-to-multipoint, and multipoint-to-multipoint calls can be set up, managed and monitored. It is a UNI protocol in the sense that it only defines the primitives used between users and the network; within the network a second protocol, the Connection Management Network Protocol (CMNP) is used. The latter has yet to be specified. CMAP was submitted to the ATM Forum in an (unsuccessful) attempt to avert the adoption of the ITU-T approach to ATM signalling.

Calls are distributed objects and may comprise many connections, which can be dynamically added or removed. Additional endpoints may be added either by the owner of the call, by explicit request of the new party, or by third party request. Security features, in the form of access control and ownership are built into the protocol. Most procedures use two-phase signalling, but all network-initiated communication is three-phase. Owners are requested for permission for modifications to call membership. Because it is ATM-based, CMAP includes specific parameters for bandwidth and QoS specification, as well as a best effort service. Connections are either single or full duplex, and may optionally "echo" back to the sender. The communication mechanism is based on message passing using a metasignalling approach, and requires reliable transmission between user and network. This should be contrasted with the unreliable RPC mechanism used by the MSN-CMA. Proxy signalling is also supported. Parameters such as QoS requirements may be negotiated between user and network.

The CMAP specification is a complete statement of the requirements of the service requirements of multimedia applications, and may be used in both the local and wide area. A total of 38 primitives is defined for use between network and user, and although these seem sufficient to implement most functions required of current networks, the large size of the protocol makes it difficult to implement, and, more seriously, difficult to be certain that it operates correctly. In this sense CMAP differs from the MSN-CMA, which offers a simplified service interface which can be demonstrated to operate correctly, but which allows sophisticated services to be constructed on top of it. No attempt, of which the author is aware, has been made to verify the correctness of the CMAP protocol, and no reports of its implementation have been made.

4.5.5 EXPRESS

The EXPRESS architecture [Minzer91] is similar to CMAP in its service interface, but offers an object-based approach to call construction: call components can be treated as building blocks of more sophisticated connection services and thus the architecture is more open than CMAP. Applications formulate service requests in the form of (atomic) extended transactions, and operations in a transaction can involve multiple users. Within the network each user is represented by its User Request Manager. A Derived Service Manager examines related basic call objects and instructs a Transport Resource Manager to build the appropriate communications paths. The network service interface here is comprised of the highly expressive and large set of operations which users can invoke. This has the drawback that it is difficult to specify and verify. Unlike the MSN-CMA, this work only addresses signalling aspects, and fails to address wider architectural issues regarding network control and service implementation.

4.5.6 The Multimedia Binding Architecture (MBA)

Recognizing that more than a signalling protocol is required to adequately support multimedia services, Lazar *et al.* [Lazar94] have recently proposed a binding architecture based on the use of the CORBA [OMG91] distributed environment. Similar to the MSN-CMA, the MBA is open in the sense that it provides building blocks for higher level services. Communication is abstracted as a binding process between CORBA objects; each exports a binding interface which is represented as a Binding Interface Base (BIB). Establishment of a binding enables multimedia objects to communicate.

The architectural model defines five planes which model the familiar aspects of control, management, and information transport, and a data abstraction for information sharing within the network. User plane services include both distributed computing and transport protocols. This is similar to the MSN-CMA in which MSRPC2 offers the distributed computing environment, and MSNA offers stream based communication, but their model of multimedia communication needs is more complex. An architecture such as IMAC [Nicolaou91], integrated with the MSN-CMA would provide a service interface similar to that of MBA. Resource control in MBA is located in a separate plane from connection management and binding; the latter draws on the BIB, which is located in the data abstraction plane, and implements binding services within the control and management planes. All objects within the network are represented in the BIB by their CORBA IDL interface definitions, including both network components such as switches, and producers and consumers of multimedia data, their QoS constraints and resource requirements. Essentially the BIB is represented by the CORBA naming mechanism, and provides facilities for naming and location of object instances within the network. This is similar to the ANSA trader used by MSRPC2.

Owing to its use of CORBA, the architecture has access to the standard features of a distributed programming environment, including naming, and automatic instantiation of objects by a "factory". Several "virtual resource" classes such as virtual links and switches are defined as base types in the architecture. These can be refined to incorporate new equipment types. Binding is achieved by invoking standardized binding primitives supported by each object. Rather than standardize on a signalling protocol the authors call for standardisation of the binding methods supported by interfaces in the BIB. This allows great flexibility in the underlying implementation of the binding method: it could involve signalling using a standardized protocol such as the ATM Forum UNI, or something more flexible. In this the architecture differs from the MSN-CMA, which adopts a single signalling protocol network-wide for a specific reason, namely the reliability requirements of the network. The MBA BIB could be implemented on top of the MSN-CMA since it requires only a distributed programming environment in the multimedia network. The MSN-CMA, however, forces the control of equipment types with disparate control protocols to a lower level (managed as subtypes of the SCI), enabling verification of the signalling protocols. This makes the MD the worst-case unit of failure in the MSN-CMA as opposed to the entire network. MBA represents the binding algorithm by a binding method represented by its IDL description. No specification of the underlying binding protocol is given.

The process of binding in MBA relies on CORBA RPC, and as such requires a reliable transport. Similar to the MSNL Master, a *connection manager* is used to set up bindings across the network. Like the MSNL Master, the connection manager may reside outside the network. The MBA architecture is currently being implemented.

4.5.7 DCAN

The basic architectural approach of the MSN-CMA described in Chapters 3 and 4 has led to the establishment of a three year research project at the Computer Laboratory, APM Ltd. and Nemesys Research, to develop an architecture for distributed control of ATM networks. The project commenced early in 1995, and will use real time extensions to the ANSAware distributed programming environment [Li95] to develop management and control systems for an ATM LAN, and an ATM applications programming interface with suitable features for multimedia applications. Implementations of the architecture will be carried out on UNIX, Windows NT and the Nemesis operating system [Roscoe95]; a goal of the project is to enhance the portability of ATM applications between these platforms.

4.5.8 SPANS

The SPANS signalling protocol [FORE95] is used by the LAN ATM equipment vendor, FORE Systems. It implements message-based connection control between users and switches, and between switches, and does not define any architectural aspects of network control. Like the MSDR representation used in the MSN-CMA UMI, SPANS message encoding is procedural, using the XDR [Sun87] representation. SPANS is also lightweight, using an unreliable transport with timeouts and retransmission as in the UMI. Unlike the UMI protocol, it is not RPC based. The protocol incorporates 26 signalling message types which enable the management of point-to-point and point-to-multipoint connections, including multicast membership. In the MSN-CMA multicast is implemented on top of the UMI. SPANS is poorly defined, perhaps due to its proprietary nature – no protocol procedures are specified, in spite of the large vocabulary. It is thus difficult to be at all confident of its correct operation.

4.5.9 The Paris / plaNet Approach

The approaches mentioned above all require several messages or RPC exchanges to set up a single connection across the network. This is true also of the UMI protocol in the MSN-CMA. An alternative mechanism has been proposed by Cidon *et al.* [Cidon93], which aims to reduce the latency of the resource allocation phase of connection setup. The protocol supports fast bandwidth reservation, and guaranteed release of resources in the face of switch or link failures. Although designed for the Paris/plaNet network, it could also be used in the ATM context.

Each switch is assumed to be under control of a single Network Control Unit (NCU). A user makes a call request at its local NCU, providing bandwidth and destination parameters. The NCU determines a (source) route to the destination and then employs a fast setup protocol to establish the connection. It first performs resource reservation at all nodes along the path, and then verifies that they all had sufficient capacity to grant the request. Only if the verification is positive can transmission commence. The protocol attempts to optimize the time taken to reserve resources, and not necessarily the connection setup time.

A connection SETUP request is forwarded along the route obtained from the NCU. When the request reaches the destination an ACKnowledgement (or a REJECT) is returned. When the source receives an ACKnowledgement from the destination, it issues a COMMIT request to verify the path. Any node which failed to allocate the required resources may ABORT the connection. A second ACKnowledgement is returned to confirm the path. Thereafter transmission may commence but the source NCU periodically issues a REFRESH to ensure that the path is maintained. Failure of the source or destination will cause these messages to be lost, and the connection's resources will be reclaimed. The authors provide a proof of the correct operation of the refresh mechanism. The protocol is robust in the face of NCU failure: a node which reboots may learn which connections were routed through it by monitoring the REFRESH and ACKnowledgement messages it receives.

This protocol shares with the MSN-CMA the property of correct operation in the face of node failures. To offer this it requires the use of a three-way handshake. It also attempts to reduce the contention time for resource allocation during setup, like the modified three-phase approach of the MSN-CMA. It differs significantly in many ways, requiring source routeing, permanent virtual channels for connection setup, and intelligence in every switching entity. It also requires a reliable data-link protocol for signalling, which is based on message exchange. These restrictions limit the applicability of this approach.

4.6 Summary

This chapter has described in detail the UMI protocol used by the MSN-CMA. Key features include its lightweight approach and the use of RPC for signalling. The detailed specification and verification of the UMI affords greater confidence in its correct behaviour in the operational environment. The use of TLT permits direct translation into the RPC paradigm, and facilitates compositional verification of the architecture and network services. In contrast to many related approaches, the MSN-CMA offers more than simply a network interface: its novel approach to distributed control, based on a lightweight connection model, makes its architectural approach attractive.

Chapter 5

Implementation of the MSN-CMA

The MSN-CMA, described in the previous chapters, offers flexibility and enhanced network control. To verify that the design is both practical and efficient, it is necessary to implement it on the target network. This chapter describes the implementation of the MSNL Master and the control interfaces within the entities it manages. The master is implemented as one or more independent processes, external to the operating system kernel (the platform), and may run on a different entity. Removing the control plane functions from the operating platform offers several benefits:

- Use of platform independent communication mechanisms allows the control plane code to be easily ported to different operating platforms.
- Development of the control plane can be accomplished in an environment suited to program development, which includes testing and debugging facilities.
- The control plane can be readily modified without having to modify the operating platforms in each entity within the network.
- The code size of each operating platform is decreased by removing the control plane functions.
- The control plane can be readily adapted to incorporate evolving ATM standards.
- The enhanced flexibility of the system permits easy experimentation with different network control algorithms, and assessment of their performance.

In the Cambridge ATM environment, two platforms implement the MSNA protocol suite, namely the Ultrix operating system (4.3 BSD Unix based) and the Wanda micro-kernel. In both of these the existing implementations were re-written to implement components of the MSN-CMA.

The MSNL Master and its related services were implemented on Unix as a user space daemon. Careful implementation ensured that it is portable to Wanda. Section 5.1 describes the Unix implementation and comments on its performance. The Unix kernel implementation was performed by Richard Black of the Computer Laboratory, and is not described in detail. Details of both the master and the kernel support may be found in a joint paper [Black94a].

Section 5.2 describes the implementation of the UMI and SCI service interfaces within the Wanda micro-kernel, on a Fairisle port controller. The implementation of the in band control functions of MS-Access, represented by the CLS, is described in Section 5.3.

5.1 Implementation of the Unix MSNL Master

In the current implementation each Unix host implements the full services of the MSN-CMA within a single user space daemon named the MSNL Master. Master and kernel communicate to establish connections for Unix applications. Only simple extensions are required to enable a single master to control multiple Unix hosts.

5.1.1 The Socket Interface

The 4.3 BSD Unix IPC interface, the socket interface, presents each application with a set of control and user plane functions. In the user plane the socket interface to MSNA provides the application with direct access to an MSNL liaison. MSNL is offered as a new address family, `AF_ATM`. User plane primitives include the `send()` and `recv()` system calls. Each Protocol Data Unit (PDU) to be transmitted over the connection is handed directly to MSSAR (both FDL block reassembly and AAL5 are supported) for segmentation and transmission. Similarly, received PDUs are handed up to the socket layer by MSSAR after reassembly.

In the control plane, the application uses the `bind()` system call to bind an MSNL SAP to a socket. This corresponds to the SAP allocation procedure in the UMI. A passive user may `listen()` for and `accept()` incoming connection requests, and an active user may `connect()` its socket to an offered SAP. Although the UMI permits a user to specify the QoS required for a connection, there is no clean mechanism in the socket interface for an application to provide these parameters to the `connect()` system call. Consequently, in this implementation all ATM connections make use of AAL5, and are assigned a default "best-effort" QoS. Figure 5.1 shows the structure of the MSNA implementation for UNIX, which is described below.

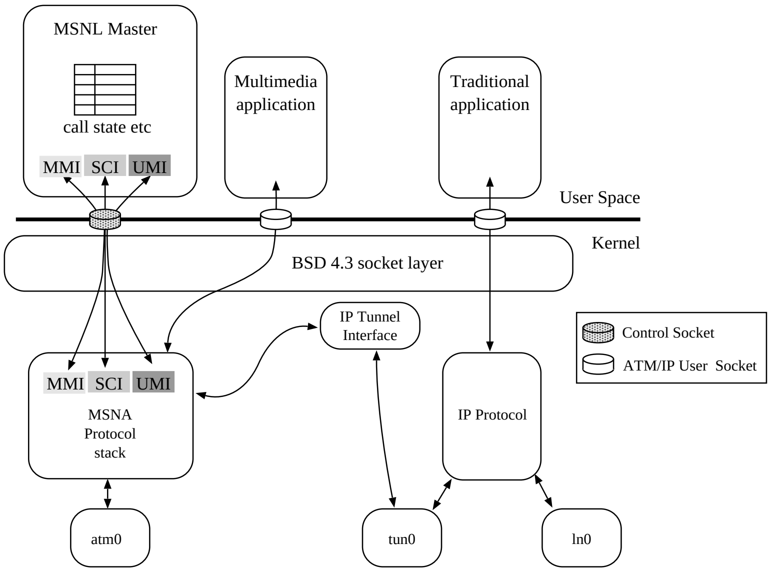


Figure 5.1: Overview of the UNIX implementation

5.1.2 Kernel Support

Within the kernel, at the MSDL layer each connection is represented by an *association* record, and each socket is associated with a single association. In the case where the connection is local, no association record is required and each of the two sockets corresponding to the endpoints of the connection contains a reference to the other. This is similar to the local connection case in `TCP`. During connection setup the master instructs the ATM protocol code in the kernel to build the data path for the socket. When the connection is complete the protocol control block (PCB) associated with the socket contains a reference to the ATM association record for the connection. Final authority as to the state of a connection lies with the master.

Additionally the ATM protocol provides logical interfaces to the IP code within the system: in this case its client is the IP code and requests from the IP layer to set up tunnels over the ATM network to carry IP traffic are sent to the MSNL Master.

The master can be extended to accommodate emerging international standards for ATM networks, such as the ATM Forum signalling protocol and the

IETF standard for IP address resolution on ATM [Laubach94, Perez95], without modifying the kernel.

5.1.3 Communication with the Kernel

The MSNL Master requires a mechanism to communicate with the socket layer and MSNA protocol stack within the kernel. Both the SCI and UMI need access to the kernel, and in addition the MSNL Master must receive all signalling from the attached networks. Two options were identified, namely the provision of a special device in `/dev` or a "magic" control socket type. The latter option was chosen and implemented by adding a socket type `SOCK_RAW` to `AF_ATM` which can have only one instance per machine. Two major considerations motivated this choice:

- For concurrency reasons it was preferable for the master to interface directly with the networking code rather than via the file system. This is particularly important because most of the information for the master is generated as a result of calls from the socket layer.

- The master is responsible for all network signalling; these signalling requests must be presented to the ATM layer in the form of queues of `mbufs`. Since the mechanism for encapsulation of data in `mbufs` is provided by the socket layer, it is preferable to transmit and receive signalling via the socket interface. The asynchronous nature of interaction between the kernel and the daemon maps well onto asynchronous socket communication.

The control socket is used to exchange three types of information: socket layer requests, ATM layer requests and ATM signalling. Socket layer requests to the master include, for example, requests to `bind()` an ATM address to a socket, `connect()` a socket to a destination ATM address, and `close()` a connection, and correspond to the UMI protocol. The master responds to socket layer requests with the appropriate responses, enabling the kernel to modify the state of application sockets. ATM layer requests correspond to the SCI: the master uses the control socket to issue commands to the ATM layer to build each association, provide it with a virtual circuit identifier (VCI), and monitor its status. At the time of this implementation the MMI protocol was not supported on other platforms and thus the MMI signals are converted within the MSNL Master to and from metasignalling messages.

Communication between master and kernel takes the form of message exchange. Each message identifies the control interface to which it is directed (socket layer or ATM protocol). Most messages are of fixed length except when accompanied by signalling cells. In addition, a few management operations cannot be performed asynchronously and are transferred using the `ioctl()` system call, which allows the kernel to synchronously modify the message to return the required response.

The kernel code and the MSNL Master identify individual MSNL liaisons by means of their associated PCBs. Within the master the PCB is regarded as the unique call reference. The kernel verifies that each command from the master refers to a valid PCB. Each PCB contains methods for use of the user protocol, and methods which implement the transmission path, as well as an opaque value which, in the case of a local connection, is a pointer to the PCB for the peer socket, and in the case of a remote connection, is a pointer to the MSDL association.

5.1.4 Implementation Problems

The following subsections address several of the implementation problems which were encountered, some of which resulted directly from the design choices.

5.1.4.1 SAP Allocation

The BSD 4.3 socket code requires the implementation of the `bind()`, `getsockname()`, and `getpeername()` system calls for all protocols to be non-blocking. Many applications perform a `getsockname()` immediately after a `bind()` in order to publish or export their service address. This unfortunately means that the implementation of the MSNL SAP allocation function must remain within the kernel. The addresses allocated are reported to the user space master, which checks the consistency of the kernel allocation.

5.1.4.2 Semantics of the `accept()` System Call

MSNA allows the passive user to consider both the address of the peer and the quality of service requested before deciding to accept a connection request. This is directly at odds with the BSD 4.3 semantics which require that a connection be declared "soisconnected" before the receiver is informed of the connection. In the implementation an incoming connection request is forwarded to the master which in turn presents the connect request to the kernel ATM layer. On receipt of the connect request the ATM layer declares to the socket layer that the connection is complete (soisconnected), and prevents subsequent transmission on the socket associated with the connection by manipulation of the socket transmit buffer. It then instructs the master to accept the connection. Once the master has accepted the connection and built the data path, it informs the ATM layer that the connection is complete and transmission and reception are enabled.

Although a UNIX server is still prevented from vetting its peer before accepting a connection, the implementation preserves the expected ATM protocol semantics. This, and the additional drawback that applications cannot cleanly specify their QoS requirements makes the BSD interface unsuitable for multimedia applications.

5.1.5 The IP Logical Interface

Within the kernel a logical IP interface is provided which makes use of ATM connections as "tunnels" – each tunnel is a logical link in the IP network. The IP tunnelling code generates requests to the MSNL Master to set up an ATM connection to each active IP destination. Idle connections are torn down by the MSNL Master after a period of inactivity. It would be preferable to use one ATM connection for each TCP connection or UDP session. This, however, would have required extensive modification of the BSD 4.3 IP code.

To set up an ATM connection the IP tunnelling code must map the IP address to an ATM address. This is performed by the master, allowing an `AF_INET` address to be specified as the destination for an ATM connection. This provides more flexibility than the static mapping as used in a previous mechanism for IP address resolution over X.25 [Malis92, Sun90].

Although the UMI protocol permits the specification of user QoS requirements, such as the IP MTU, the current implementation uses 4 different MTU sizes to enable experimentation. Details of the IP implementation are to be found in [Black94a].

5.1.6 Design of the User Space master

The user space master implements the control and management planes for MSNA. It is responsible for routeing, MMI signalling (in the form of metasignalling) and communication with the control interfaces in the kernel. It also performs mapping of IP to ATM addresses, and implements a RARP daemon. Figure 5.2 shows the internal design of the MSNL Master, which is described below.

Since the control socket delivers requests asynchronously and since the master needs to be able to process several requests (from all layers) simultaneously, it is implemented as a multi-threaded process. The implementation uses the POSIX Pthreads package [IEEE92] to implement several threads of control to manage communication, timers, and management functions. Independent threads are scheduled by the Pthreads package, which also implements concurrency control primitives for shared data. As the performance measurements in Section 5.1.7 show, these primitives have a high cost.

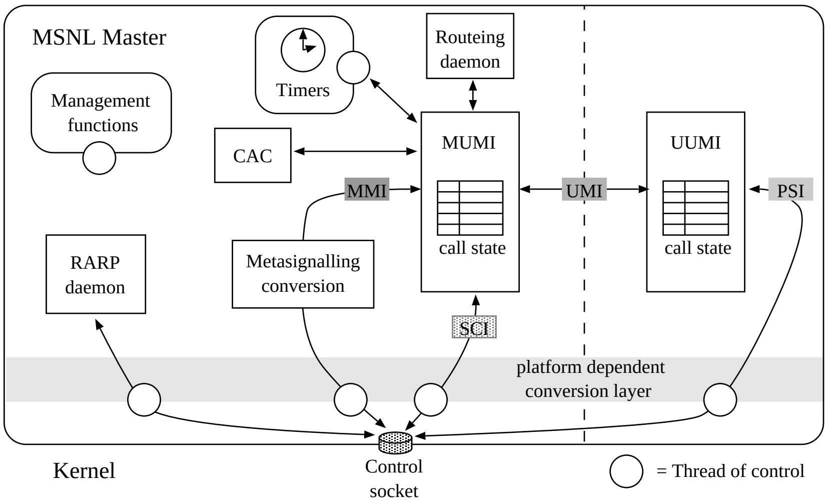


Figure 5.2: Detailed design of the MSNL Master

Messages from the kernel are received by one of a number of independent threads, and depending on the message type, delivered to the appropriate service interface. A conversion layer translates kernel messages into RPC requests for use within the MSNL Master. Internally the MSNL Master is structured as two halves, implementing the UUMI and MUMI respectively. This split is not necessary – it would have been easier to regard the kernel requests as UMI requests. Treating them as PSI requests and implementing both sides of the UMI in the master, however, has the advantage that the implementation of the UUMI can be split off and run as a separate process which communicates with a remote master, allowing a single master to manage more than one entity. It also enabled the UMI protocol to be thoroughly tested.

At the socket layer each kernel request specifies the unique call reference (the PCB). Requests are received asynchronously on the control socket. For each call reference the MUMI maintains a record of the state of the connection, the association fields such as the VCI, and ATM addressing information. The MUMI issues requests to the protocol layer (via the UUMI) to process incoming connection requests and to inform the socket layer of the state of each connection. The MUMI uses the SCI to build the data path for each connection at the ATM layer. It issues requests via the control socket to build associations, assign VCIs and monitor the status of each connection. Received metasignalling messages are converted into the MMI format for injection into the MUMI, and MMI signalling is converted into metasignalling for transmission on the appropriate ATM interface.

An important consideration when designing the master was the potential in-consistency between the master and socket layer views of the state of each connection. For example, an application could issue a `listen()` and then immediately `close()` the socket. Due to the asynchronous nature of the control socket, the master might attempt to complete an incoming connection to the listening socket after the application had already issued the `close()`. To prevent this, the kernel code checks all instructions issued by the master to ensure that they pertain to valid PCBs. Similarly, the master will destroy any socket for which the kernel issues an invalid request. Final authority as to the state of a connection lies with the master.

5.1.7 Performance

The performance of the control path was investigated by measuring the time taken to set up connections over the ATM network and the Ethernet. It is affected by several factors: scheduling of the master as a user space process, concurrency within the master, and communication between the master and the kernel. Table 5.1 gives typical connection setup and teardown times over the Ethernet, a direct ATM link and a switched ATM link between two lightly loaded DecStation DS5000/25 hosts running the kernel and master, and for local IPC on a single host. The performance measurements were made using the `etp` profiling tool [Burrows93].

Network	Setup (ms)	Teardown (ms)
Ethernet	21.7	8.4
ATM	20.4	8.1
Local	21.8	3.2
ATM (switched)	42.1	8.7

Table 5.1: Connection setup and teardown times

The implementation performs worse than an earlier implementation of a (simpler) version of the control plane within the UNIX kernel, for which average connection setup times of 3.5 ms were measured between two UNIX hosts over a lightly loaded Ethernet [Dixon92]. Although it would be unreasonable to expect this implementation to perform as well as a more mature, though less functional, kernel implementation, better results might have been expected. The results can be only partially explained by competition between the client and master (and in the local case, the server), all of which were running at the same, default, user priority level.

To determine why connection setup took longer than expected, the master was profiled. The results showed that the master typically spends 52% of its active

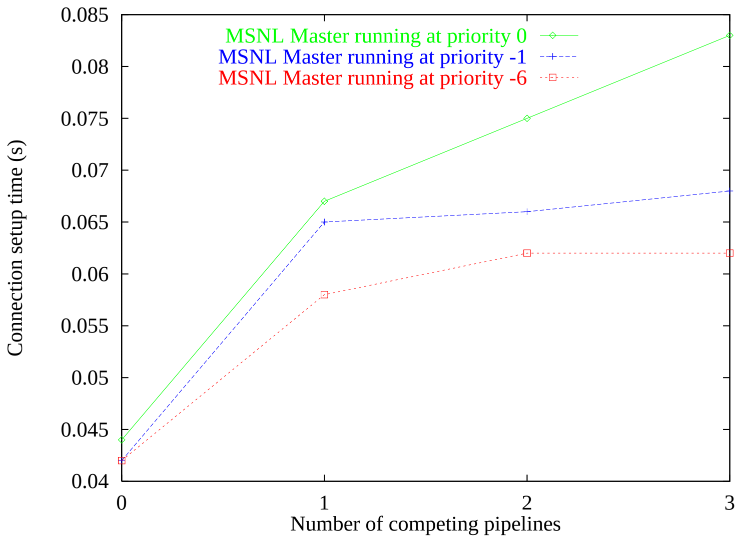


Figure 5.3: Influence of system load on mean connection setup time

time (i.e. time not blocked in the `select()` call) executing code in the Pthreads library. In all of the measurements, the ten most frequently called functions were in the Pthreads library. Most of these functions are concerned with thread context switching and concurrency control; for example, the function `setjmp` (a context save) was called on average 151 times when a connection was setup and immediately torn down. Clearly the concurrency control primitives have a high cost. The master could be made more efficient by implementing it as a single threaded process and by discarding the redundant UUMI. On an operating system which supports lightweight threads, however, such as OSF/1, the multi-threaded design of the master should not impact its performance, and the connection setup time should be proportionately reduced. This, in conjunction with some further optimisations identified, would enable a reduction of approximately 50 % in the connection setup time, reducing it to the order of 10 ms.

5.1.7.1 Influence of System Load

Three further sets of measurements were taken to study the influence of system load on connection setup time. In each set the system load was varied by running between 0 and 3 CPU-bound processes in competition with the master. Each consisted of a shell pipeline[1] which required both user space and kernel processing. The competing pipelines were run at the default user priority of 0. For a given system load the performance of the master was measured by profiling

[1] `while true; do a=‘echo $PATH | grep wanda | sed -e "s/wanda/foo/g"‘; echo $a >/tmp/foo.$$; done`

a client which repeatedly set up a connection, exchanged a single message with a remote server, and then tore down the connection. In each experiment, the client set up 100 connections with a random interval between each. In each experiment connections were set up from a DS5000/25 over the ATM network, via 2 Fairisle ATM switch ports, to a second DS5000/25 running the same code. Both workstations were fitted with a simple, cell-based ATM interface [French92].

In the first set of measurements, the master was run at a priority of 0. As expected, the connection setup time degraded as the system load increased. Figure 5.3 shows an initial sharp increase in connection setup time with one competing pipeline, followed by a linear increase with increasing load. In the second set, the priority of the master was increased to -1. As can be seen from the graph, the connection setup time initially increased rapidly with increasing load, but the rate of increase slowed thereafter. In the third set of experiments the priority of the master was set at -6, higher than that of the automount daemon. This served to stabilise the performance, in spite of an initial sharp increase with a single competing pipeline. The results suggest that the master should be run at a high priority, to ensure that an increase in system load does not affect its performance excessively.

5.1.8 Concluding Remarks

An important consequence of the design is that the complexity of the system, in terms of code size, has been significantly reduced. The master compiles to approximately 300 KB, most of which is due to the Pthreads library. The size of the kernel ATM code has been reduced by about 30 % to a mere 27 KB of text segment including the ATM device driver.

In similar work, reported subsequent to the implementation described here, an ATM protocol manager external to the UNIX kernel was implemented at AT&T Bell Labs [Sharma94]. Their implementation uses the UNIX file system for communication with the kernel, an option rejected in this work, and requires applications to request services from the manager in addition to using the socket interface.

5.2 Implementation of the Wanda Control Plane

The implementation of the MSNL Master described in Section 5.1 is portable apart from a single platform dependent layer, shown in Figure 5.2, which is specific to the UNIX socket layer. Replacing this layer with an equivalent layer for Wanda permits the MSNL Master to be directly ported[2]. The following

[2] This is being done by Shaw Chuang of the Computer Laboratory.

subsections describe the implementation of the kernel support for the MSN-CMA within the Wanda kernel.

5.2.1 The UMI Functions

On Wanda communication protocols are implemented within the kernel. Applications use an IPC mechanism which appears similar to the socket interface; indeed the application abstraction is also called a socket. It differs from the BSD socket interface in two important ways:

- The IPC system is multi-threaded: multiple application threads may send or receive on a single socket.
- The semantics of the accept call allow the application to identify the connecting user and the requested QoS before the connection is accepted by the kernel. The application may refuse the connection by closing the (new) socket on which the request is made.

Wanda supports only two protocols within the kernel, namely MSNL and a raw Ethernet protocol. Each is implemented as a different protocol family, presented to the IPC system as a protocol block with methods for each of the control and user plane functions.

In the case of a Fairisle switch, the entire collection of ports may together form the management domain. Most ports will run a simple user space application which is equivalent to the UUMI in Figure 5.2. Some ports may additionally run services used by the MD, or the MSNL Master. Similar to the UNIX implementation, the MSN-CMA control plane is implemented in user space. Communication between kernel and user space represents UMI activity. On Wanda the communication mechanism used is a new protocol family specifically for the MSNL control plane, named `PROTO_MSNL_CTRL`. Any number of sockets may be open in this protocol, allowing each thread of control in the MSNL Master to have its own socket.

An additional transport was added to the MSRPC2 runtime which uses the `PROTO_MSNL_CTRL` family. The transport offers EXO semantics and allows RPC between user space and kernel. A service interface within the kernel (for example the UMI) exports an interface reference which the MSNL Master presents to the MSRPC2 runtime when binding to the service interface. A similar arrangement allows service interfaces in user space to be used from the kernel. A kernel client binds to the exported interface reference which in this case contains the PCB reference of the control socket on which the user space service is awaiting requests. RPC from kernel to user space supports only announcement operations,

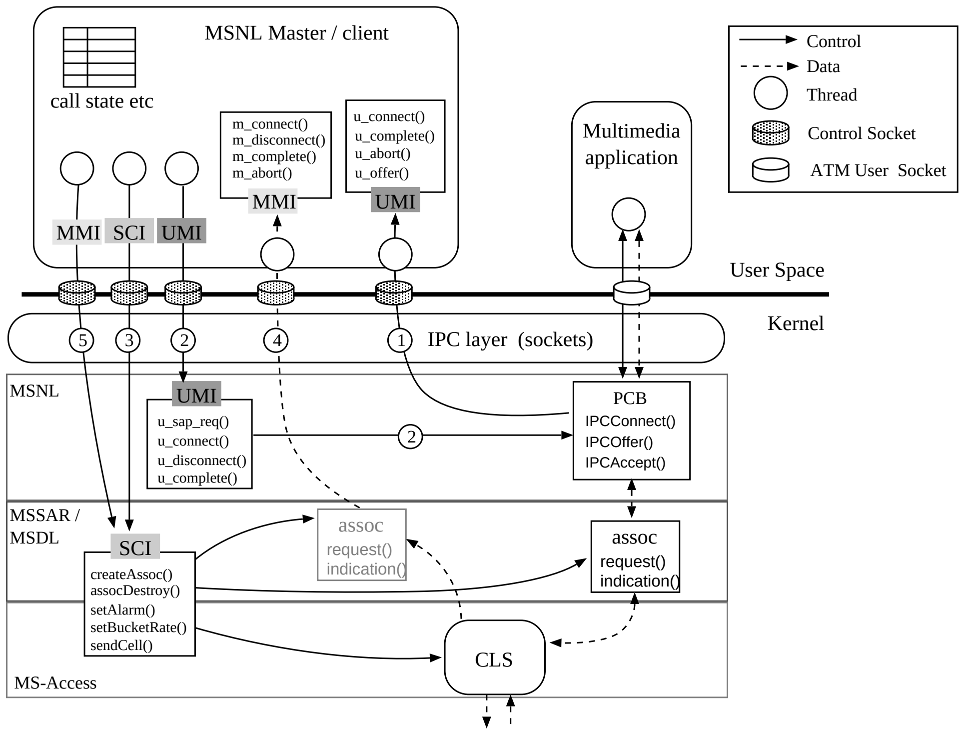


Figure 5.4: Implementation of the control plane in Wanda

which are non-blocking. Interface reference trading is implemented via a simple trader which exports interface references into the system environment space.

Figure 5.4 shows the design of the Wanda control plane implementation, for an entity which hosts the MSNL Master. A multimedia application with a socket of family `PROTO_MSNL` uses the IPC interface to control communication and send and receive data. In the kernel, control plane functions at the MSNL level are translated into invocations of UMI operations which are delivered to the master (step 1). Operations on the UMI invoked by the master cause actions on the user's PCB (2). Control functions at the MSSAR, MSDL and MS-Access layers are exercised via the SCI (3). These include operations to build associations, parameterise the CLS and build the data path. In this implementation MMI signalling was not supported, hence metasignalling cells are transferred (via a translation function) to the user space MMI (4), and metasignalling cells are sent via the SCI (5). The Wanda IPC system is multi-threaded, and care is required to ensure that the export of control to user space does not affect the consistency of the kernel state.

5.3 Implementation of the Fairisle CLS

The CLS implements the in band control functions within the user plane, and was described in Section 3.4.6. It is controlled via the SCI and presents to each MSDL association an instance of MS-Access which delivers its requested QoS whilst sharing access to buffer space and bandwidth in a fair manner. The resource allocation task is complex and is located within the master. The CLS simply implements the mechanisms required to deliver the allocation. Each physical device is controlled by its own CLS and its functions are necessarily dictated by the features supported by the device and the underlying MAC layer. For each device–MAC layer pair a subtype of the CLS is defined by a separate MIDDL interface.

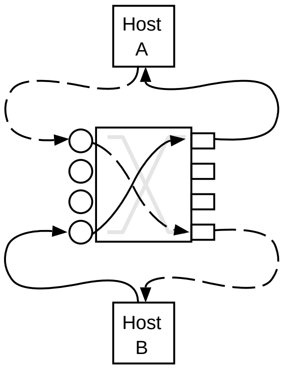


Figure 5.5: Directional transmission through the Fairisle switch

This section describes the implementation of the Fairisle CLS, which is the multiplexing point for all traffic entering the switch fabric. The device is presented to the Wanda kernel as two logical pods which share the buffer pool and access to the fabric, namely a point-to-point link (the transmission pod) and the fabric pod which enables communication between ports on the switch. A single CLS implements the cell level functions for both pods. The Fairisle CLS is central to the research described in the balance of this dissertation: its design permits maximum experimental flexibility whilst maintaining high performance. It is also interesting because it highlights the need to treat each direction of communication for an association separately. Figure 5.5 shows how the two directions of transmission between two hosts A and B, attached to a single Fairisle switch, are routed through its ports. Each direction of communication passes through only one port – the switch output is passive.

5.3.1 Association Specific Context

In the Fairisle CLS a *context* is maintained per association, in the form of a reference to association-specific state. A one-to-one mapping between a VCI and its context enables rapid context lookup. Each context exports the `rx()` and `tx()` methods and keeps a private pointer to a queue of cell buffers to which an arriving cell should be added. To preserve cell ordering, cells are always added at the tail of a queue.

5.3.2 Policies

Each policy, defined by its MIDDL interface, represents a scheduling mechanism and a buffer allocation policy which dictate how cells from different contexts are given access to the Fairisle switch fabric and cell buffers. The policy methods together implement these functions, and are defined in the policy interface; dynamic configuration may be performed by the master or by the methods themselves. Two types of scheduling mechanisms may be implemented: work conserving and non-work conserving. A work conserving policy is never idle whilst there are cells in the CLS, whilst a non-work conserving policy may be idle with queued cells. Each policy defines a context subtype, adding additional state, such as buffer thresholds and additional methods.

5.3.3 Control and Management of the Fairisle CLS

The Fairisle SCI exports operations in the control and management planes which allow the MSNL Master to:

- create and parameterise MS-Access instances in the CLS,
- select, install and control the CLS policy,
- monitor CLS statistics and status, and
- request and receive notification of events, such as alarms.

5.3.3.1 Control Plane Operations

Operations in the control plane are divided into two sets: the first controls the operation of the CLS as a whole, permitting the master to disable and enable the CLS, to load a named policy, and to schedule a policy-specific, periodic task. The second set contains context-specific operations, and permits the master to create, destroy and parameterise contexts. Since the context definition is policy-specific, these methods are implemented by each policy, and refine the generic methods exported by the CLS.

When a policy is loaded into the CLS, its policy-specific refinements of the CLS methods are used to replace those offered by the currently installed CLS. The policy-specific method table and context definitions become current. The context-specific operations allow the MSNL Master to set methods for each context, get context-specific performance statistics, and assign a context to an MSDL association, building the data path for transmission.

5.3.3.2 Management Plane Operations

In the management plane the Fairisle CLS allows the MSNL Master to retrieve status information and set control parameters such as the line rate, whether or not to perform header error checks and handshaking, and to reset the device. The status information includes performance statistics for the device as a whole, including transmission link quality, the fabric load, and the number of free buffers. The management plane also generates alarms in the form of events. Each event is a small record with a type and information field. The information field contains event-specific details. For example, an event indicating excessive cell loss could include a count of the cells lost. A method which generates an event awakens a kernel thread which forwards the events to the master. Within the master each policy implements a mechanism for receiving and interpreting its own events.

5.3.4 User Plane Access to the Fairisle CLS

Two types of access to the CLS are required. The first is for transmission from the fabric pod and for forwarding liaisons routed through the transmission pod, and the second for local liaisons which receive traffic from the fabric and transmission pods. The first type describes traffic which leaves the CLS via the MAC layer interface, and the second describes traffic which arrives from the MAC layer for the local system. The two cases were pictured in Figure 3.7 which shows the principles underlying the CLS. Local liaisons also require resources at layers above MS-Access, typically CPU time and memory. This problem has been addressed by the Pegasus project, for example in the *Fawn* implementation [Black95], and is beyond the scope of this dissertation.

In both cases an active *producer* and *consumer* are required which inject cells into and eject cells from the CLS respectively. These are described below for the two cases of interest. An example is given in Section 5.3.5.

5.3.4.1 Traffic Leaving via the MAC Layer

Again two cases are identified, namely traffic sourced locally and traffic from forwarding liaisons. For the former, the *producer* is the user thread transmitting data, which invokes the `IPCSend()` system call and carries the flow of control through the kernel to MSDL, where its data are segmented into cell buffers from a pool maintained at the MSDL interface. Each association record maintains a reference to its CLS context. The thread invokes the `rx()` method of the association's transmit context, passing as argument the context and cell buffer to transmit. This is repeated for each cell[3].

In the case of forwarding liaisons, the *producer* is the FIQ interrupt handler which, on receiving an interrupt from the Fairisle receive interface, removes a cell from the receive queue and, using the VCI, identifies the association context. It invokes the `rx()` method on the context, passing as argument the context and the received buffer.

Consumers may be either work conserving or non-work conserving, and execute a transmission schedule of contexts. For work conserving policies the *consumer* is the transmission mechanism, which raises an interrupt whenever transmission capacity is available. In response the FIQ invokes the `tx()` method of the context scheduled for transmission. For non-work conserving mechanisms the consumer is the timer which generates interrupts according to its schedule. The FIQ responds to the timer by executing the method corresponding to the scheduled event; this is the `tx()` method for the context scheduled for transmission.

5.3.4.2 Traffic Arriving from the MAC Layer

The *producer* is the FIQ interrupt generated by the receive interface. The context is identified from the cell VCI, and the `rx()` method invoked. This, in turn, is responsible for delivering an indication to the MSDL *consumer* to alert it that a cell awaits reception. Buffering must be implemented because priority is given to cells from forwarding liaisons. The indication is delivered by raising a lower priority IRQ interrupt which is delivered to the Wanda kernel when the FIQ is idle. The IRQ interrupt handler consumes the scheduled cell and, on the basis of the VCI, delivers the cell to its awaiting association, invoking the `tx()` method on the context.

5.3.5 The FIQ Interrupt Handler

Each policy provides a FIQ interrupt handler which is installed when the policy is initialised. The FIQ implements the *producer* function for cells arriving from the fabric or transmission interface, the *consumer* function for cells transmitted into the fabric, and the timer clock. It also performs retransmission for cells which exceed their retry counts and other device management functions. The FIQ is hand coded for each policy, and must be highly efficient. In practice

[3]For efficiency reasons batch transmission is also supported.

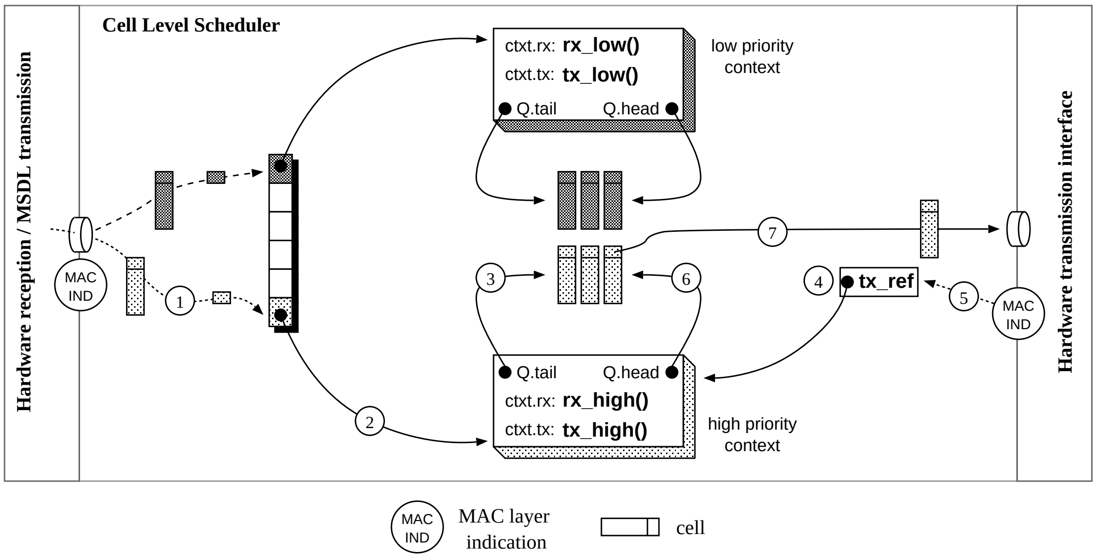


Figure 5.6: Implementation of Static Priority on the Fairisle CLS

only two FIQ implementations were required, for work conserving and non-work conserving policies.

The per-context methods are responsible for implementing the policy. This is more easily understood with the aid of an example. Consider a policy which implements two levels of priority for transmission, `high` and `low`, and which has a limited buffer size of 200 cells shared by both classes. The policy is pictured in Figure 5.6.

Only two contexts are required, for high and low priority traffic leaving the interface. The policy methods are `rx_high()`, `rx_low()`, `tx_high()` and `tx_low()`. An association of high priority is given `rx_high()` as its `rx()` method and `tx_high()` for its `tx()` method. A low priority association is given the low priority methods. On the arrival of a cell the hardware delivers an indication (alternatively, a cell could be delivered by MSDL); the FIQ uses its VCI to locate the context (steps 1 and 2). Subject to the buffer constraint, the cell is added to the tail of its queue (3). The `rx()` methods are responsible for setting the value of a context reference, named `tx_ref` which indicates which context is scheduled next: on reception `rx_high()` sets `tx_ref` to the context which invoked it (4).

The policy is work conserving: whenever transmission space is available the hardware delivers an indication (5) and the FIQ loads `tx_ref`, invoking its `tx()` method to remove a cell from the head of the queue (6) for transmission (7). The method `tx_high()` checks whether the high priority context is empty (i.e. no cells queued) and if so sets `tx_ref` to point to the low priority context. The *rx* and *tx* methods increment and decrement the queue length respectively; additional methods implement cell discard.

5.3.5.1 Hardware Features

In any high speed ATM device, aspects of the CLS will be implemented in hardware. Fairisle supports automatic VCI remapping for forwarded cells, hardware detection of faults and errors, some performance measurements, limited cell queue management, and automatic cell retransmission across the fabric. The FIQ and policy methods are thus optimized for the hardware. The layout of the context fields enables maximum use of the load/store multiple and branch instructions on the ARM CPU.

5.3.6 Monitoring and Statistical Measurement

The Fairisle CLS includes a library of methods frequently used by all policies, named `Qlib`. These methods are used for statistical sampling and monitoring, and include functions to:

- trace absolute cell arrival times,
- trace sequence numbers of lost or discarded cells from each context, and the absolute time (using a 32 bit clock) of discard,
- measure inter-arrival, and inter-departure time distributions,
- measure the instantaneous and averaged load, and
- measure the queueing delay, queue length and cell delay variation distributions for each context.

5.3.7 Performance of the CLS

Performance of the implementation depends on the complexity of the policy methods. For a simple FCFS queueing policy the CLS can sustain a cell forwarding rate of approximately 60 Mb/s or 0.6 of the line rate whilst performing simple measurements. Although less efficient than the FIQ implementation reported in [SRG94] this implementation has the advantage of being flexible enough to implement far more complex scheduling policies.

Under work conserving policies the FIQ continues to operate whilst there is traffic in the CLS. This means that the management code does not get time to run. This is a drawback which can only be remedied either by scheduling a (periodic) method which forces the CLS to return control to Wanda, or by using a non-work conserving policy. *Fawn* solves this problem by scheduling the Fairisle device driver as an application domain [Black95].

The overhead of FIQ interrupts is severe: measurements indicate that the latency from the instant of cell reception until it is transmitted into the fabric is typically 2 cell times (approximately $7.11\mu s$). When the traffic load is sufficiently high, this overhead is avoided because the CPU remains in FIQ mode between cells.

5.4 Concluding Remarks

This chapter has described the implementation of the key features of the MSN-CMA, validating the architecture presented in Chapter 3 and the UMI protocol. Unlike virtually all of the related protocols and architectures reported in Chapter 4, most of which have yet to be implemented, the MSN-CMA has been shown to operate reliably in a large network where it is used on a daily basis to support experimental work. Although some aspects of the implementation are Fairisle specific, the architecture could be implemented on any network. The implementation has demonstrated that significant benefits can result from removing the control plane functions from the operating system kernel, whilst still delivering good performance. Furthermore, the implementation paves the way for further work in the DCAN project, which will build on the architecture and implementation to provide a fully functional service interface suitable for complex multimedia applications.

One of the principal aims of the MSN-CMA was to facilitate experimentation with cell level control functions and accurate performance measurement in the experimental environment. Successful implementation of the Fairisle CLS was key to the success of the research described in the remaining chapters of this dissertation. The implementation is both flexible and efficient: the system performance can be precisely measured with minimum interference, using real ATM traffic and at high system loads.

Chapter 6

Cell Delay Variation in ATM Networks: a Practical Study using Fairisle

Previous chapters have described the design and implementation of an experimental control architecture for an ATM network. This and subsequent chapters describe the practical use of the architecture in a set of investigations which investigate the performance of ATM networks under various control regimes. This chapter investigates the effects of multiplexing within the network on CBR connections, by studying the cell delay variation they sustain. Some of its findings are reported in [Crosby94]. Chapter 7 presents results of an investigation into the expansion of bursts within the network. Finally, Chapter 8 investigates cell level control mechanisms which aim to deliver performance guarantees to delay and loss sensitive traffic.

6.1 Introduction

The Cell delay variation (CDV) is a measure of how closely the temporal characteristics of a stream of ATM cells match those of an ideally spaced, constant bit rate (CBR) cell stream of the same mean rate. The causes of CDV are well known: queueing delays and the effects of switching and multiplexing can cause cells from a well behaved connection to become grouped together, artificially increasing the peak transmission rate and making the traffic more bursty. This, in turn, makes it difficult for switches in the network to provide guarantees on cell delay without discarding traffic (alternatively, more buffer space is needed to ensure that loss constraints are met). The CDV experienced by a connection determines the size of the buffer required at the receiver to ensure smooth playout of the signal; if this is too large the perceived quality will deteriorate.

The CDV is also intimately related to the leaky bucket algorithm, adopted by the ITU-T [ITU-T93b] and the ATM Forum as a technique for policing the peak rate of traffic entering the network. The algorithm relies on observation of the cell stream at a single point, and is consequently known as the 1-point CDV measure. The CDV y_k experienced by cell k is defined to be the difference between its theoretical arrival time at the measuring point, c_k, and its actual arrival time a_k. If T denotes the peak emission interval (the minimum inter-cell time) then the set of theoretical arrival times is defined recursively:

$$\begin{aligned} c_{k+1} &= c_k + T \qquad \text{if } y_k \geq 0 \\ &= a_k + T \qquad \text{if } y_k < 0 \end{aligned}$$

Cells which arrive "early" (i.e. in excess of the peak rate) may be discarded, subject to a tolerance parameter τ.

Although the leaky bucket algorithm views the problem from a network-centric point of view, CDV is a problem for both users and the network. If traffic which is compliant with the leaky bucket suffers significant additional CDV, it may severely impact both other users and the quality at the receiver. It has been suggested that connections should be policed at inter-network boundaries to drop cells which exceed the negotiated peak rate. If this is the case, what degree of tolerance is required so as not to affect a well behaved user? It is important that users who comply with their traffic contracts are not penalised as a result of the (hidden) effects of multiplexing. It is not yet clear, however, whether such controls are necessary, nor, if they are required, where they should be located. It is also not clear what traffic and network parameters critically affect the CDV sustained by traffic of different types. This chapter investigates the CDV sustained by ATM traffic to determine the degree of tolerance required by the network and end-systems in the absence of traffic shaping functions.

6.1.1 Related Work

To date, most research into CDV in ATM networks has been based on analytical models, for example [Roberts92, Molnár94, Gravey94]. These studies tend to approach the CDV problem by studying traffic at a single switch in isolation, and deriving bounds on the maximum queue length seen by an arriving customer. Only three studies of which the author is aware have attempted to model analytically the CDV arising in a tandem arrangement of queueing systems typical of a multi-hop ATM network [van den Berg92, Kelly94, Matragi94]. The first two are used in this chapter, which compares their predictions with measured results. They are described in Section 6.3 below. The conclusions of the third also offer valuable insight into the causes of the CDV observed in the experiments described below.

The results of this study represent the only published observations to date of the CDV suffered by connections in a real ATM network carrying real traffic. Recently the RACE EXPLOIT project has reported the practical validation of several analytical models of CDV [Brinkmann94, Louis94, Nellas94]. This work, however, suffers from several important drawbacks: first, the traffic is artificially generated to conform to the assumptions of the analytical models, and bears little relation to real ATM traffic. Second, the ATM network used (the RACE EXPLOIT testbed) is equipped with few traffic generators, and only four switches. To achieve the high loads required for experimental work the output traffic from a switch must be re-introduced (several times) at different inputs of the same switch, causing correlation in the traffic. Finally, the commercial ATM equipment used lacks the required facilities to measure important features of the cell scale process.

6.1.2 Aims of the Study

In contrast to the limited validation studies described above, this chapter reports on the findings of a study of CDV using sources of real ATM video and data traffic, on the Fairisle ATM network. It relates the measured results to the predictions of analytical models. Fairisle is ideally suited to this work because it provides hardware support for high time-resolution measurements.

This chapter considers the effects of the following factors on the CDV experienced by a given reference connection which traverses a network of ATM switches in which cells are queued in a FCFS fashion:

- the *sizes of the switches* (number of ports) in the network which the connection traverses,
- the *types of the switches* involved (either input or output buffered),
- the *types of background traffic* with which the reference source contends for buffer space and transmission capacity,
- the *number of switching stages* traversed by the connection,
- the total *network load* on each link traversed by the connection, and
- the *proportion of the total load* contributed by the connection under study.

6.2 Experimental Network Configuration

Each experiment aims to determine the CDV (and other performance measures) experienced by a single reference connection in a network of input or output

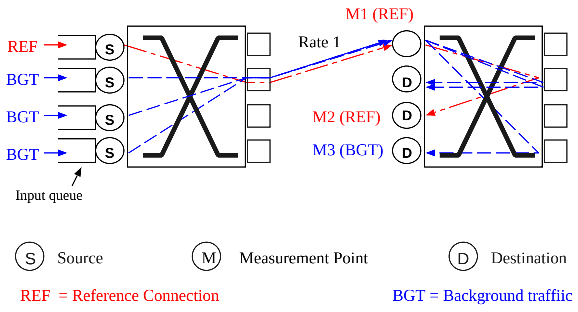


Figure 6.1: Single stage switching topology for blocking (input buffered) configuration

buffered ATM switches. Input buffered switches are termed "blocking" because cells must be queued at the inputs during switching. An output buffered switch is "non-blocking" because all queueing and buffering occurs at the switch outputs. Typically they are based on non-blocking fabrics which ensure that in a single switch cycle one cell from every input can be delivered to a single output. In Fairisle contention occurs at the output of each crossbar element and also for access to the output FIFO. Contention at any of these points may generate backpressure across the fabric to the inputs, causing queueing. Fairisle is thus by design a blocking switch. Section 2.1 described Fairisle in detail.

6.2.1 Input Buffered (Blocking) Configuration

The input buffered configuration can be realized by using the Fairisle equipment in its "natural", input buffered state. These experiments used switches of sizes 4×4 and 8×8. Throughout, the symbol D is used to represent the switch size. Figure 6.1 shows a typical experimental setup using a four port ($D = 4$) input buffered switch. A source enters at each input and sources contend for the same output port. The contention causes blocking, which in turn causes CDV. At the second switch the connections are looped back to sinks where measurements are taken. The abbreviation BGT is used in diagrams and figures for "background traffic", and REF for "reference source".

6.2.2 Output Buffered (Non-blocking) Configuration

Many manufacturers of commercial ATM equipment base their switch designs on an effectively non-blocking (output buffered) fabric. Although this increases the complexity and cost of the switch, output buffered designs are appealing because they alleviate the effects of contention on the switch fabric. To construct an

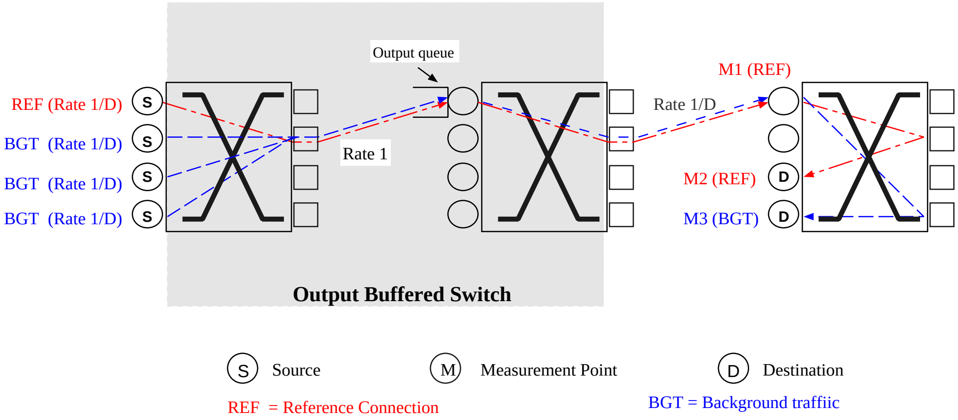


Figure 6.2: Single stage switching topology for non-blocking (output buffered) configuration

output buffered switch the switching stage must be made effectively non-blocking. In these experiments this was accomplished by using two input buffered Fairisle switches in tandem to implement a single output buffered switch, as shown in Figure 6.2.

In the diagram each link is labelled with its transmission rate, relative to the full line rate, 1. The transmission rates of traffic sources are scaled by a factor $1/D$, where, as before, D is the degree of the switch. The transmission link rate for the output ports of the first switch is set to the full link rate. Traffic arriving at the inputs of the switch is switched by the first fabric and multiplexed onto the full rate link. The output link of the second switch is scaled by $1/D$. Traffic arriving at the input of the second switch is blocked by backpressure across the fabric from the output whose link runs at the reduced rate. Queueing occurs at this point, which can be viewed as the output buffer of the switch so constructed. The output buffered switch has a switching capacity of D times the input and output transmission rates, and is therefore effectively non-blocking. This study used output buffered switches of sizes $D = 4$, 8, 16, and 32 ports.

6.2.3 Measurement Points

Figures 6.1 and 6.2 show some of the measurement points implemented to obtain performance results for each experiment:

- M1 – measures the CDV experienced by cells from the reference source, and their inter-arrival time distribution.
- M2 – measures the offered load from the reference source, and traces the arrival times of the first 10^6 cells.

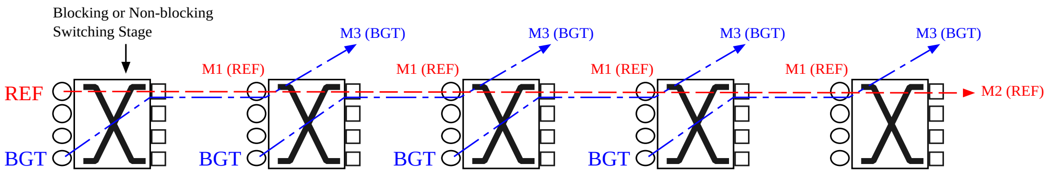


Figure 6.3: Network topology for multiple switching stages

- M3 – measures the offered load of the background traffic to ensure experimental accuracy.

The accuracy of the experimental load was monitored by measuring the contributed load from the reference stream and the background traffic at independent measuring points. The load from each source and the total multiplexed load is accurate to within less than 1%. All measurements were made using a clock which has a resolution of about 1/10 of the cell slot time.

The network topology for experiments involving more than one stage of switching is shown in Figure 6.3. At each switch the background traffic departs and a new (independent) source of background traffic of the same type is multiplexed with the reference stream.

6.3 Statistical Measures

In each experiment the reference connection was monitored at each multiplexing point and at the sink. To assess the impact of multiple stages of switching, experiments with up to 4 stages of switching were carried out. In each experiment 10^7 cells from the reference connection were transmitted. A desire to extend the experiments further later led to the use of 12 stages however the method used limited the experimental length to the transmission of 10^6 cells of the reference source. In spite of their limited length, however the results seem to be accurate[1].

6.3.1 Measurement of the CDV

In these experiments the CDV was measured using a version of the ITU-T Recommendation I.371 leaky bucket algorithm which samples each positive CDV value, shown in Figure 6.4. For a given reference source s the leaky bucket was parameterised with T_s, the minimum inter-cell time of the source, and the selected

[1] In order to state this with confidence, a statistically significant number of repetitions of each experiment would be required. Owing to the large number and complexity of the experiments, this was not possible.

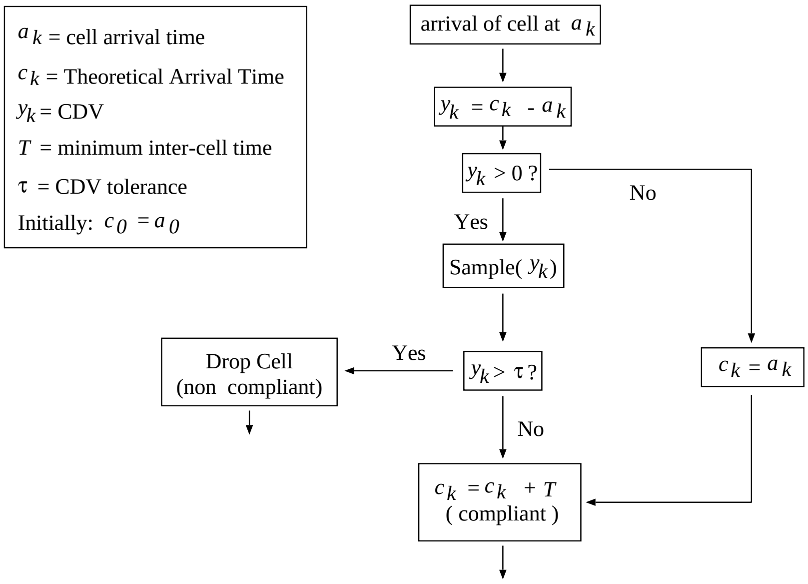


Figure 6.4: The Sampling I.371 Leaky Bucket Policer

tolerance τ_s to accommodate CDV introduced by multiplexing. A value $\tau_s = \infty$ was used, so no cells were dropped. To do this it was necessary to speed up the leaky bucket to ensure that the CDV remained bounded – a value of $T = 0.95T_s$ was used, a 5% speedup in the leak rate.

A problem with the 1-point CDV measure is its dependence on the time at which measurement commences: given the same input stream, but starting at different points in time, the algorithm will produce a different sequence of observed CDV values, and if used to police a reference connection the algorithm will discard different cells in each case. It can be shown, however that as the length of the observation period tends to infinity, the *proportion* of cells discarded is the same irrespective of the starting point [Gravey94]. Recently an exponentially weighted CDV measure has been proposed, which produces the same CDV distribution (with mean 0) irrespective of the starting point of observation [Roosma94]. A drawback of this algorithm is the complexity of performing the exponential weighting on-the-fly.

An alternative definition, the so-called 2-point CDV, is defined to be the difference between the absolute cell transfer delay between two measurement points and the delay of a reference cell which experiences no contention [Gravey94]. The 2-point CDV is difficult to measure without a (synchronous) global clock, and was thus not measured. Because of this difficulty, the usefulness of the 2-point measure is also questionable.

6.3.2 Variance in Inter-cell Times

To provide insight into the effects of multiplexing the Inter-cell Departure Time (IDT) distribution of the reference source was measured at the exit of each switch. This can be used in a comparison with an analytical approximation for a series of output buffered switches due to van den Berg and Resing [van den Berg92], which is reproduced below. The analysis shows that a CBR connection traversing successive switches where it is multiplexed with Poisson traffic at high load, asymptotically approaches a maximum perturbation. Let X_n be the inter-cell departure time of a given pair of reference cells from the n^{th} multiplexer. The process $(X_n, n = 0 \ldots \infty)$ is a Markov chain which is independent of the inter-cell departure times of other pairs of reference cells.

Now X_n is distributed as $1 + P(\lambda X_{n-1})$ where $P(\mu)$ is a Poisson distribution with mean μ, and here $\lambda = 1 - 1/T$. The model may be solved using a probability generating function for the X_n: let

$$h_n(z) = \mathbb{E}(z^{X_n}).$$

In [van den Berg92] it is shown that

$$h_n(z) = z h_{n-1}(g(z)),$$

where

$$g(z) = e^{\lambda(z-1)}$$

is the probability generating function of a Poisson random variable with mean λ. From this recurrence relation expressions for the mean and variance of X_n can be derived. In particular, after n switching stages the variance of the inter-cell time distribution is given by

$$\mathrm{Var}(X_n) = \frac{T^2(T-1)}{2T-1}\left(1 - \left(\frac{T-1}{T}\right)^{2n}\right). \tag{6.1}$$

It is clear from Equation (6.1) that the convergence of $\mathrm{Var}(X_n)$ is slow if T, the source period, is large. Thus a low bitrate source should take longer to reach its maximum perturbation. Also, the absolute value of $\mathrm{Var}(X_n)$ is greater for a reference connection with a larger value of T – thus a low bitrate source should experience a greater variance in inter-cell times than a high bitrate source. $\mathrm{Var}(X_n)$ differs significantly from the I.371 CDV measure which depends on the order in which cells arrive and only records cells which are "early" i.e. which exceed the peak rate. The sections below attempt to relate the two measures, on the basis of measured results.

6.3.3 Bounds on the Maximum CDV

The work of van den Berg and Resing has been extended by Kelly and Key, who derived a large deviations bound for the buffer requirement of the reference stream [Kelly94]. Studying the renewal process of the above model after the reference stream has passed through an infinite series of heavily loaded queues, they derived an expression for the tail behaviour of a $GI/G/1$ queue (representing a playout buffer at the sink) serving arriving reference cells.

Denote by W the unfinished work in the queue found by an arriving cell. Then the stationary distribution of W is given by

$$a_1 e^{-\kappa w} \leq \mathbb{P}(W > w) \leq a_2 e^{-\kappa w} \qquad w \geq 0 \tag{6.2}$$

for constants $a_1, a_2 \leq 1$, and $\kappa > 0$. The buffer models a leaky bucket of depth b and output rate a^{-1}. Using the bound of Equation (6.2) it can be shown that to ensure

$$\mathbb{P}(W > w) \leq e^{-\gamma},$$

it is sufficient that

$$a = -\frac{w}{\gamma} \log h(e^{-\gamma/w}). \tag{6.3}$$

Here $h(z)$ represents the renewal process after an infinite number of stages. For given γ and w, this fixes a and then $b = w/a$. Using $h_n(z)$ permits bounds to be calculated for the buffer after n stages of switching. For Bernoulli traffic the bounds are even more conservative: Equation (6.3) becomes

$$a = 1 + \frac{1}{\kappa} \log[T - e^{-\kappa}(T-1)]. \tag{6.4}$$

To lose less than one cell in e^{γ} corresponds to the choice $\kappa = \gamma/(ab)$.

The CDV is given by $w = ab$. Fixing the maximum tolerable loss rate γ permits the leak rate of the bucket to be determined. Alternatively, fixing the leak rate at $1.05T$, as used in this study, w represents an upper bound on the CDV with loss rate γ.

Finally, note that [Matragi94] offers a model of CDV which is more general than the above, using a model of bursty background traffic in which the burst size distribution is determined by a Markov chain. The analytical techniques permit the calculation of the probability generating function of the marginal density of the departure process for the reference traffic from each node, which is used as the input traffic to the following node.

6.4 Traffic Sources

In each experiment a single reference connection was multiplexed with one of several types of background traffic. The buffer space available for queueing was limited to 2000 cells, which was sufficient to ensure that the experiments were loss free[2].

6.4.1 Traffic Generators

The cell streams required for the experiments were generated by trace-driven traffic generators, located on FPC3 port controllers at the inputs of a Fairisle switch. The traffic generator is software based and was implemented by the author within the Wanda micro-kernel. It consists of two parts: a user-space application which is responsible for loading the trace and implementing a control interface, and a kernel-based generator, hand coded in assembler, which is responsible for transmission of the cell stream whose activity is described by the trace. The user-space component of the generator exports an RPC-based interface which allows it to be controlled remotely. The generator can be instructed to load a trace, to set up a connection to a network sink, and to start or stop transmission.

The trace format used is simple, encoding only the inter-cell time between cells of the stream to be generated. Memory restrictions on the FPC3 limit the trace length to 1.5 million cells for a typical trace with large inter-cell times, however the traffic generator provides several options for sub-sampling, or replaying the trace with either a fixed or random delay (depending on the type of traffic required). This removes the problem of correlation in the background traffic at multiples of the trace length. Each cell is stamped with a departure time by the hardware when transmitted, and the generator can also insert sequence numbers and other information into the cell payload. The generator was extensively tested and tuned to ensure that it is highly accurate – typical error in a generated CBR stream is less than 0.5 slot times.

The following subsections describe the traces used to generate both the reference connection and background traffic cell streams.

6.4.2 Reference Traffic Sources

The primary aim of these experiments was to investigate the cell-scale CDV effects of different types of background traffic on a single CBR reference connection. CBR connections are representative of synchronous services adapted onto an ATM bearer, for example an E1 (2 Mb/s) trunk. Table 6.1 shows for each source

[2] It was an express aim of the study to avoid cell loss, which is dealt with later in Chapter 8.

its name[3], its (AAL) bitrate, the number of cells it transmits per second, the period T between emitted cells (in cell times on a 100 Mb/s TAXI link) and the load ρ_{ref} it places on the link.

Reference Source	Bitrate (Mb/s)	Cells/s	T	ρ_{ref}
B	2	5461.3	41	0.024
E	10	27306.7	8	0.120

Table 6.1: Reference source parameters

For the output-buffered configuration and for each of the switch sizes $D = 4, 8, 16$ and 32, the reference source parameters were scaled to ensure that the total offered traffic from the reference source remained the same proportion of the link bandwidth. In each case the period T was chosen to be the closest integer to the actual required value, so some small differences are to be expected when comparing results across different values of D, for the same source.

6.4.3 Background Traffic

Key parameters of the background traffic which critically affect the CDV experienced by a reference connection include its peak rate and its average or mean rate. The ratio of the two is sometimes called the "burstiness" of the traffic[4]. The parameters and burstiness for each of the three types of background traffic used, namely Bernoulli, AVA-200 ATM Camera video, and LAN interconnection data traffic, are given below.

6.4.3.1 Bernoulli Traffic

Bernoulli traffic is the discrete time analogue of a Poisson process, and is frequently used in analytical models of ATM multiplexers. Bernoulli traffic was used to facilitate comparison of the experimental results with the predictions of analytical models and simulations, although it is not representative of any particular ATM traffic load.

In each experiment the *total* load ρ_{tot} placed on each link in the system was one of 3 values: 0.5, 0.8 or 0.95. The load placed on the link by the Bernoulli background traffic is given by

$$\rho_b = \rho_{\text{tot}} - \rho_{\text{ref}}$$

[3]These names will be used repeatedly throughout the text. The early experimental work used a set of CBR sources named A to E, of which only two (B and E) were selected for the extended work. Because of this, results for only two sources are presented.

[4]Other definitions use the variance to mean ratio or mean burst length [Bae91].

where ρ_{ref} is the load placed on the link by the reference connection. The Bernoulli traffic and the reference stream, when multiplexed, combine to yield the correct load.

6.4.3.2 ATM Camera Traffic

A trace of AVA 200 ATM Camera video[5] was made and sections superimposed to form a multiplex representing the superposition of 20 and 30 such devices. The camera was set up to transmit 25 frames per second, JPEG compressed, 24 bits per pixel colour video from normal BBC TV transmission over Fairisle, where the trace was recorded. The picture size was 48×33 tiles (each 8×8 pixels); with each tile compressed using a JPEG quantisation factor of 32, and transmitted with a single cell of overhead per tile. The amount of information per tile differs according to the video content. The ATM Camera's spacing policer was disabled, allowing back-to-back cells to be transmitted. The peak rate of each source is thus 100 Mb/s and the mean rate approximately 2.5 Mb/s, yielding a burstiness of about 40[6]. The trace represents about 1.5 minutes of video and includes both action and static scenes, giving a wide range of tile contents[7]. The total mean load with each source is given in Table 6.2. Figure 6.5 shows a section of a trace of the measured load of a multiplex of 20 and 30 ATM Camera sources. Each point on the graph is the number of cells which arrived during that 30 ms interval. Notice how the traffic becomes smoother when a higher number of sources is multiplexed. The traces show strong correlations over multiples of the frame time, and a visual appearance of correlation over several seconds, due to intra-scene correlation of the superimposed traces. The total trace length is about 15 seconds for 20 sources and about 10 seconds for 30 sources.

Reference Source	ρ_{tot}(20 sources)	ρ_{tot}(30 sources)
B	0.58	0.83
E	0.68	0.93

Table 6.2: AVA-200 ATM Camera traffic parameters

6.4.3.3 Data Traffic

A substantial portion of WAN ATM traffic is likely to be "LAN interconnection" traffic linking remote LAN sites, and consequently is must be taken into consideration in performance models. Such traffic is packet-based and may be extremely

[5] An early version of the device is described in [SRG94].

[6] When averaged over the whole trace.

[7] JPEG does not compress action scenes differently from static scenes.

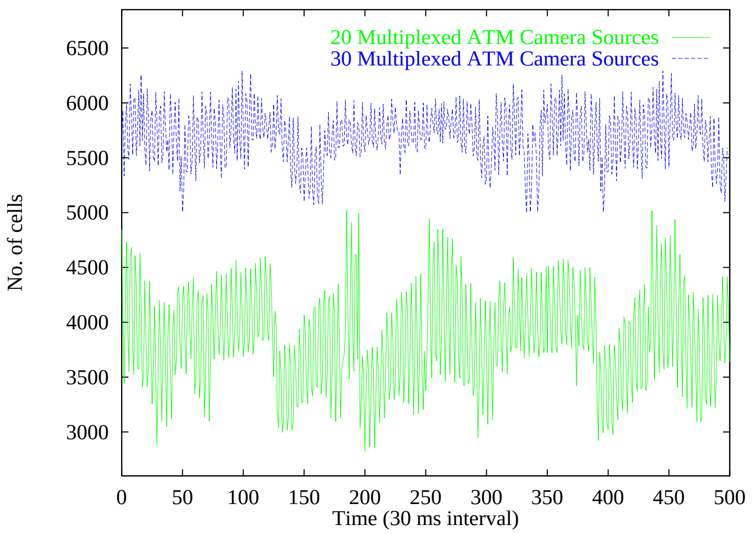


Figure 6.5: Measured load for multiplexed ATM Camera sources

bursty. The data sources used in these experiments were constructed from the Bellcore Ethernet traces [Leland93], which contain the measured activity on a busy Ethernet over tens of hours. Two traces were constructed from the measurements of wide area traffic by multiplexing 50 and 60 independent segments of the Bellcore traces. Each individual segment represents the contribution of a single LAN interconnection traffic source, and its traffic contribution was peak rate limited to half the link rate (50 Mb/s), to reduce the burstiness of the aggregate load. Without this the background traffic is too bursty for the available buffer space at a high mean load. The two traces used differ significantly – the trace with 60 Ethernets has far bigger bursts and a higher mean load per segment which contribute to the disproportionate increase in load shown in Table 6.3. The average burstiness of an Ethernet segment in each case is 60.1 (50 sources) and 48.7 (60 sources). Figure 6.6 shows the measured load for a short period of the data traffic composed from 60 multiplexed Ethernet segments. The graph shows extreme burstiness of this type of background traffic. Although much wide area traffic will be TCP-based and therefore sensitive to network congestion, experimentation with a large number of dynamically controlled sources was not possible.

6.4.3.4 Summary

To enable comparison of this work with other studies, detailed statistics for the background traffic types are presented. Table 6.4 presents the measured properties of the inter-cell time distributions for each type; these are also displayed

Reference Source	ρ_{tot}(50 sources)	ρ_{tot}(60 sources)
B	0.44	0.64
E	0.55	0.75

Table 6.3: LAN interconnection traffic parameters

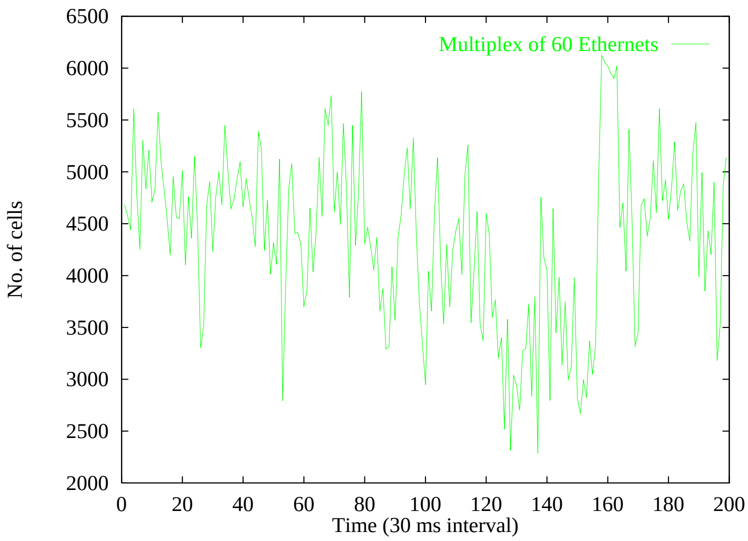


Figure 6.6: Measured load for 60 multiplexed Ethernet sources

graphically in Figure 6.7 (a). Of particular interest is the shape of the distributions for the data traffic: a large number of cells arrive back-to-back or close together (packets of traffic), and there is an exponential tail which is extremely long. Consequently this traffic is extremely bursty. Both multiplexed AVA traffic distributions have short tails. Since the CDV is influenced by the burstiness of the background traffic, Table 6.5 presents properties of the burst-size distribution for each type, where a burst is defined to be a group of cells separated by 5 or more

Name	No. sources	Mean	Std. Dev.
Bernoulli	load 0.95	1.206	1.149
AVA	20	2.194	2.077
AVA	30	1.459	1.135
Data	50	2.350	6.657
Data	60	1.611	3.677

Table 6.4: Inter-cell statistics for background traffic

Name	No. sources	Mean	Std. Dev.
Bernoulli	load 0.95	1.726	1.128
AVA	20	8.668	15.605
AVA	30	29.578	150.01
Data	50	27.128	43.032
Data	60	52.368	150.977

Table 6.5: Burst size statistics for background traffic

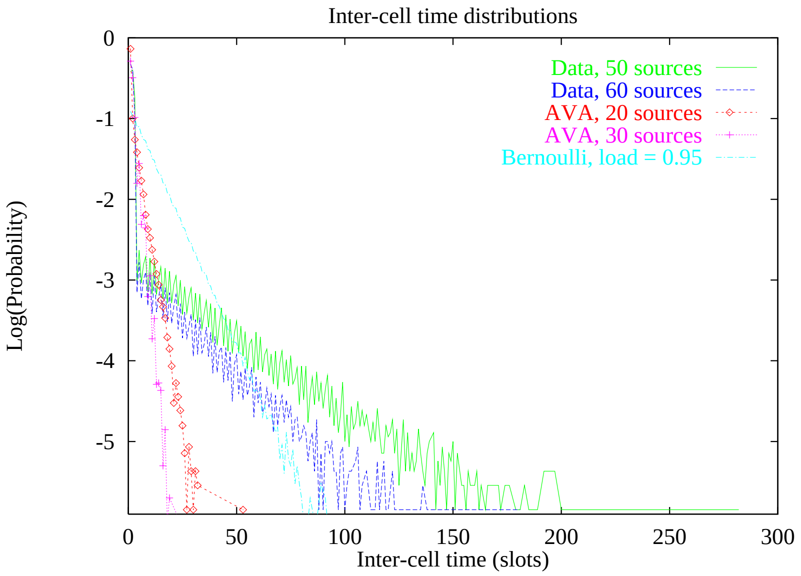
cell times. The complementary Cumulative Distribution Function (CDF) of the burst size distributions is shown in Figure 6.7 (b). The graph shows clearly the effects of multiplexing bursty sources – the burst size distributions have extremely long tails compared to Bernoulli traffic, and point to potential performance problems when these traffic types are multiplexed with CBR reference streams.

6.5 Results and Discussion

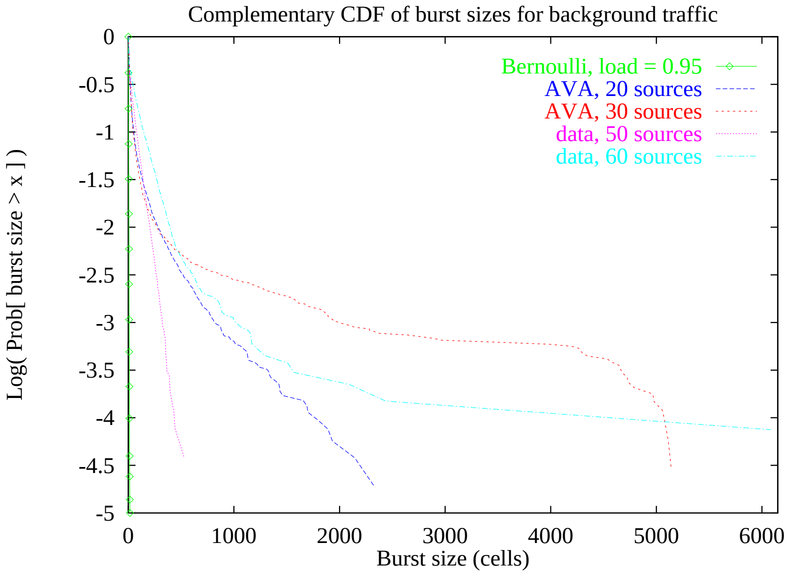
This section investigates the influence of several experimental parameters on the CDV experienced by CBR reference sources. For each source s the results are presented in units of T_s, the source period. To convert them into link slot times simply multiply by the value of T_s given in Table 6.1. The results are typically presented on a log scale, to enable the tails of the distributions to be seen. In each graph the key describes the parameters for a particular result. The parameter D gives the switch size, and the terms "blocking" and "non-blocking" refer to the switch type (equivalently input or output buffered). For the video and data background traffic the number of sources is the number of video streams or Ethernet segments respectively which were multiplexed to form the background traffic. The resulting total loads for these values are given in Tables 6.2 and 6.3. In all cases where not explicitly stated, a switch size of $D = 8$ was used.

6.5.1 Total Load

Figures 6.8 (a) and (b) show the effect of increasing Bernoulli load on the two reference sources, with a single stage of non-blocking switching. The increase in CDV with load is non-linear, although each curve itself is approximately log linear where sufficient data points occur. Note that although the relative CDV experienced by source E is worse than that experienced by source B (in units of T for each source), the absolute values of the CDV (in slot times) experienced by source B and E are almost the same at high load. Source B at a load of 0.95



(a)



(b)

Figure 6.7: Inter-cell time distributions and complementary CDF of burst sizes for background traffic

suffers CDV less than T_B, whilst for source E the CDV is about $3.5T_E$ for the same load.

These results, together with others not presented here, suggest that the CDV for high rate reference sources rapidly increases, whereas for low rate sources the

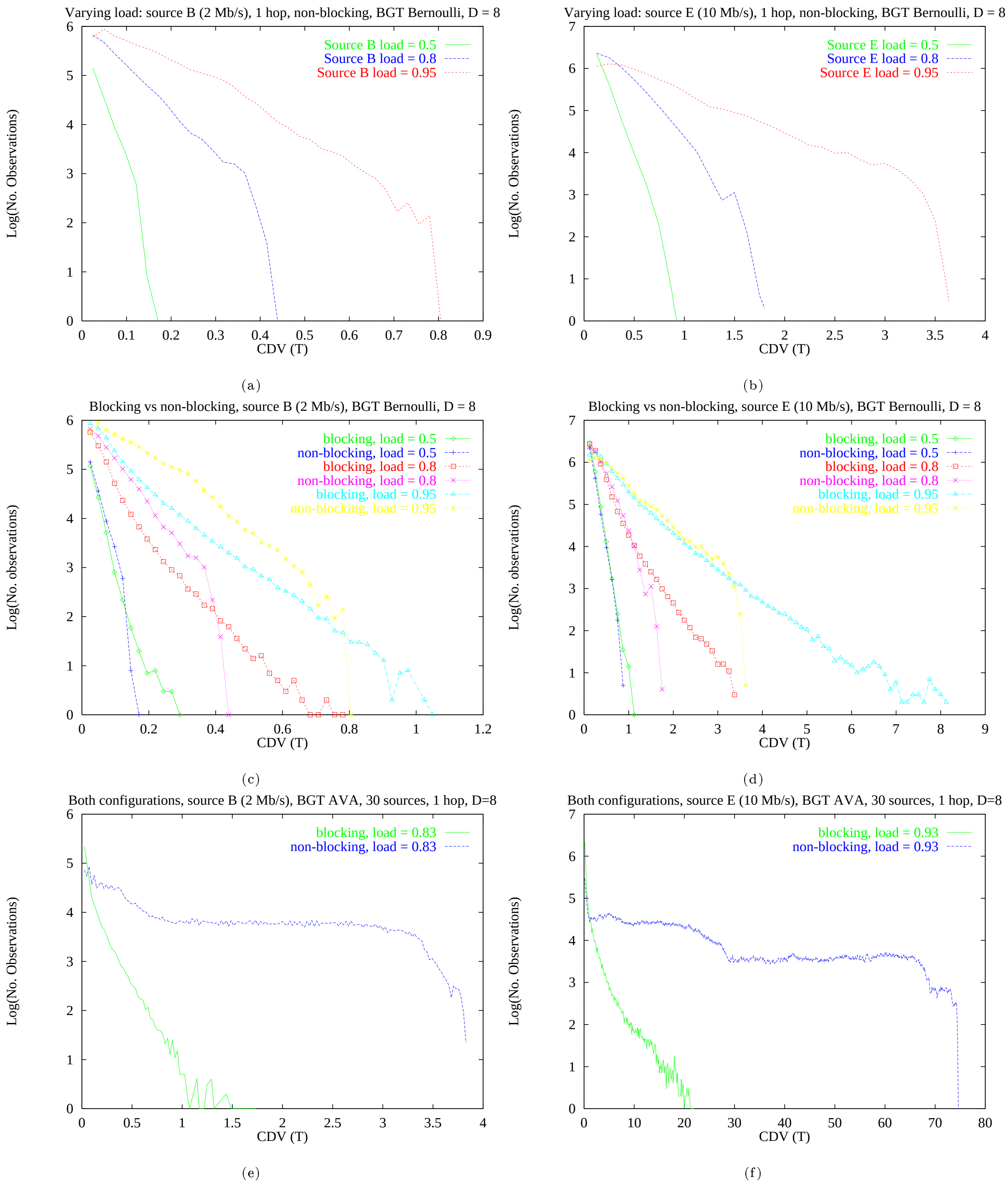


Figure 6.8: CDV caused by a single stage of switching, CBR sources B and E

rate of increase is much lower. This is studied further in Section 6.6 below.

6.5.2 Switch Type (Blocking or Non-blocking)

The comparison of the blocking and non-blocking switch configurations is complicated by Fairisle specific hardware features which directly effect the CDV under the blocking configuration. They include the fabric arbitration mechanism (round robin between contending ports) the relative rates of the input links and the fabric, and a small FIFO at each output which performs rate adaptation between the fabric and the output link. Because of this the results for the blocking configuration may not be representative of the performance of other input buffered switches, although all equipment will include similar features. In the results reported below the reference stream entered the switch from a dedicated input port and the background traffic was symmetrically distributed over the remaining input ports of the switch.

Figures 6.8 (c) and (d) compare the CDV sustained by each reference source under the blocking and non-blocking configurations, again using Bernoulli background traffic at different loads, for a single stage of switching and a switch of size $D = 8$ ports. The results are consistent in all cases – the tail CDV value under the non-blocking configuration is lower than using the blocking configuration. The CDV histogram for the non-blocking configuration has a truncated tail in all cases (as observed previously), indicating that the CDV experienced is better with output than input buffering.

This result seems counter-intuitive – the input buffered switch has a separate queue for each traffic stream (one at each input) and the round robin scheduling mechanism implemented by the fabric might be expected to ensure that the queueing delay experienced by a reference cell at its input would be strictly bounded. This is true only in the case of the 4×4 fabric, where if all ports simultaneously contend for a single output then the round robin mechanism will service them in order, guaranteeing a maximum queueing delay across the fabric. For larger input buffered switches the switching fabric is internally blocking, and contention for the output of the fabric can also cause blocking. The CDV is further influenced by two Fairisle-specific features: the fabric operates at a rate 1.2375 times higher than the line rate to partially overcome the head-of-line blocking problem [Black94b]. In addition a small output FIFO is used to accommodate the rate difference between the fabric and line. The FIFO applies back pressure across the fabric when half full, complicating the blocking mechanism. A reference cell which succeeds in arbitration across the fabric may find the output FIFO full, and will have to begin contention again; the resulting delay will cause CDV. The effect of fabric blocking on performance is studied in more detail in Chapter 8.

6.5.2.1 Effect of load

For source B the difference between the tail CDV values under the two configurations does not change significantly with increased load. Using source E, however, the load has a far greater effect. For a load of 0.95 the CDV experienced using the blocking configuration is double the CDV using the non-blocking configuration. The difference in CDV under the two configurations is greater for source E than source B. One reason for this is that low rate streams see an effective priority across the Fairisle switch fabric because the round robin arbitration mechanism will favour an input which has not recently succeeded in contention for the output port, hence cells from the lower rate source B are subject to a lower probability of fabric blocking. This phenomenon was also observed by Hayter in experiments on the DAN workstation [Hayter93].

6.5.2.2 Bursty Background Traffic

Whilst the above arguments hold for Bernoulli traffic, with bursty background traffic the picture is somewhat different, as can be seen in Figures 6.8 (e) and (f) which compare the two configurations using a high load of AVA background traffic. In this case the non-blocking configuration performs worse because cells from the reference connection, arriving at the output buffer will tend to be grouped by the bursts of background traffic. The blocking configuration divides its total load into several queues (one at each input), and arbitrates more fairly for space in the small output FIFO. This results in a more even distribution of the available output bandwidth amongst the input ports. Because of this, non-blocking switches seem to offer better performance under high loads of bursty background traffic, in spite of the unpredictable delays due to contention across the fabric and for the output FIFO.

6.5.3 Switch Size

Figures 6.9 (a) and (b) investigate the effect of the switch size on the CDV experienced, for both the blocking (a) and non-blocking (b) configurations, for a single stage of switching. For the blocking configuration results are given only for switches of size 4 and 8. The loads were scaled by the appropriate switch size in the experiments with the non-blocking configuration. For both configurations (and for both reference sources) it is clear that an increase in switch size increases the CDV experienced by the reference source. At low loads this also holds although the difference is not significant. For the blocking configuration the increase in switch size from 4 to 8 ports entails a change from a crossbar to a delta interconnection network. The additional fabric stage causes greater variability in the queueing delay seen by a reference cell. For the nonblocking

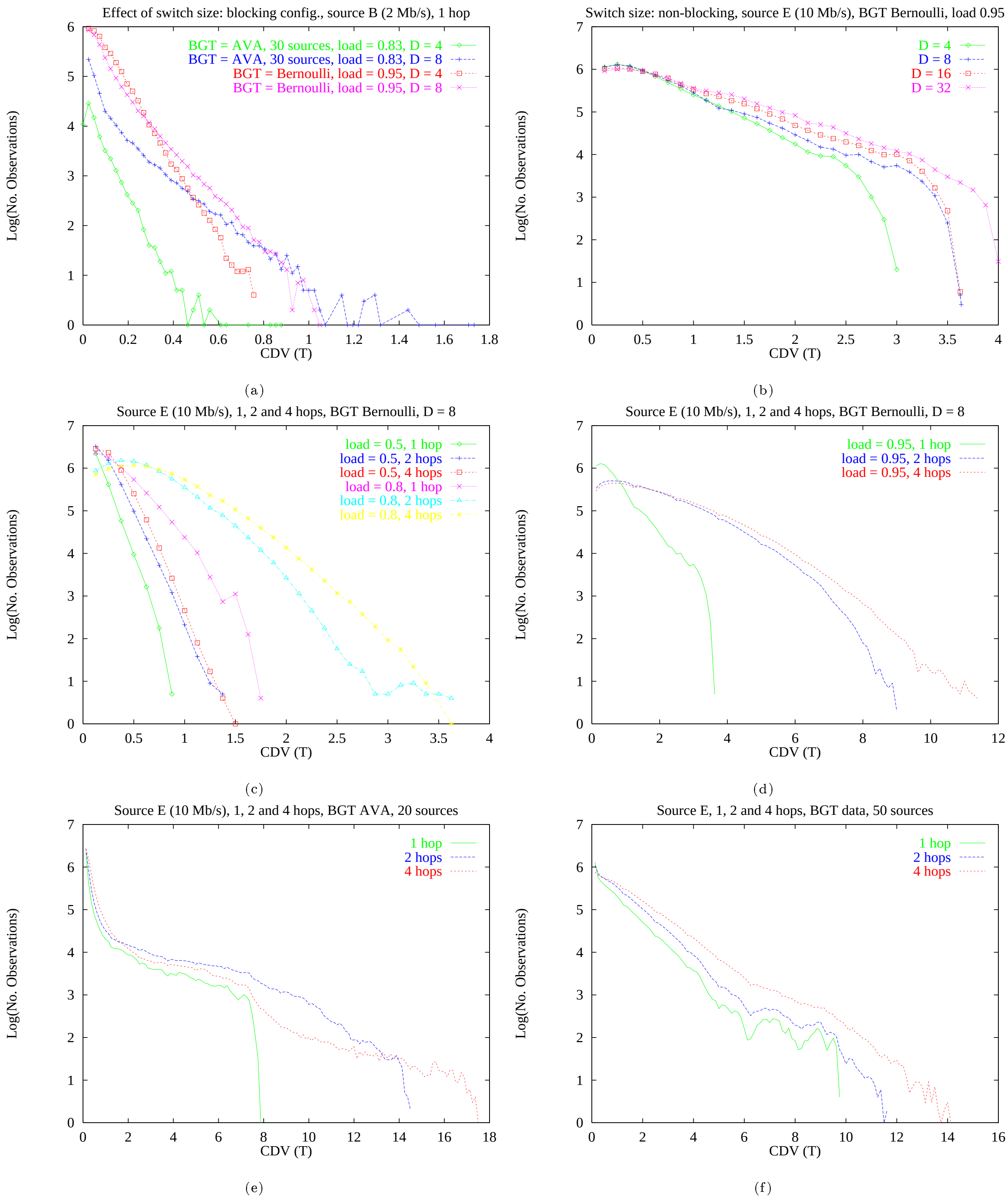


Figure 6.9: CDV results to 4 switching stages, for CBR sources B and E

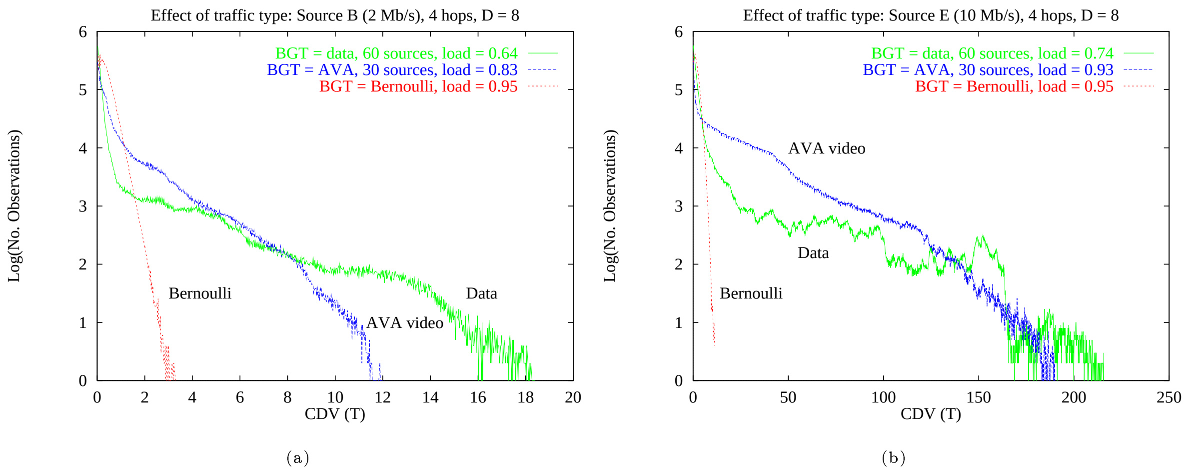


Figure 6.10: CDV after 4 hops, sources B and E, varying background traffic

configuration the difference is also easily understood: given the same load, in the $T-1$ slots between cells of the reference connection, on average the same number of background traffic cells would be expected, irrespective of the switch size. However, during the cell time (on the link) during which the reference connection transmits a cell, up to D cells from the background traffic can simultaneously be switched by a switch of size D. Thus, as D increases, a greater absolute number of cells are expected to interfere with the cell from the reference connection, causing greater CDV.

6.5.4 Number of Switching Stages

Figures 6.9 (c-f) compare the CDV experienced as a result of 1, 2 and 4 stages of switching, under the non-blocking configuration, for reference source E. Using Bernoulli traffic, and a load of 0.5, Figure 6.9 (c) shows that with an increase from 1 to 2 stages of switching the change in CDV is small, and appears to increase no further with an additional two stages of switching. With a higher load of 0.8, the results are similar – an initial rapid increase in CDV from 1 to 2 stages, followed by a much lower increase when using 4 stages. The CDV appears to grow in an asymptotically bounded fashion which is dependent on the total load. This is confirmed with a further increase in load to 0.95, shown in Figure 6.9 (d). The initial increase in CDV is again greater than in the previous two cases, but the CDV appears bounded above with an increase in switching stages.

Figures 6.9 (e) and (f) investigate the same relationship, using low loads of bursty background traffic, namely 20 ATM Camera (AVA) sources (e) and

50 data sources (f). Here again it appears the relationship holds – an initial rapid increase using AVA traffic from 1 to 2 stages, followed by a much smaller increase to 4 stages. The total load using the data traffic is sufficiently low that the increase is small in each case. For low loads the increase in CDV with additional switching stages is small and bounded, whilst for high loads the increase is far greater and takes more switching stages to stabilize. The results also suggest that for each particular load and type of background traffic the CDV is bounded above and will not continue to increase indefinitely with additional switching stages. Similar observations hold for high loads, however the CDV incurred is significantly greater.

Finally, Figure 6.10 compares the CDV for sources B and E using different background traffic types, after 4 stages of switching. The most important feature of this graph is the apparent inability of the Bernoulli model to predict the CDV values experienced with real traffic. This suggests caution when using Bernoulli models in obtaining performance bounds for ATM networks.

6.6 Addition of Further Switching Stages

The results of Section 6.5 suggest that the CDV experienced by a CBR reference connection is bounded, and does not continue to deteriorate indefinitely with additional switching stages. The maximum CDV appears to be bounded above irrespective of the background traffic type. For low total loads the increase in CDV with additional switching stages is small, whilst for high loads the increase is far greater. For lower bitrate reference sources the rate of increase in CDV is lower and takes more switching stages to stabilize.

The process causing the development of the CDV can be more easily understood with the help of the approximation of Equation (6.1). The model predicts that a low bitrate source should take longer to reach its maximum perturbation than a high rate source. Also, the absolute value of the variance $\mathrm{Var}(X_n)$ is greater for a reference connection with a larger value of T – in the context of the earlier results this means that the lower bitrate reference source should experience a higher variance in the inter-arrival times of its cells at a given stage than the higher bitrate source. This section investigates further the effect of the number of switching stages on the CDV experienced by CBR connections.

6.6.1 Experimental Method

The results reported above were measured from the transmission of 10^7 cells from the reference connection. The following experiments are based on measurements of the first 10^6 cells of the reference connection only. In each experiment the the

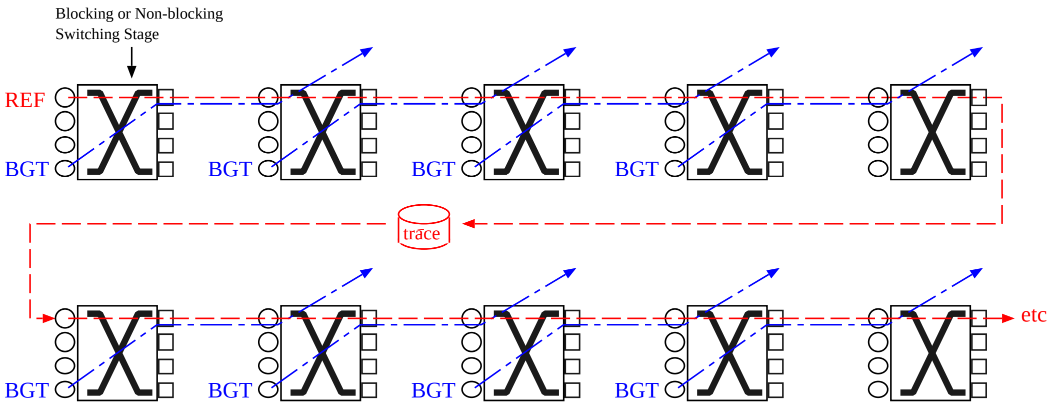


Figure 6.11: Extended switching topology for additional switching stages

absolute arrival times of the first 10^6 cells belonging to the reference connection were recorded after 4 stages of switching (the number of physical stages of output-buffered switching in the current Fairisle network). The trace was used as the input reference source to a second experiment, with the same background traffic, logically extending the path length by 4 stages. Repeating this process permitted the total path length to be extended to 12 stages, with performance measures taken after each stage[8]. Figure 6.11 illustrates the method graphically.

6.6.2 Results

The following subsections present the results for the experiments using the extended topology.

6.6.2.1 Inter-cell Time Distribution

Figures 6.12 (a) and (b) compare, for reference sources B and E, the measured variance in the inter-cell departure time from each switching stage and the predicted variance from the model of Equation (6.1). The approximation assumes Poisson background traffic and high load (close to 1). The results show for the different types of background traffic and their respective loads, the measured variance in IDT. Notably, the measured results using Bernoulli background traffic with a load of 0.95 compare well with the predicted values, particularly for source E. The difference between the measured and predicted values results from the analytical modelling assumption of a load of 1, and the experimental load

[8]A recent trial between Cambridge and the University of Twente, in the Netherlands, involved about 12 stages.

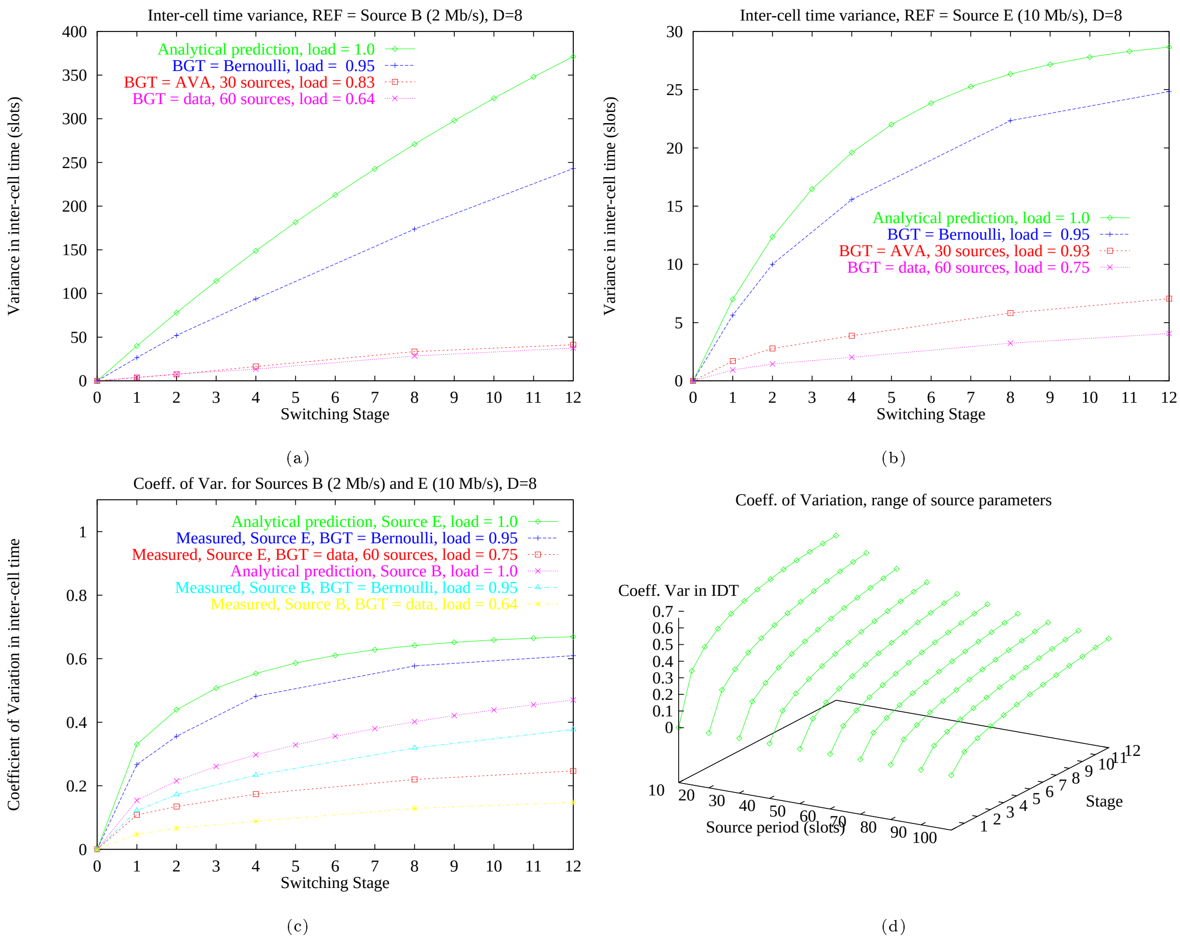
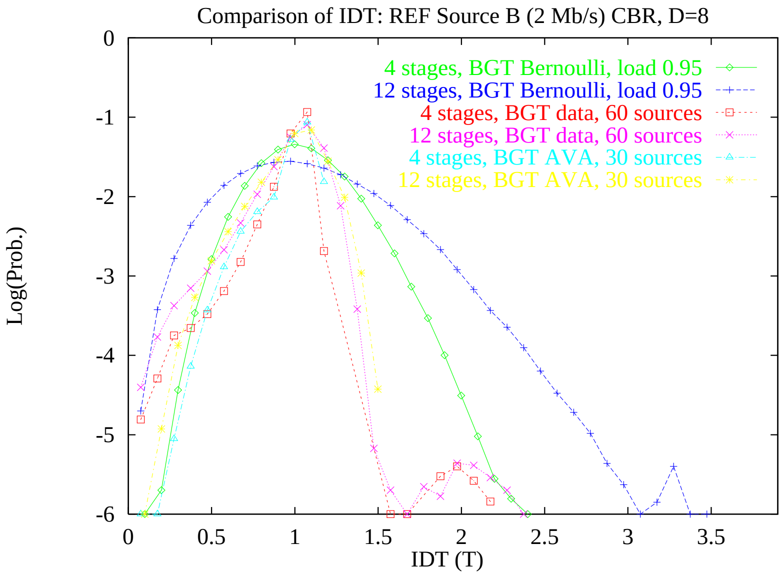


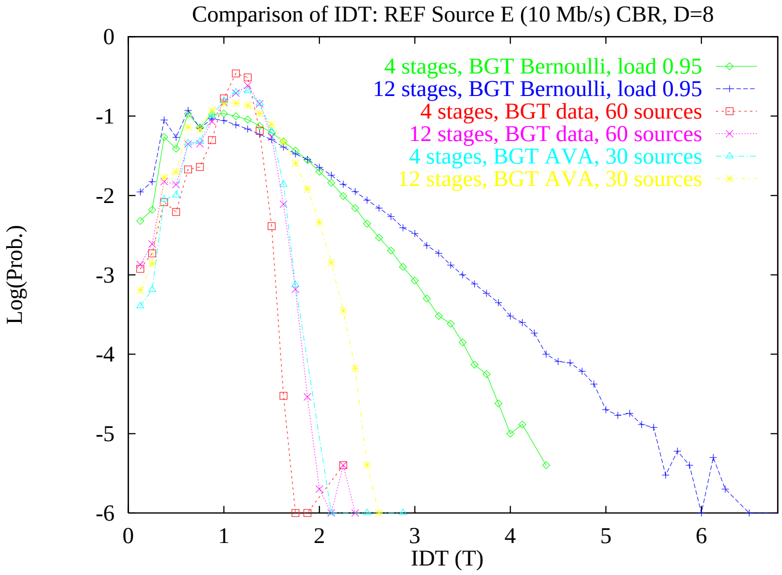
Figure 6.12: Variance (a,b) and coefficient of variation (c) in inter-cell time, for CBR sources B and E and for a range of source parameters (d), over 12 switching stages

of 0.95. Comparison of the two reference sources multiplexed with Bernoulli traffic at a total load of 0.95 shows that the variance is far greater for source B. Figure 6.12 (c) shows the same results, expressed as the coefficient of variation. In this normalised form the two sources can be directly compared: for the same load of 0.95, the coefficient of variation for source E is almost double that for source B. The higher rate source is affected by multiplexing more rapidly, and reaches a maximum perturbation which is relatively worse than for the lower rate source. Figure 6.12 (d) shows the coefficient of variation for a wide range of source parameters, confirming the relatively worse performance for high rate connections.

The measured results also show that for more bursty background traffic, the variance is lower than with Bernoulli traffic, for the same load (e.g. source E,



(a)



(b)

Figure 6.13: Comparison of the inter-cell departure time distribution for sources B and E after 4 and 12 stages, with Bernoulli, data and AVA background traffic

with Bernoulli and AVA traffic). Figure 6.13 compares the log of the IDT distributions for reference sources B and E after 4 and 12 stages of switching, for Bernoulli, data and AVA background traffic. The tail of the IDT distribution with the bursty background traffic types is extremely short, explaining the lower

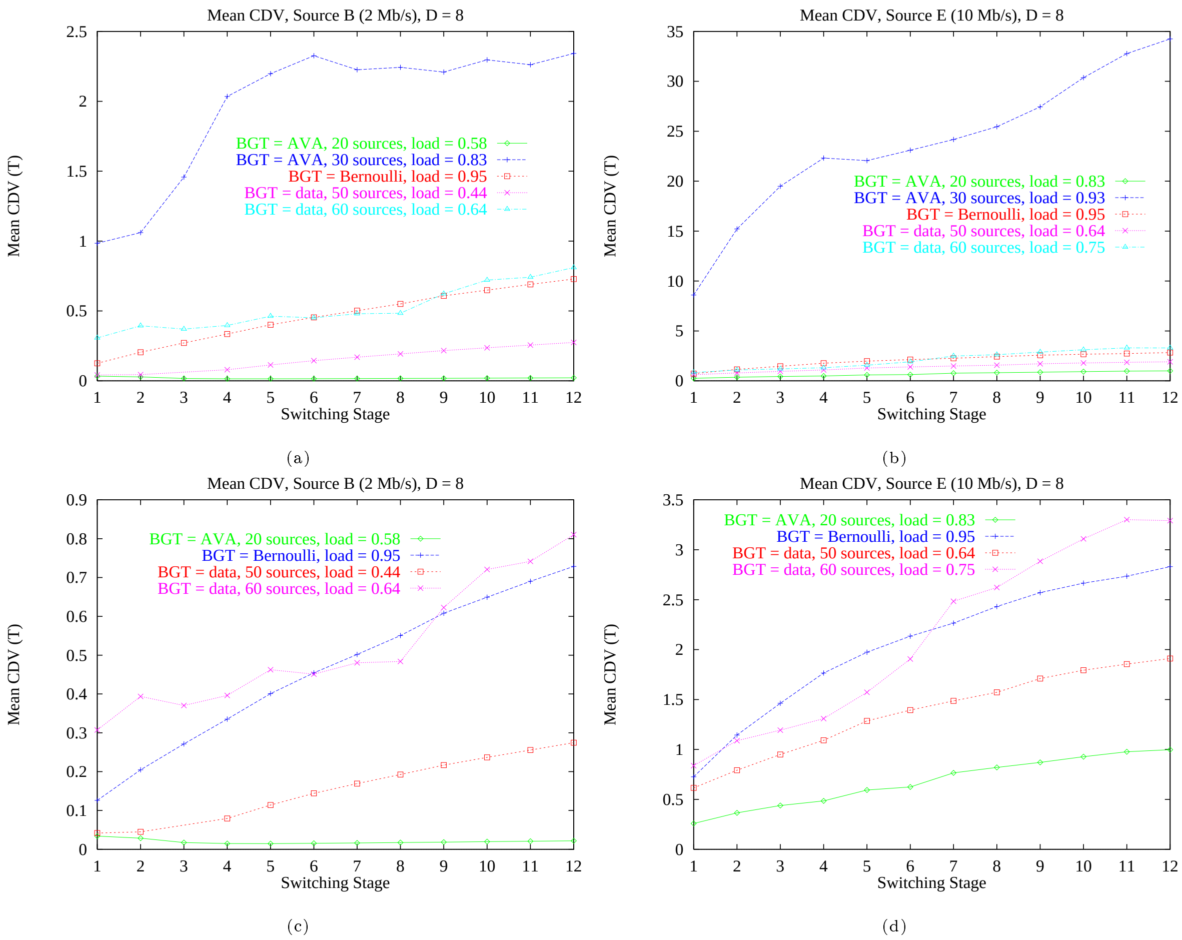


Figure 6.14: Mean CDV, CBR sources B and E, for 12 switching stages

observed inter-cell time variance of these distributions than for the Bernoulli traffic, in Figure 6.12. In spite of this, the bursty background traffic types introduce correlation into the reference stream. This is shown in Appendix C.

6.6.2.2 Development of CDV over Successive Switching Stages

The approximation shows that the variance in inter-cell times is bounded, and consequently one might hope that the CDV will develop in a similar way. Examination of the mean CDV experienced by each CBR reference source as it proceeds through the network confirms that this is the case. Figures 6.14 (a) and (b) display the mean CDV experienced by the two CBR sources, at each stage in the network. The same results are shown in Figures 6.14 (c) and (d), omitting the high AVA load, to show the other results in more detail. For both sources the AVA traffic with 30 multiplexed sources behaves the most erratically. Omitting

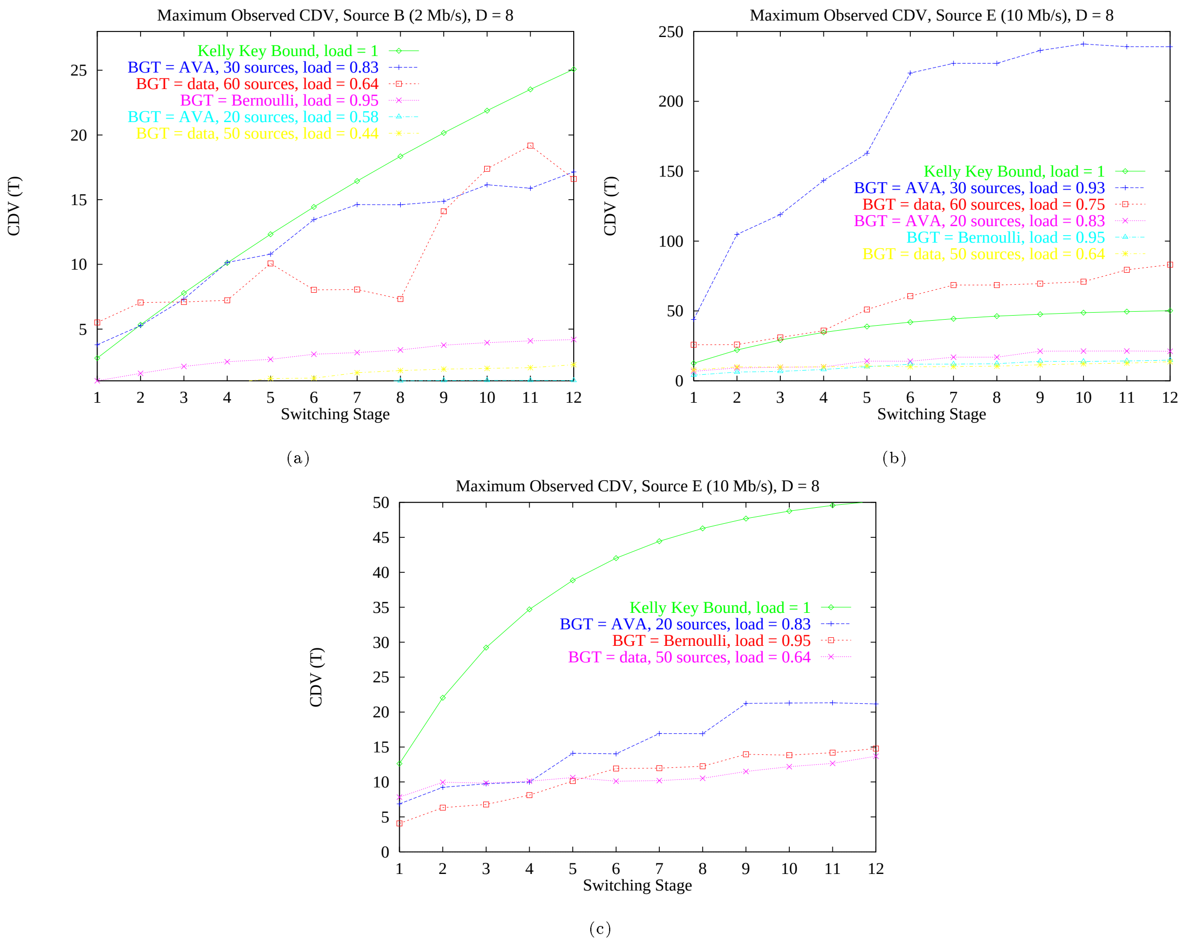


Figure 6.15: Maximum observed CDV, with Kelly Key bound, CBR sources B and E, for 12 switching stages

the high AVA load shows that for source B the CDV increases almost linearly with switching stages, and slowly approaches its asymptotic limit for each type of background traffic. Source E develops CDV more rapidly because it represents a higher proportion of the total link load. In both cases the mean CDV of the source when multiplexed with the high data load is erratic, probably due to the extreme burstiness of this traffic.

The tail of the CDV distribution is not as well behaved[9]. Figures 6.15 (a) and (b) show the evolution of the maximum CDV value measured for each type of background traffic and for each CBR reference source, together with the Kelly Key bound of Equation (6.3). For source B the Kelly Key bound contains all measured CDV values, however for source E, the bound does not contain the observations

[9]Note, however that these results are for a single point, rather than a distribution.

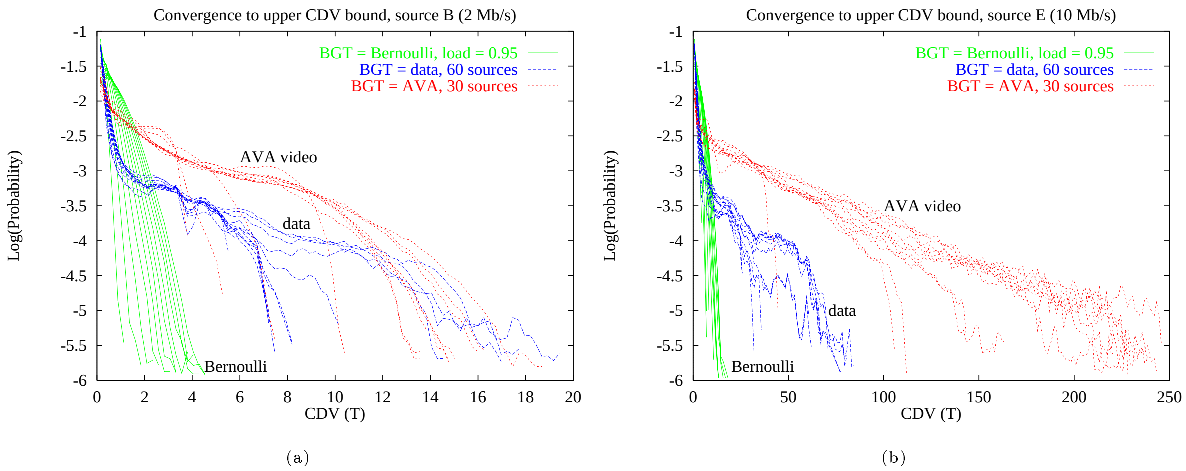


Figure 6.16: Convergence to limiting CDV distribution, sources B and E, over 12 stages, with different background traffic

using the high rate data and video background traffic. Note though that the total load is different in each case. Figure 6.15 (c) shows for source E the Kelly Key bound and the maximum observed CDV values for the lower rate sources and the Bernoulli background traffic. For Bernoulli traffic, the bounds are conservative: they contain the measured value for a total load of 0.95. Unfortunately it is extremely difficult to operate the Fairisle switch at a load close to 1, owing to the exponential growth of the queue length. In their paper, Kelly and Key note that the bounds of Equation (6.3) suggest that lower rate sources require larger buffers at the network output to ensure the same loss rate. The two bounds and the measured results confirm this. For almost the same load, source E with Bernoulli and AVA traffic shows that the key parameter influencing the CDV is the variance of the background traffic, as suggested in [Matragi94].

In all cases except for the high loads of data and video traffic the measured values of the tail of the CDV distribution develop in a way which is dependent on the background traffic type and the total link load. In most cases the development is regular and seems to approach an upper bound. Figure 6.16 confirms that for each type of background traffic and each reference source a limiting distribution of the CDV exists. Each graph shows, for each high load of background traffic, a collection of curves indicating the development of the CDV with additional switching stages. Although the individual curves are not labelled, they generally progress (within each background traffic type) from left to right, for additional switching stages.

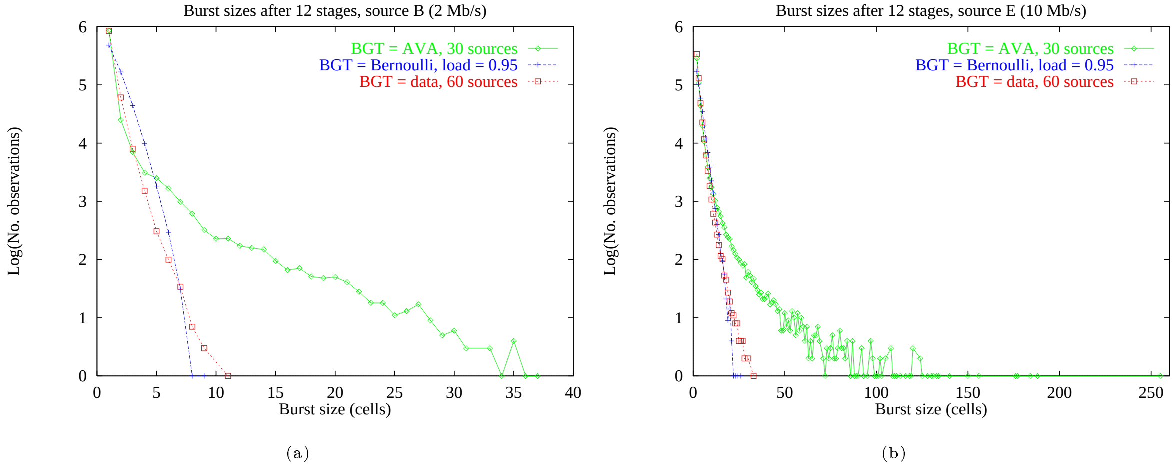


Figure 6.17: Comparison of burst sizes in the reference stream for sources B and E after 12 stages, with Bernoulli, data and AVA background traffic, using a burst gap of $1.2T$

6.6.2.3 Clustering of Reference Cells

Given that the bursty background traffic types give rise to a lower variance in inter-cell times for the reference stream, and cause less spread in the inter-cell time distributions pictured in Figure 6.13, one might well ask why they cause such high CDV, particularly given that their loads are lower than the Bernoulli traffic. The answer lies in their bursty behaviour: when multiplexed with bursty background traffic a reference cell arriving at the switch output will tend to find either a large queue of background traffic (and other reference cells), allowing the cell to "catch up" with other reference cells, or an almost empty queue causing little delay. This makes the reference traffic group into bursts of cells close together.

This hypothesis is borne out by Figure 6.17 which compares the burst sizes observed in each experimental sample after 12 stages of switching, for both reference sources. Here a "burst" of reference cells is defined to be a group of cells separated from the preceeding and following bursts by a gap of $1.2T$ cell times, where $T = 8$ for source E and $T = 41$ for source B. For only a small increase in the inter-cell time of two reference cells from T to $1.2T$ in each case (recall the narrow spread of the IDT distributions with bursty traffic in Figure 6.13) large bursts of reference traffic are formed by the bursty background traffic types, particularly for source E. Because the source period T is so much smaller for source E, for a given queueing delay of a reference cell, proportionally more reference cells will catch up to it than for the lower rate source B, and hence source E suffers a far worse CDV than source B. Note the similarity in shape between the graphs of Figures 6.16 and 6.17 – whilst not an accurate predictor of CDV, the burst size

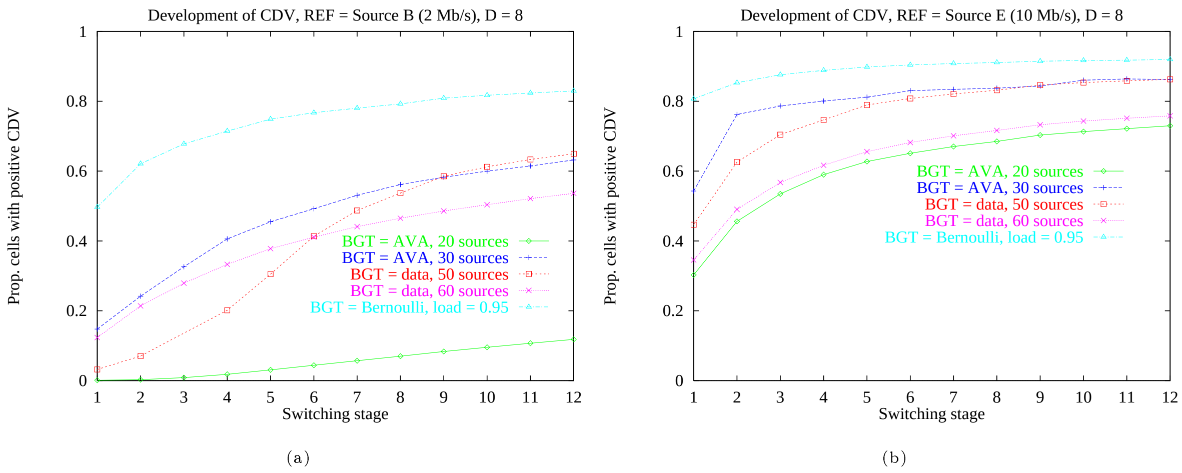


Figure 6.18: Proportion of cells which experience positive CDV, CBR sources B and E, for 12 switching stages

distribution indicates the relative effects of the background traffic types.

An important factor determining the size of the playout buffer required at the receiver is the degree of correlation in the perturbed reference stream. Appendix C presents results using the Index of Dispersion for Counts (IDC) [Sriram86], which show that the reference stream assumes the properties of the background traffic, and is thus correlated in the same way. The IDC can be used to parameterise a queueing model which enables the playout buffer size to be determined.

6.6.2.4 Proportion of the Sample with Positive CDV

This chapter concludes by examining the proportion of the total number of cells transmitted which incur positive CDV, for each of the sources and background traffic types, as the traffic proceeds through the network. Figure 6.18 shows that the proportion of cells which develop positive CDV (and are therefore liable to be discarded by the I.371 leaky bucket, subject to the tolerance parameter τ) is also bounded above. Unfortunately it also appears that the bound is relatively high, with about 80-90% of cells experiencing some CDV using Bernoulli traffic.

With the bursty background traffic types a lower proportion of the sample experiences CDV, in spite of the fact that the bursty background traffic causes greater CDV. This observation indicates that it is the relatively few large bursts of reference traffic at the tail of the burst size distributions in Figure 6.17 which cause the high CDV. Far more bursts of size 2 are observed using the bursty traffic than using Bernoulli traffic.

Another interesting feature of the graphs in Figure 6.18 is that the proportion of cells experiencing CDV is the same for the two loads of data traffic after 12 stages, in spite of the difference in their loads, and the difference in the CDV they cause.

6.7 Summary

The following conclusions can be drawn from this experimental investigation:

- Use of Bernoulli background traffic permits the results to be compared to analytical models. The measured results confirm the broad conclusions of the models, and indicate that the CDV experienced by a reference connection is bounded and dependent on its proportion of the total link load as well as the type of background traffic with which it is multiplexed.
- Both analytical models used, of the inter-cell time variance and the CDV bound, are conservative. The Kelly Key bound, however, fails to include all measured results using non-Bernoulli traffic.
- The variance of the background traffic process is critical in determining the CDV. Experiments with background traffic constructed from real traffic processes give a good indication of the limits of analytical models based on Bernoulli traffic, and show the degree of CDV which may be expected in a busy ATM network.
- The models suggest that unless all traffic is shaped in some way the CDV which CBR sources experience may be greater than they can tolerate. Chapter 8 investigates policies which reduce CDV.

6.8 Concluding Remarks

Since the CDV appears to be bounded above, it is probably unnecessary to implement policing mechanisms within the network. Certainly for CAC purposes it is necessary only to police a connection at its entry point to the network, since its CDV can be bounded from that point on. At inter-network boundaries operators may well choose to re-shape or police connections because the CDV will, in general, have grown significantly from the peak rate specified by the user. Alternatively, given that the CDV bound can be determined for a given traffic mix, the operator might choose to set the tolerance parameter τ according to the predictions of the analytical models. This would simplify the intra-network control mechanisms.

Finally, validation of the multi-stage model of [Matragi94] should be considered for future work, since this would permit more direct comparison with the measured results presented in this chapter.

Chapter 7

The Burst Expansion Phenomenon

7.1 Introduction

This chapter studies a process known as burst expansion – the mechanism whereby bursts of cells entering an ATM network tend to become smoothed by mixing with other traffic. An understanding of burst expansion is important for several reasons: badly behaved users who transmit bursts of traffic into the network could impact the QoS experienced by other connections. Similarly, bursts of traffic from a well behaved CBR connection which has become bursty as a result of multiplexing (as shown in the previous chapter) will affect the performance delivered to other connections. The rate at which bursts expand determines the damage (in terms of delay and loss) which they can inflict on other traffic. By measuring the CDV of bursty traffic it is possible to gauge the effects of an I.371 leaky bucket policer on a well behaved CBR connection which, as a result of multiplexing with other traffic, has become more bursty.

This chapter investigates burst expansion using the experimental platform based around the Fairisle network, described in previous chapters. The results of the study are compared to two analytical models, namely the approximation of van den Berg and Resing [van den Berg92], described in Chapter 6, and an analytical approximation from the COST 224 project, which is introduced in Section 7.2 below.

7.2 The Burst Expansion Rate

The COST 224 project developed an analytical approximation for burst expansion [Roberts92], pp 193, which aims to approximate the rate at which a burst of traffic will expand as it proceeds through a network of output buffered switches. Consider a periodic burst, the cells of which arrive at the multiplexer at time instants $S_i = iD, i = 0 \dots M$. Here D is, as before, the switch size. Assume that other arrivals to the multiplexer constitute a Poisson process with parameter ρ, and let $\lambda = 1/D$. Let $V(t)$ be the amount of unfinished work at time t. Now $V(S_0-)$ is distributed as the virtual waiting time in an $M/D/1$ queue with load ρ whereas $V(S_M-)$ is distributed almost exactly as that of an $M+D/D/1$ queue, just before the deterministic arrival of a CBR cell. As the burst passes through additional multiplexers its inter-arrival times become more random and so the unfinished work seen by the last cell can be estimated as that of an $M/D/1$ queue with load $\rho + \lambda$. The burst expands as the cells become separated by the background traffic. The expected increase in the total time spanned by the burst, from the time the first cell leaves a switch until the last cell leaves the switch, is called the expected increase in burst size $\mathbb{E}(\Delta B)$, expressed in slots per stage of switching. The pessimistic estimation for the expected increase of the burst size is given by

$$\mathbb{E}(\Delta B) \approx \frac{\lambda}{2(1-\rho)(1-(\rho+\lambda))}. \tag{7.1}$$

For example, with $D = 20, \rho = 0.9$ and $\rho + 1/D = 0.95$ the expected burst expansion is $\mathbb{E}(\Delta B) \approx 5$.

In [Roberts92] this approximation was compared to simulated results for a network with 100 switching stages, using the above parameters. In the simulation the position of each cell in a burst was traced over successive switching stages. The simulation results show an approximately linearly proceeding expansion and suggest that the bound in Equation (7.1) overestimates the expansion. The simulation yielded a measured mean expansion per multiplexer of 2.69, suggesting that the analytical approximation is almost two times too conservative.

To test the accuracy of the COST 224 model, the experiment was replicated using Fairisle. The same experimental parameters were used: an output buffered switch of size $D = 20$, Bernoulli background traffic of load $\rho_b = 0.9$ and using a periodic bursty reference stream with a burst size of 20 cells. Within the burst, cells are emitted with a period of 20 cell times (this will be referred to as the 1 in 20 source). The inter-burst gap was set to 1200 cell times, to allow the $Geo/D/1$ queue to stabilise between bursts, so that each burst can be regarded as an independent experiment. The total load during a burst is thus $\rho + 1/D = 0.95$. Figure 7.1 shows the bursty reference source graphically, on two time scales, namely a single burst, and multiple bursts separated by a long silence period.

Figure 7.2 (a) shows the measured inter-cell gaps for one particular burst of 20 cells in the experimental sample proceeding through 12 switches, illustrating the separation of the cells by the Bernoulli traffic. The first cell is the closest to

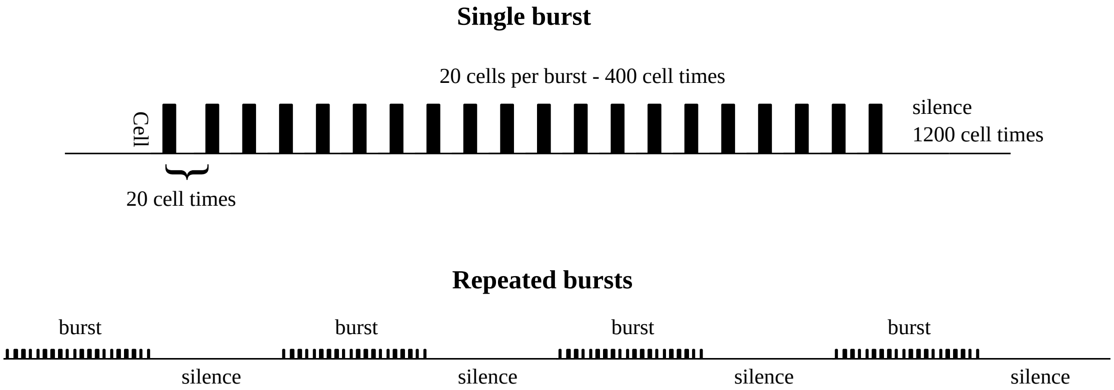


Figure 7.1: Graphical representation of the COST 224 reference source

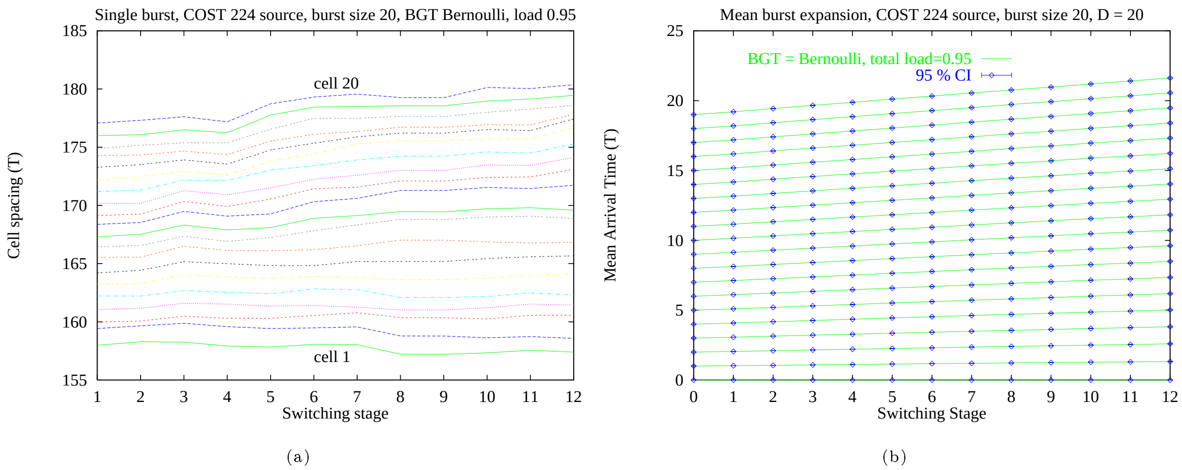


Figure 7.2: Expansion of a single burst (a) and mean burst expansion (b), COST 224 experiment

the x axis. The absence of a global clock means that each switch sees the arrival of the first cell at a different absolute time, but it is only the inter-cell spacing within the burst at each stage which is of interest. Figure 7.2 (b) shows the *mean* burst expansion over the 12 switching stages for each of the 20 cells within a burst. The data were obtained by sampling the inter-arrival time of each cell pair at each switch. The mean inter-cell time for each pair of cells was calculated from 10000 successive bursts. The confidence intervals on the mean position of each cell in the burst are extremely tight.

The measured mean burst expansion was $\mathbb{E}(\Delta B) = 4.36$ slots per switching stage using Bernoulli background traffic. This result matches very well with the analytical approximation and the simulation results. The experiment was

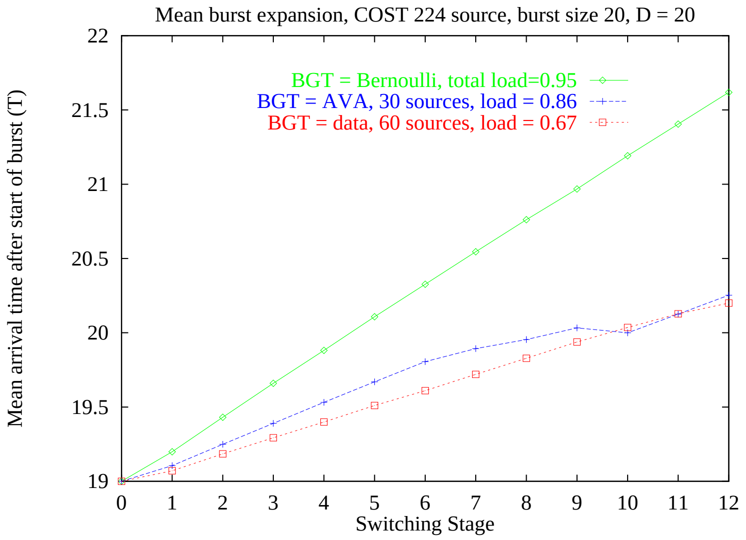


Figure 7.3: Mean burst expansion with Bernoulli, video and data traffic, COST 224 experiment

repeated using the AVA video and data background traffic described in Chapter 6. Figure 7.3 displays the mean position of cell 20 after the start of each burst, with each traffic type. Table 7.1 presents the full results. The experiments with this reference source confirm the observation in the COST 224 report that the burst expansion proceeds approximately linearly with additional switching stages. However, this is does not hold in general, as the following experiments show.

Background traffic	$\rho + 1/D$	$\mathbb{E}(\Delta B)$ (slots)
Poisson	0.95	5.00 (analytical)
Poisson	0.95	2.69 (simulation)
Bernoulli	0.95	4.36 (measured)
AVA video, 30 sources	0.86	2.09 (measured)
Bellcore data, 60 sources	0.68	1.99 (measured)

Table 7.1: Comparison of COST 224 results with experimental results from Fairisle

The lower mean burst expansion for the video and data background traffic can be explained on the grounds of their extreme burstiness and lower loads. Bursts of traffic from the reference connection have a higher probability of being unmodified by the background traffic because the latter tends to transmit bursts followed by silence periods. Reference connection bursts which find the link relatively idle will be modified only slightly.

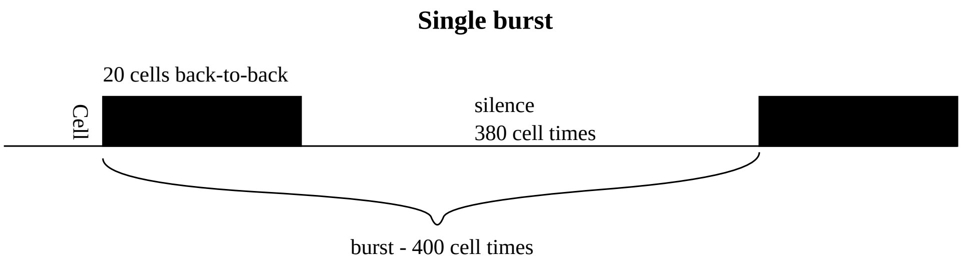


Figure 7.4: Graphical representation of the highly bursty reference source

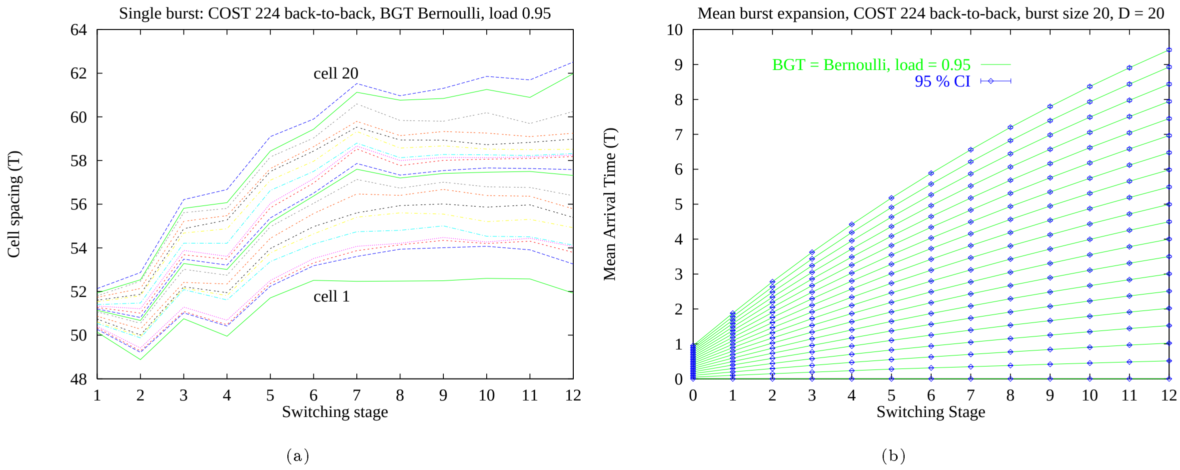


Figure 7.5: Expansion of a single burst (a) and mean burst expansion (b), burst transmitted back-to-back, COST 224 parameters

7.3 Other Bursty Sources

The results of the Section 7.2 validate the accuracy of the COST 224 approximation. The model gives useful insight but is limited in application to modelling bursty arrivals in which the inter-cell times within a burst are equal to the switch size. This section applies the same techniques as used above to examine the effects of multiplexing on highly bursty traffic. The burst expansion experiments were repeated using a modified reference connection which transmits the cells within a burst back-to-back (referred to as the COST 224 source with back-to-back bursts). The reference connection thus transmits 20 cells back-to-back in a period of $20 \times D = 400$ cell times.

In the COST 224 experiment the aim was to achieve a load of 0.95 within the burst, and 0.9 during the silence period. Here the mean load is 0.95 over the full burst and silence period. As a result this source has a slightly higher aggregate

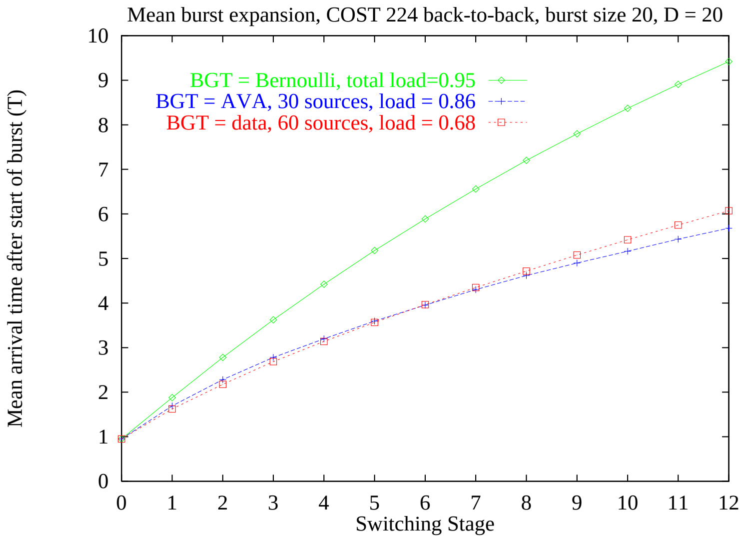


Figure 7.6: Mean position of cell 20 in each burst, with Bernoulli, video and data background traffic, burst transmitted back-to-back, COST 224 parameters, D = 20

mean load. Figure 7.4 depicts the highly bursty reference source graphically, and Figure 7.5 shows the expansion of a single burst and the mean burst expansion in this case, for Bernoulli background traffic. Again, only the relative positions of the cells in the burst are of interest. It is clear that with these parameters the burst expands far more rapidly than previously.

These experiments were also performed with the data and video background traffic; Figure 7.6 displays the mean position of cell 20 in the burst, for each type. Once again the results show that the burstiness of the background traffic causes the bursts of the reference connection to be separated *less* than by the Bernoulli background traffic. A burst of traffic from the reference connection is less likely to be separated by the background traffic because the background traffic tends to emit cells in bursts which are widely separated in time. Table 7.2 presents the measured burst expansion for these experiments.

Background traffic	$\rho + 1/D$	$\mathbb{E}(\Delta B)$ (slots)
Bernoulli	0.95	14.11 (measured)
Bellcore data, 60 sources	0.68	8.53 (measured)
AVA video, 30 sources	0.86	7.88 (measured)

Table 7.2: Mean burst expansion, burst transmitted back-to-back, D = 20

It is also clear that with highly bursty background traffic the expansion does

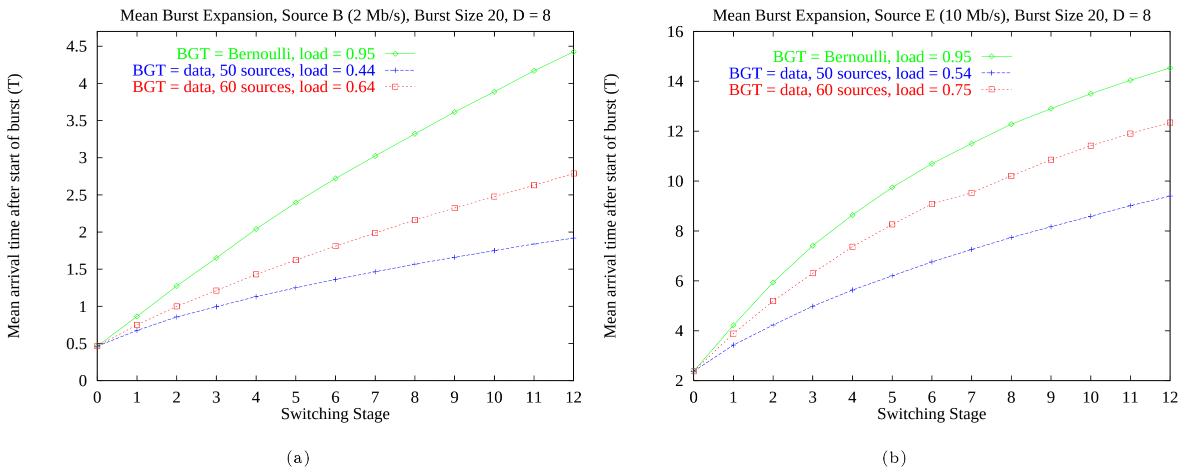


Figure 7.7: Mean position of cell 20 in each burst, for bursty sources B and E with Bernoulli and data background traffic, D = 8

not proceed in a linear fashion. When measuring the burst expansion it therefore only makes sense to measure the mean expansion over the switching stages. The bursts appear to expand more rapidly initially, with an apparent upper bound on the expansion rate. The initial rate of expansion, and the upper bound would appear to be a function, once again, of the proportion of the load contributed by the reference connection, the total load, and the background traffic type.

7.3.1 Periodic bursty sources B and E

To compare these results with the results of the previous chapter, the burst expansion experiments were carried out for the reference sources B and E (2 Mb/s and 10 Mb/s respectively) used in Chapter 6. A periodic, bursty version of each of the sources was created which has the same mean rate as its CBR counterpart, when averaged over the burst period. The periodic bursty CBR version of the source transmits 20 cells back-to-back in a burst, with a burst period equal to 20 times the source period T. Figure 7.7 presents some of the results, showing the position of cell 20 in each burst. The bursts expand more rapidly for the bursty source E than for bursty source B, for the same total load. This is similar to the development of the CDV and the variance of the inter-cell times. The absolute values of the expansion also fit the hypothesis that lower rate sources have a greater absolute variance. After 12 stages of switching bursts from source B have expanded by a factor of 9, compared to 5.6 for bursty source E, with Bernoulli traffic.

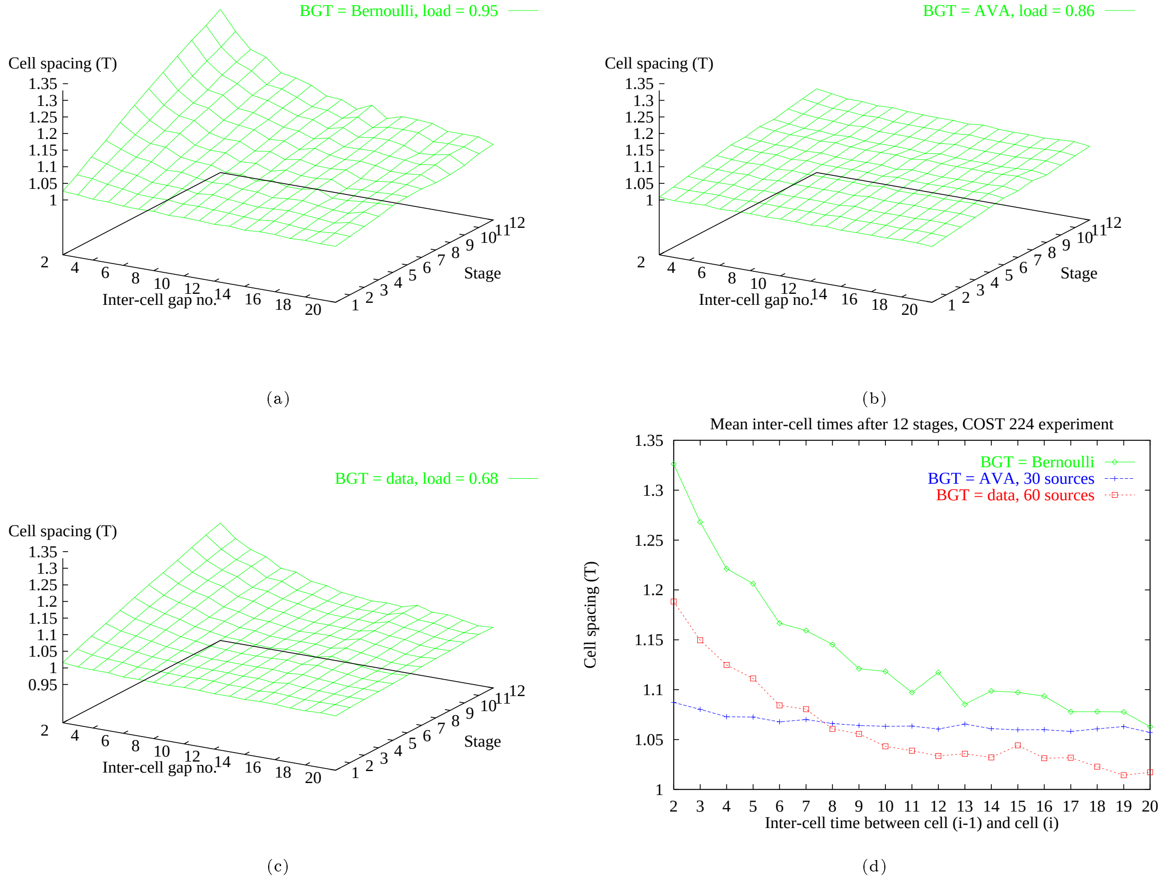


Figure 7.8: Mean inter-cell gap surface for Bernoulli, AVA and data background traffic over all switching stages (a-c), and comparison of mean inter-cell times after 12 switching stages, showing exaggerated separation of leading cells (d) – COST 224 source, $T = 20, D = 20$

7.4 Relative Expansion within a Burst

In their simulation work the COST 224 authors noted that the first cell in a burst tends to be separated more widely from the burst than later cells. Our results confirm this observation; the exaggerated separation of the leading cells is shown clearly in Figures 7.8 (a) to (c) which show for all stages the mean inter-cell gaps within the bursts of 20 cells which were observed in the experiments of Section 7.2. The leading cell becomes more rapidly separated from the burst with additional switching stages. The mean inter-cell times within the bursts after 12 stages of switching, for the 1 in 20 source used in the COST 224 experiments is shown in Figure 7.8 (d).

Earlier cells tend to become increasingly separated from the body of the burst because the cell arriving at the head of a burst sees a system which behaves like an $M/D/1$ queue, just before a Poisson arrival, whereas the remaining cells in the burst see a system which behaves increasingly like an $M+D/D/1$ queue, just before a periodic arrival. The difference is quite marked, and causes increased delay for the later cells. The separation is less exaggerated when using data and AVA background traffic. This is probably because the total load during a burst in these experiments was lower, but in addition, burst expansion occurs more slowly for the bursty background traffic types, and so one might expect the difference in inter-cell times to be less exaggerated. The degree of burstiness of the background traffic would account for the fact that using smoother AVA traffic the leading cell is less likely to be separated than using the burstier data traffic.

When the cells in a burst are transmitted back-to-back, the expansion seems to be evenly distributed over the burst. For the back-to-back version of the COST 224 source, Figure 7.9 shows for each type of background traffic the mean inter-cell gap, similar to Figure 7.8. When comparing the two sets of results note that the inter-cell gap within a burst for the COST 224 source is T at the source, and expands to a maximum of $1.34T$ for the leading cell after 12 stages. For the COST 224 back-to-back source the initial inter-cell gap is 1 slot, or $0.05T$, expanding to a maximum of about $0.52T$ after 12 stages, for Bernoulli traffic. Similar observations hold for the bursty versions of sources B and E, whose results are not presented.

7.5 Bounds on Inter-cell Time Variance

Although the COST 224 model describes burst expansion adequately for the special case where the inter-cell gap within the burst is the same as the switch size, it would be useful to have a more general analytical model for the expected burst behaviour. Fortunately such a model is to hand: the analysis of van den Berg and Resing [van den Berg92], described in Section 6.6, suggests that *irrespective* of the input traffic characteristics of a given reference source, after large number of multiplexing stages at which the reference source is multiplexed with Poisson background traffic, its traffic will have the properties of a renewal process, with an inter-cell arrival time variance of $\mathrm{Var}(X_\infty) = T^2(T-1)/(2T-1)$. Bursts of cells from a periodic bursty source are expected to become increasingly separated by the background traffic until the inter-cell time variance is equal to $Var(X_\infty)$, in much the same way as the experiments of Chapter 6 showed CBR cells becoming increasingly perturbed by background traffic with additional stages. After a large number of stages the cell process should be the same from a statistical point of view, irrespective of its original characteristics at the source, and should depend

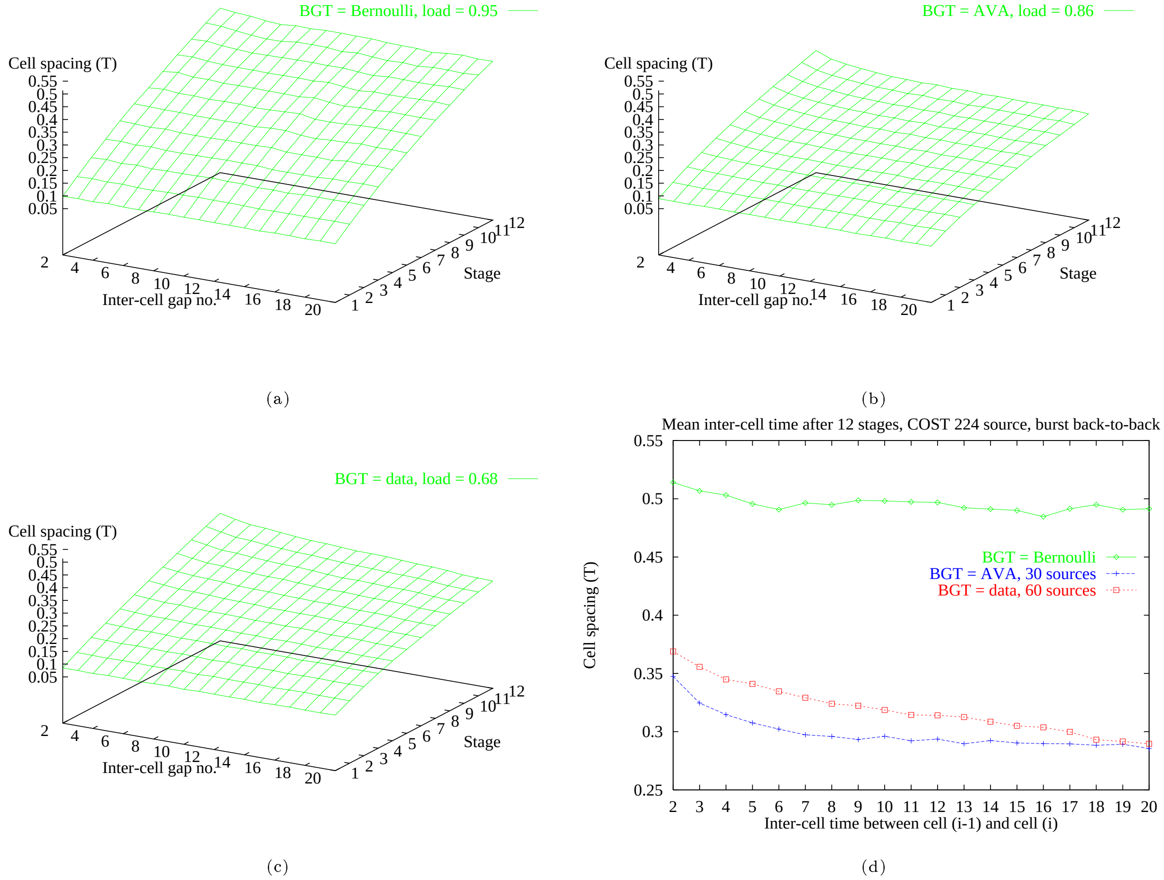


Figure 7.9: Mean inter-cell gap surface for Bernoulli, AVA and data background traffic over all switching stages, COST 224 source with burst transmitted back-to-back (a-c), and mean inter-cell times after 12 switching stages, showing separation of leading cells, COST 224 source with burst transmitted back-to-back, $T = 20$, $D = 20$

only on its mean rate.

Figure 7.10 displays the measured coefficient of variation in inter-cell times for bursty sources B and E. Each graph shows also the predicted value from the approximation of Equation (6.1), which is based on the assumption that the cells originate from the source in a CBR fashion. As the number of switching stages increases, the measured values decrease rapidly towards the analytical bounds. This is most clearly seen for source E, confirming that for a source with a high mean rate $1/T$, the burst expansion is more rapid. With the bursty background traffic types (results are given for data traffic only) the convergence is slower than with Bernoulli traffic. (The convergence will also be to a different bound).

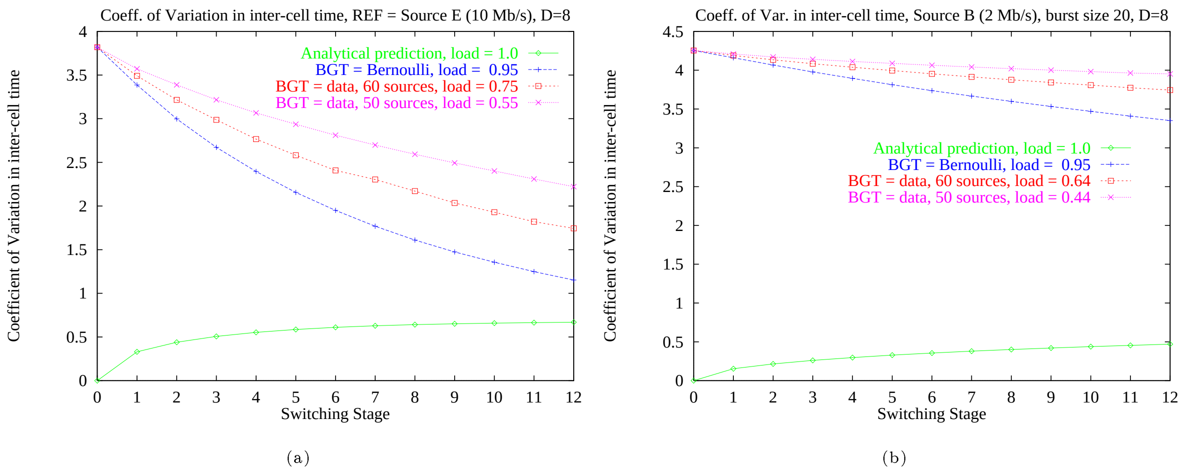


Figure 7.10: Coefficient of variation in inter-cell times for reference sources B and E, comparing analytical result and measured values

These results in effect mirror the results for CBR traffic, where the inter-cell time variance increases towards the analytical bound.

The decrease in variance can also be seen from the inter-cell time distributions for the same sources, as bursts proceed through the network. The left-hand column of Figures 7.11 and 7.12 show the IDT distributions for the same two sources at each switching stage. Each graph has two features: a peak near the origin caused by the large number of cells which are close together within a burst (these cells are back-to-back at the source) and a peak towards the extreme of the x axis which represents the inter-cell time between the last cell in a burst and the first cell in the succeeding burst. Since it is relatively rare to have an inter-cell time greater than the inter-burst time, the expansion of the bursts takes place by decreasing the inter-burst gap, with cells moving both from the tail of the previous burst and from the leading edge of the successor burst, to occupy the inter-burst gap. With Bernoulli background traffic the expansion is smooth, with the bulk of the distribution tending to move back towards the origin, and a consequent decrease in variance as displayed in Figure 7.10. With the other types of background traffic, however, a significant number of cells remain widely separated in time, and hence the decrease in the inter-cell time variance depicted in Figure 7.10 is lower than for Bernoulli traffic.

7.6 CDV Measurements for Bursty Sources

The CDV of a bursty reference connection is important because it provides insights into the expected effects of multiplexing on traffic which has been perturbed

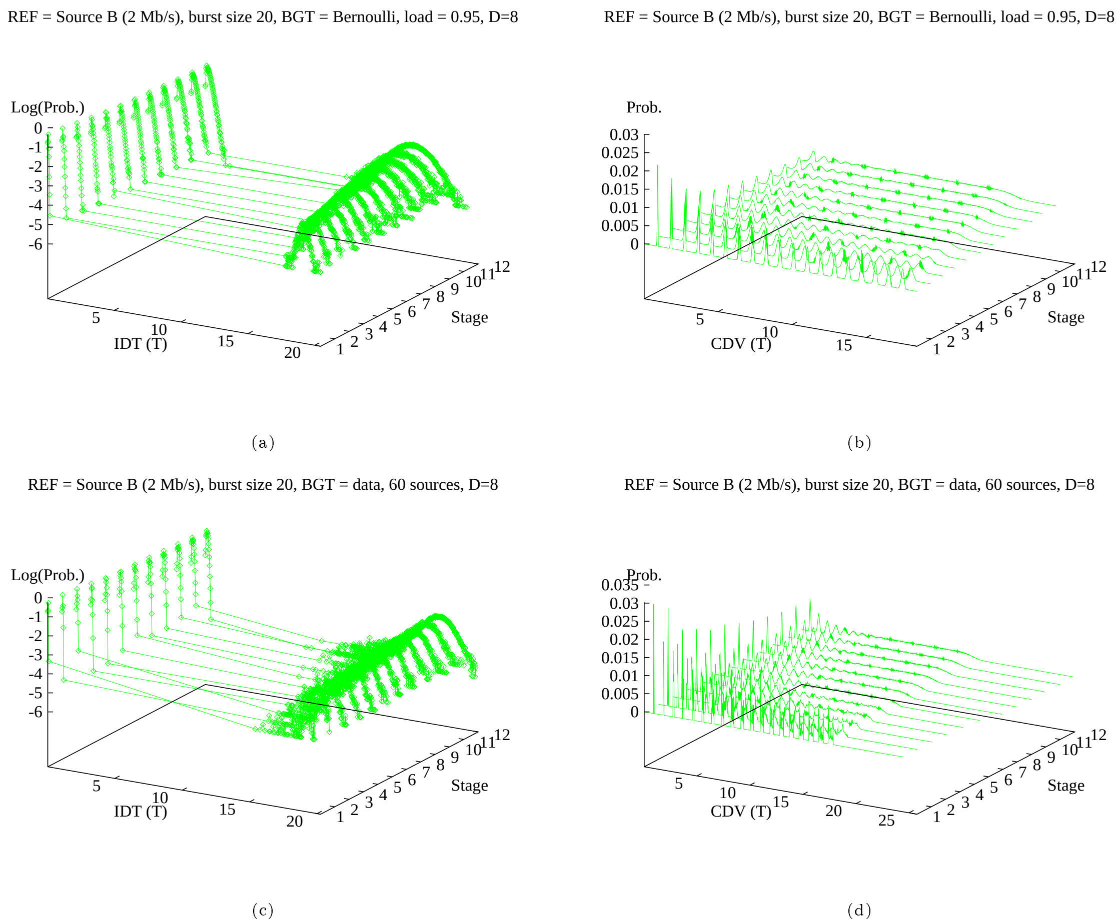


Figure 7.11: Inter Departure Time distributions and CDV for bursty source B

by the network. This section studies the CDV using the I.371 leaky bucket algorithm described in Section 6.3. If the I.371 leaky bucket is parameterised as though the source were CBR with negotiated peak rate $1/T$, then the source will appear to have a high CDV when it enters the network. Consider a periodic bursty CBR source whose bursts are of size b, then on entry to the network the CDV distribution will have $b-1$ peaks, at positions $i(T-1)$ (ignoring the speedup), for each of the i cells ($2 \leq i \leq b$) following the first cell in each burst (which will always arrive on time). The CDV distribution will be zero elsewhere.

What happens to the CDV as the burst progresses through the network? As the bursts expand and the traffic becomes smoother, the variance in inter-cell times decreases toward the lower bound. The traffic distribution becomes more even as the bursts proceed through the network, suggesting that the CDV for a bursty reference source should *decrease*. Measurements confirm that the mean CDV decreases asymptotically towards a lower bound which is dependent on the

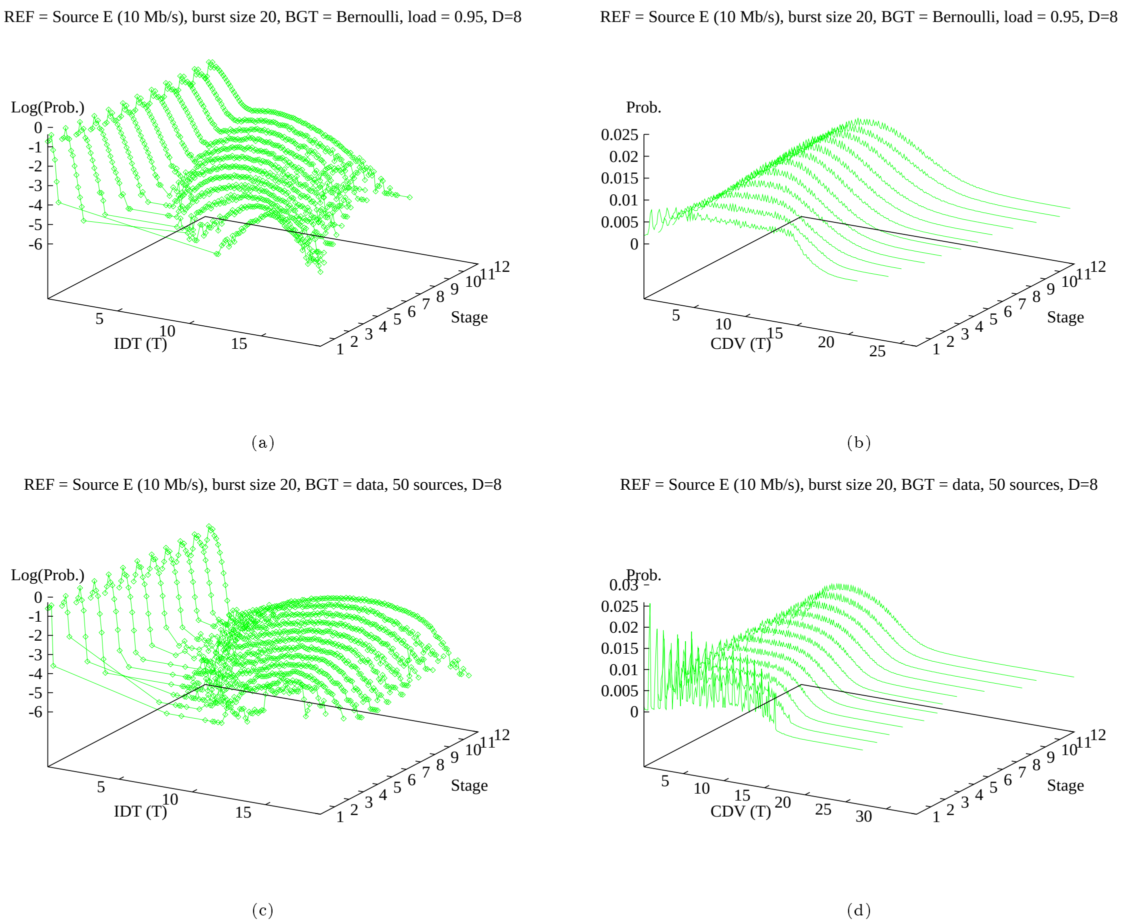


Figure 7.12: Inter Departure Time distributions and CDV for bursty source E

background traffic type, the value T, and the total load. Unfortunately the same is not true of the tail of the CDV distribution, which decreases only for Bernoulli background traffic. For all other background traffic types it actually increases, suggesting that bursts can "catch up" with one another.

The evolution of the CDV is pictured in the right-hand column of Figures 7.11 and 7.12 for sources B and E. The CDV distributions show the burst expansion clearly – the initial peaks become less visible as the reference connection cells become displaced by the background traffic. Depending on the total load and the type of the background traffic this mixing process is more or less effective at removing the burst peaks.

In spite of the fact that the tail of the CDV distribution is not bounded, the total proportion of cells which experience positive CDV (i.e.those which would be candidates for discarding by the I.371 leaky bucket) appears to be bounded above. This is shown in Figure 7.13. On entry to the network the proportion of

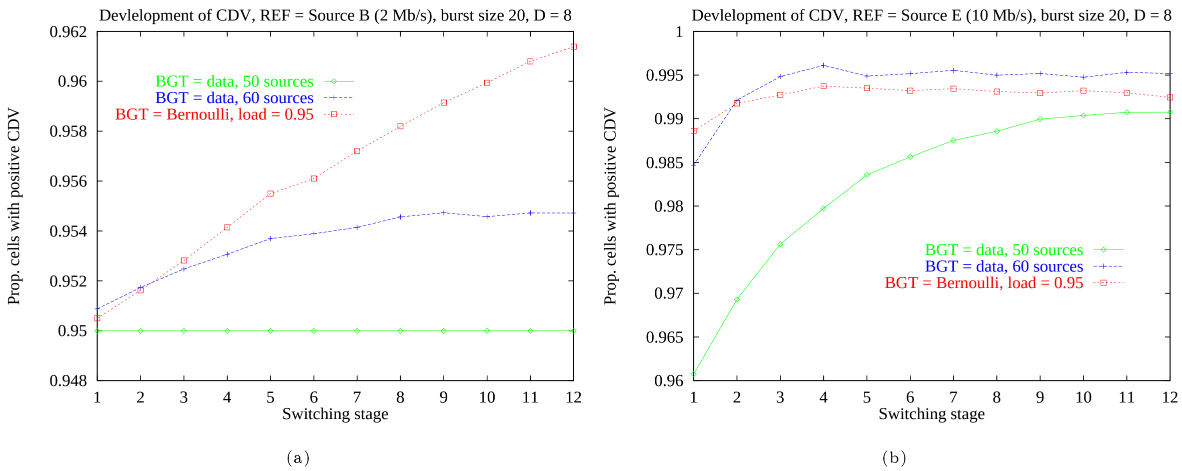


Figure 7.13: Proportion of cells with positive CDV as bursts proceed through the network

cells with positive CDV is 0.95, since only 1 in 20 cells is not early. For source E the bound is close to 1, whilst for source B it is not clear what the value of the bound will be using Bernoulli traffic.

The decrease in mean CDV with additional stages is caused by the bursts of reference traffic becoming separated by the background traffic. After 12 switching stages Figure 7.14 shows for the bursty sources B and E, the distribution of the burst sizes in the reference stream. Here the burst boundary is defined to be a gap of 5 or more cell times, this being sufficiently small to ensure that all sub-bursts of the original burst size of 20 cells are measured. With source B, it is clear that the Bernoulli traffic has caused the greatest decrease in burst size, whilst the data traffic still exhibits sharp peaks at the original burst size of 20. The effects of load are clearly visible – the bursts have expanded more with the higher load of data background traffic. In the case of source E, the graphs for data background traffic show peaks at multiples of the original burst size of 20 cells. Bursts, catching up with one another, cause larger bursts which are responsible for an increase in CDV. Even with Bernoulli traffic there are still large bursts of traffic after 12 stages. As a final note, the IDC of the reference streams is studied further in Appendix C which demonstrates that the bursty reference traffic becomes increasingly like a renewal process with additional stages of switching.

7.7 Summary

The findings of this chapter are summarised below:

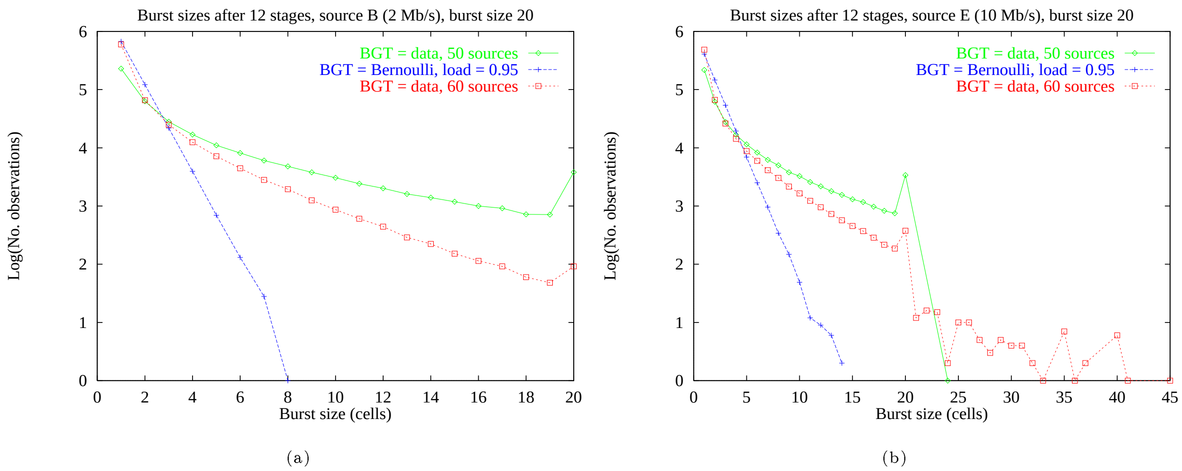


Figure 7.14: Comparison of burst sizes in the reference stream for bursty sources B and E after 12 stages, with Bernoulli, and data background traffic, using a burst gap of 5 cell times

- The COST 224 model of burst expansion seems to be accurate, although limited in its application.
- The experimental study of of burst expansion for highly bursty sources has shown that the burst expansion rate is non-linear. The rate is dependent on the mean rate of the bursty source, the total load and the type of background traffic, as well as the initial burstiness of the source.
- Burst expansion occurs more rapidly for a reference connection contributing a high proportion of the total load, and stabilises more quickly. This is similar to the more rapid increase in CDV for the higher bitrate CBR source in Chapter 6.
- Irrespective of the parameters of the bursty reference source, the burst expansion is greater for less bursty background traffic, given the same load – the expansion rate using Bernoulli background traffic is greater than when using the more bursty AVA traffic of virtually the same mean load.
- Burst separation is not uniform across cells of the burst. The leading cells are more widely separated than cells later in the burst. For bursts which are transmitted back-to-back the relative inter-cell gap between cells in the burst remains constant until the cells are somewhat separated, and then the leading cells become more widely separated.
- The mean CDV decreases as bursts become separated by background traffic. The maximum observed CDV decreases only in the case of the periodic

bursty source B. Otherwise it tends to remain stable or even increase.

- The variance in inter-cell time decreases as bursts proceed the network, and is bounded below by the bound of van den Berg and Resing in the case of Bernoulli traffic.

- The total proportion of cells which incur positive CDV is bounded above, again as a function of the total load, the reference source rate and the background traffic type.

- The analytical bounds for burst expansion are conservative, and can thus be used for network dimensioning.

7.8 Concluding Remarks

The experimental work of this chapter complements the study of CDV for CBR reference sources in Chapter 6. Its findings are reassuring – multiplexing tends to improve the traffic characteristics of bursty connections. If sources are shaped and/or policed at the entrance of the network, then their traffic characteristics should not need to be monitored (at least for CAC purposes) elsewhere in the network. Chapter 8 investigates the degree to which scheduling policies can ameliorate the CDV experienced by CBR and bursty sources.

Chapter 8

Performance Guarantees for Delay and Loss-sensitive Traffic

8.1 Introduction

Previous chapters have studied the CDV caused by queueing at both input and output buffered switches, when the cells queued for transmission are served in First Come First Served (FCFS) order. The study of FCFS queueing systems is useful for several reasons: first, performance results from FCFS systems provide worst-case bounds on the performance of many other queueing disciplines; second, most currently available ATM equipment only implements FCFS queueing, and finally, FCFS queueing systems are easier to model analytically. One of the main motivations for the use of ATM in a multi-service environment is that it enables different types of traffic to be managed differently to enable their QoS constraints to be met; non-FCFS queueing disciplines which attempt to meet the QoS requirements (in terms of delay, loss, and bounds on jitter) of different types of traffic have consequently received much attention.

This chapter reviews several non-FCFS cell level scheduling policies and comments on their performance, as a result of experiments on the Fairisle network. It also presents results from an investigation of buffer allocation policies which cater to different traffic classes.

8.2 Performance of Input Buffered Switches

Input buffered switches using FCFS queueing are throughput limited to 0.58 of the offered load because contention in the fabric can cause head of line blocking, where a blocked cell at the head of the queue prevents other cells from contending for output ports [Hluchyj88]. Provision of delay bounds for delay sensitive traffic at input buffered switches is thus more complicated than for output buffered switches: the scheduling policy must take into account not only the queued cells of each traffic class at a given port, but also the traffic loads at all other input ports and the routes cells will follow across the switch fabric.

8.2.1 Scheduling the Fabric

Several methods have been proposed to overcome head of line blocking. Oie *et al.* investigated the effect of speeding up the switch fabric by a factor L to increase the effective number of arbitration cycles per cell time [Oie89]. They found that a modest speedup of $L = 3$ improved the throughput limit to 0.975. Hluchyj *et al.* proposed a switch in which the first b cells in each input port contend in parallel; the total number of ports is increased from D to Db and the fabric operates at a (slower) rate of $L = 1/b$ [Hluchyj88]. Any more than b cells routed to one output per cycle are lost. They also suggested a switch which considers up to w cells per input, each slot. First the cells at the head of all queues contend for outputs, then the cells in the second positions contend for remaining outputs, and so on, up to w. Depending on the choice of w, throughputs close to 1 can be attained, but the overhead of searching each queue is high.

To alleviate head of line blocking the Fairisle fabric runs at a rate of $L = 1.2375$ (currently) times the line rate, theoretically yielding a throughput of 0.78 for a 16×16 crossbar fabric, and 0.63 for the delta configuration. This, in turn, requires a small degree of output buffering, which is provided by small output FIFO with space for approximately 5 cells [Black94b]. In spite of the speedup, it is impossible to bound the cell queueing delay, because the arbitration at each point of contention (the output of each switching stage and the fabric) is independent. It is even possible for a cell to always fail arbitration in the presence of pathological traffic patterns.

To bound the queueing delay it is necessary to schedule the fabric. This can be achieved either by centralising the arbitration mechanism or by using a distributed scheduling algorithm. Whilst the former is attractive, it has a cost which grows with the square of the switch size. Two distributed algorithms for scheduling the fabric (which require hardware support for efficient implementation) are proposed in [Anderson92]. The first, called "parallel iterative matching", iteratively matches inputs with outputs in a pseudo-random fashion. It can be shown to find a maximal match of inputs to outputs in logarithmic time, and to guarantee fairness. Because of its random nature, however, the algorithm cannot bound the cell queueing time. A second algorithm proposed by the same authors divides time into fixed length frames (of slots). A connection requiring a bounded delay is allocated a certain number of slots in each frame according to its transmission rate. No other input may transmit to the same output during the reserved slots.

A distributed algorithm is used to construct the schedule which the frame represents. Slots which are not allocated are used for the transmission of best effort traffic. Using this algorithm the latency of delay sensitive traffic can be strictly bounded.

Both of these algorithms apply only to non-blocking switching fabrics – contention only occurs for the switch outputs. In the case of Fairisle the fabric itself is blocking, with contention at the output of each switching stage as well as for the output FIFO. This, coupled with its lack of hardware support for distributed scheduling, makes the provision of guaranteed latency extremely difficult. Early experiments with a modified frame-based scheduling algorithm are currently in progress[1]. Scheduling algorithms for input buffered switches are not considered further.

8.3 Output Buffered Switches

At switch outputs cell level scheduling is required to avoid the performance penalties of FCFS queueing. Two classes of algorithms can be identified, namely *work conserving* and *non-work conserving*. The latter types permit the server to be idle with work in the queue, with the aim of optimizing some desirable performance criterion. Until recently only work conserving policies were studied in any detail, owing to their analytical tractability. Analytical solutions for non-work conserving policies are generally unattainable.

Two resources are typically in demand at cell level schedulers within ATM systems, namely bandwidth and buffer space. When instantaneous demand exceeds capacity, queueing will result. Policies must not only order cells for transmission, but also allocate buffer space to different classes of traffic to ensure that their loss constraints are met. It may be important to implement per-VCI queueing to ensure that connections are protected from one another, and to impose an upper bound on the queue length of any single connection.

8.3.1 Work Conserving Policies

All work-conserving scheduling policies use priorities of some form to distinguish between customers of different classes. A static priority scheme (possibly with several priority levels) can be used, for example, to transmit delay sensitive before delay insensitive traffic. An improvement of several orders of magnitude in the loss rate of delay-sensitive traffic (due to lateness) can be achieved with only a small performance penalty for the loss insensitive traffic if the proportion of priority traffic is low [Bae91].

[1] This is being done by Naeem Khan at the Computer Laboratory.

8.3.1.1 Deadline Schedulers

An alternative is to use priorities which vary in response to the traffic load. In classic Earliest Due Date scheduling (EDD) [Jackson60] customers are scheduled by their deadlines to the start of service, where the deadline is a function of the customer arrival time. EDD can be shown to minimise the proportion of customers which miss their deadlines [Saito90]. A modification of EDD, the Shortest Time to Extinction (STE) policy, discards customers which have already exceeded their due dates, reducing the delay for the remaining customers [Panwar88]. A further extension, called delay-EDD, permits the server to provide a hard delay and mean rate bound provided the source obeys a peak and mean rate [Ferrari89]. The deadline in this case is the theoretical deadline of each packet assuming that the source obeys the contract. Yet another variation on the deadline scheduler is the Oldest Customer First (OCF) policy: each packet carries a time stamp indicating its accumulated delay in the network, which is used by the scheduler to order transmission [Chen89]. Although optimal with regard to minimizing the maximum delay and maximising the minimum delay, all deadline-based algorithms require the customers to be sorted according to the deadline schedule. Computation of the schedule has high cost without appropriate hardware support. A candidate hardware queue sorting mechanism proposed by Moore [Moore94] could be implemented in ATM port controllers for this purpose. This device has a cost proportional to twice the number of elements to be sorted.

8.3.1.2 Rate Guarantees

Several other mechanisms have been proposed which guarantee each connection a minimum rate of service whilst isolating its traffic. The Fair Queueing algorithm performs round robin scheduling on each of N queues, one for each connection [Demers89]. The visit ratio to each queue may be proportional to its requested bandwidth. Per connection queueing permits the flows to be isolated from one another, but a policy must still be implemented to allocate buffer space. Fair Queueing can be shown to be equivalent to the Virtual Clock algorithm, in which arriving packets are stamped with the difference between their expected arrival times, given the negotiated source rates and the time at which they arrived [Zhang90]. This is used to schedule the cells for transmission. The two algorithms differ from EDD because they provide only a rate guarantee and not a delay bound.

8.3.1.3 Priority Mechanisms

In [Lim88] Lim and Kobza described the HOL–PJ scheduler, which stands for Head-Of-Line with Priority Jumps. An arriving cell is placed in a FCFS queue j

for its class j, and stamped with its arrival time. A cell may spend maximally T_j time units in queue j before it jumps to the next highest priority queue $j-1$. Cells in queue 1 wait for service. HOL–PJ requires less computational overhead than EDD, but can satisfy any relative average delay requirements for different classes and approximately minimize the overall maximum tail probability in the system.

Chipalkatti *et al.* [Chipalkatti89] proposed the Minimum Laxity Threshold (MLT) algorithm which transmits the delay sensitive cell with the least time remaining until its deadline, provided that its laxity is below some threshold. If no such cell exists then the delay insensitive traffic is given priority. A second scheme is the Queue Length Threshold (QLT) in which delay sensitive traffic is given priority until the queue length of the delay insensitive traffic exceeds some threshold. For a given set of parameters QLT and MLT can compromise between the loss of delay sensitive traffic and the delay of delay insensitive traffic. Whilst MLT performs better, QLT is favoured for its ease of implementation. Aawater and Schoute [Aawater91] have shown that the optimal set of priority based policies which balance low loss with low delay are the set of QLT policies with different values of the threshold parameter.

Two other algorithms of interest are FIFO+ [Clark92] and Hop Laxity Threshold (HLT) [Schulzrinne94]. Under FIFO+, each network node estimates the average delay for each class of traffic. Each packet is given a priority, on departing, which indicates its experienced delay relative to the average. At the following node packets are given priority if they experienced worse than average delay at the previous node. Under HLT, the laxity is a measure of the time remaining until a packet's deadline expires. Packets with the least remaining time till their deadlines expire are always transmitted first, however the laxity is scaled according to the number of hops remaining on the packet's path. This scheme performs better than FIFO+, FCFS and several other mechanisms, particularly when the path length increases. Neither is suited to ATM networks because of the need to stamp the cell header.

8.3.2 Non-work Conserving Policies

Non-work conserving scheduling policies can provide guarantees impossible to achieve with work conserving mechanisms. The"stop-and-go" queueing mechanism is a rate-based control which guarantees no congestion whilst preserving smoothness properties of the input traffic [Golestani90]. Each node divides time into frames of fixed size; traffic which arrives during frame i is transmitted during frame $i+1$. Sources are policed at entry, and the mechanism guarantees a rate and the minimum and maximum end-to-end delay. Choice of the frame size is difficult, because it must accommodate multiple rates. The jitter-EDD policy is similar to OCF, but maintains an additional variable per packet which represents the difference between its deadline and its actual service time [Verma91]. At the next node, the packet is held for that amount of time before entering the scheduler. Thus the mechanism can bound the variance in end-to-end delay.

The Hierarchical Round Robin (HRR) mechanism has a similar time-framed structure [Kalmanek90]. To enable the provision of multiple transmission rates to different traffic types, HRR has a hierarchy of frames: a frame at level i is allocated a proportion of the bandwidth at level $i-1$. The server processes frame 0 repeatedly. Each time it encounters an entry for a lower rate frame, it allocates the appropriate number of slots to that frame. Each connection is given the appropriate bandwidth allocation at a frame level of the correct rate. An efficient hardware implementation of the algorithm is possible. Whilst HRR guarantees delay bounds, it cannot guarantee end-to-end delay variance, because it does not reshape the traffic to its original form at every node. In addition, HRR is unable to cater for low latency, low bandwidth traffic. An excellent comparison of work conserving and non-work conserving policies is given in [Zhang91]. Finally, a mechanism named Rate Controlled Static Priority scheduling (RCSP) which separates the regulation of traffic and the ordering for transmission has been proposed by Zhang and Ferrari [Zhang93a]. RCSP has the advantage that it avoids the frame-based granularity of bandwidth allocation of HRR and stop-and-go, whilst still enabling the implementation of work conserving and non-work conserving policies. Specifically, it provides delay, throughput and loss guarantees, decouples delay and bandwidth allocation, is easy to implement and requires relatively simple admission control tests.

8.4 Experiments with Delay Sensitive Traffic

To demonstrate the difference between the two classes of non-FCFS queueing policies which aim to improve on the FCFS delay performance for delay-sensitive traffic, two policies were implemented: Static Priority queueing (SP) and a Rate Based scheduler (RB), equivalent to HRR. SP is work conserving, and RB non-work conserving.

8.4.1 Implementation Issues on the Fairisle Switch

The location of the cell level scheduling policy for the non-blocking switch configuration using Fairisle is shown in Figure 8.1. The diagram shows two Fairisle switches configured as a single output buffered switch. The implementation of the scheduling policy is performed by the input port of the second Fairisle switch, before the cells are transmitted across the second fabric and onto the output link.

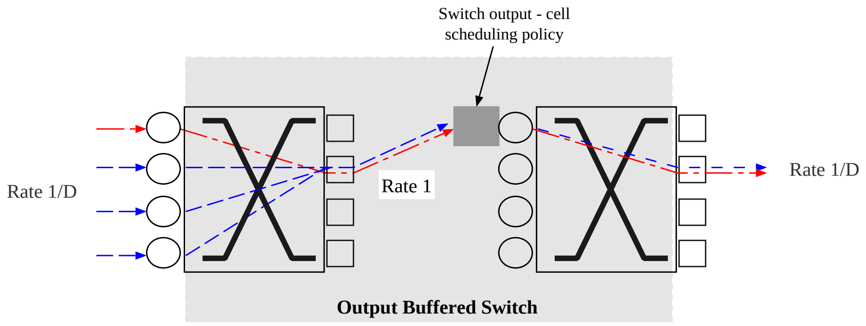


Figure 8.1: Cell level scheduler implementation in the non-blocking configuration

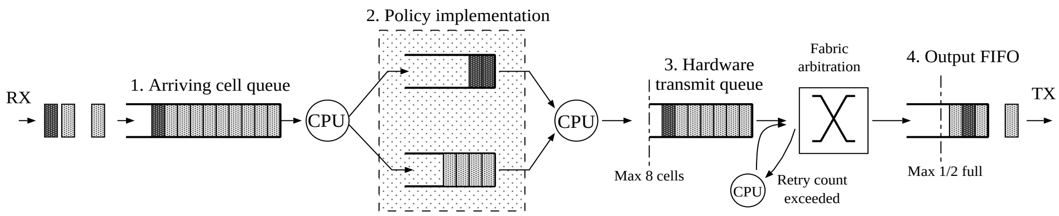


Figure 8.2: Cell level queueing in the Fairisle port controller

Recall from Chapter 6 that the reduced rate of the output link causes backpressure from the output of the second switch so that queueing occurs at the input. The queueing mechanism itself is complicated and deserves attention because it directly influences the performance of the system.

Figure 8.2 shows the detailed structure of the Fairisle port controller, from input to output, from the point of view of a cell awaiting transmission. An arriving cell (in this case from the non-blocking switching fabric formed by the transmission link) is framed and transferred into a cell buffer by the reception hardware, and the CPU is informed by the posting of an interrupt. Cells awaiting CPU processing are queued in a FCFS fashion in an arrival queue (1). The CPU removes cells from the queue and, on the basis of the VCI, locates the context describing the (queueing) policy to implement for each cell. Typically this involves the processor executing some receive method within the context, and placing the cell on a queue (2). The policy itself is not important within the scope of this discussion. The processor also executes the transmission schedule which implements the policy. Transmission scheduling may occur in response to actions scheduled for specific time instants or simply when bandwidth becomes available for transmission. Cells scheduled for transmission are enqueued on the hardware-managed transmission queue (3). This queue has a maximum size of 8 cells and the processor is informed whenever space is available. Cells on the transmission queue are transmitted across the fabric by the hardware, subject to the arbitration mechanism of the fabric. Cells which fail in contention for the fabric are automatically retried for a (per cell) programmable number of fabric arbitration times (with a limit of 8), after which CPU intervention is required. Finally, a cell which successfully traverses the fabric is inserted into an output FIFO (4) which accommodates the rate discrepancy between the fabric and output link.

The output FIFO generates backpressure across the fabric if it is more than half full (in bytes) when the first byte of a cell is transmitted across the fabric (256 bytes in the current implementation). The FIFO is drained and framed into cells by the transmission hardware.

A number of issues affect the performance of this system. First, at high load the output FIFO will frequently be full; backpressure will be applied across the fabric and cells will need to be retried several times before crossing the fabric. Second, the hardware transmit queue (3) will also frequently be full, with the result that a cell leaving the policy queues (2) may find itself queued in a FCFS queue which "stretches" across the fabric to the transmission link. Depending on the load the processor will have to step in to retry cells which exceed their hardware retry limit, and further, the CPU must divide its time between servicing the arriving customers and keeping the transmit queue non-empty.

8.4.2 Static Priority Scheduling

Implementation of a simple priority-based policy in the cell level scheduler on the Fairisle port controller, described in Chapter 5, is straightforward: the policy implements two queues, one for each priority of traffic. On arrival, a high priority cell schedules itself for immediate transmission. On departure, a high priority cell, if it is the only such cell in its queue, schedules the low priority queue for transmission. The implementation is attractive because it has low cost.

8.4.3 Rate Based Scheduling

The RB policy requires scheduling decisions to be made periodically according to the negotiated transmission rate for each connection. In these experiments a single reference connection was used and the RB scheduler was parameterised to deliver its mean rate. The implementation is equivalent to the HRR server with best effort traffic.

Implementation of the simple rate based scheduler used requires 2 queues: the first is dedicated to the background traffic, and is allocated transmission bandwidth on a "best effort" basis; the second is parameterised with the assigned mean rate of the reference connection. An arriving rate-guaranteed cell is simply enqueued. Using the notation of Chapter 6, for a reference connection with mean rate $1/T$, the rate-guaranteed queue is serviced by a periodic interrupt with period T. Experience with the implementation suggested that a value of $T - 0.1$ slots was more appropriate, to accommodate the scheduling latency which can arise for the following two reasons: if the system is idle each RB scheduling epoch causes a CPU interrupt, with its associated latency; alternatively if the CPU is busy (i.e. in FIQ mode) when the epoch occurs, then because reception and transmission are non-preemptable, there will be some delay before the rate-guaranteed traffic is served.

An arriving best effort cell is enqueued in the best effort queue, which is served whenever the rate-guaranteed queue is not scheduled for transmission. The transmission schedule for the RB scheduler uses the timer in the CLS described in Chapter 5. The timer schedule contains a single method which schedules the rate-guaranteed queue for immediate transmission and resets the timer.

8.4.4 Experimental Method

Each experiment aimed to determine the effects of one of the policies on a single delay-sensitive connection competing with the different background traffic types described in Section 6.4.3. As before, the connection was traced over 12 switching stages, with measurements taken after each stage. The CDV of the delay-sensitive reference connection was measured using the sampling leaky bucket described in Chapter 6, parameterised with a leak rate of $0.95T$, where $1/T$ is the mean rate of the reference source.

Two types of reference connections were used, namely CBR and periodic, bursty sources. The parameters for the sources and the names used to refer to them are as described in Table 6.1.

8.4.5 Results and Discussion

This section presents results of the experimental study, comparing the effects of the different policies, loads, background traffic types and reference connection characteristics.

8.4.5.1 Comparison of Policies

The performance of the two non-FCFS policies is best assessed in comparison with FCFS queueing, which was investigated in Chapter 6. Theoretically under SP each high priority cell should be transmitted immediately and the connection properties preserved throughout. The RB policy, serving a reference connection with mean rate $1/T$, should result in a stream which is strictly periodic. Slight rate mismatches between the source and the RB scheduler will result in some

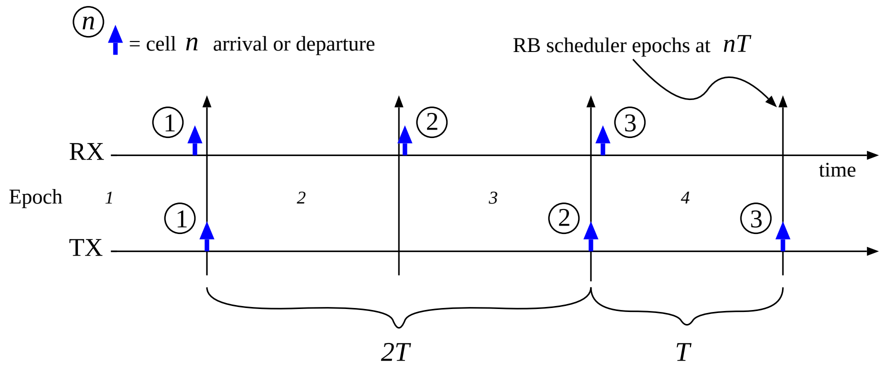
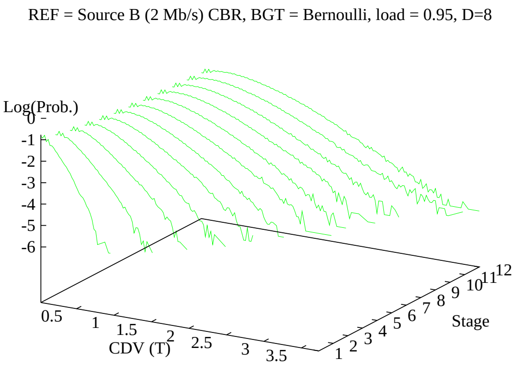


Figure 8.3: Theoretical bound on inter-cell time for the RB policy

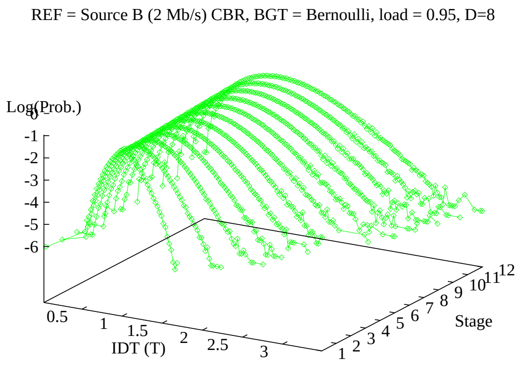
variation, with the result that the inter-cell time for reference connection cells should be bounded by $2T$ [Zhang91], as shown in Figure 8.3. The diagram shows the arrival and transmission times for each of 3 cells at the RB scheduler. Cell 1 arrives just before a RB scheduling epoch and is transmitted on the epoch. Cell 2 arrives slightly late (due to a slight rate mismatch) and can only be transmitted at the end of epoch 3. Cell 3 arrives during epoch 4 and is transmitted at the end of the epoch. There should be no measurable CDV because the 5% speedup in the leaky bucket rate should accommodate experimental variation.

Figure 8.4 compares the results for the 3 disciplines, showing the measured CDV and inter-cell departure time distributions after each switching stage for CBR source B (2 Mb/s), with Bernoulli background traffic and a load of 0.95. The results are startling. Superficially, RB performs best, followed by SP and FCFS. The CDV under SP is far greater than expected, with a maximum of just over $2.5T$ after 12 stages, compared to $3.6T$ under FCFS. The CDV under RB is held constant, as expected, but has a higher maximum value than expected, of $0.29T$. For SP the IDT distribution becomes increasingly dispersed and the initial peak (which should have been preserved throughout) is rapidly spread out. Similarly with RB the spread around the expected peaks at T and $2T$ is fairly substantial, with a span of $0.64T$.

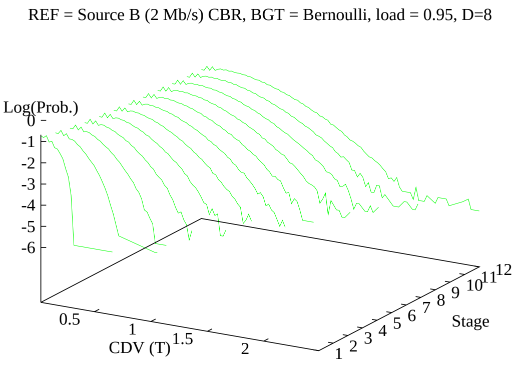
Figure 8.5 plots the same results for CBR reference source E. In this case the measured tail CDV value under SP is $3.625T$, and under RB $1.75T$. Comparison between the two sources B and E shows that source E, because of its higher rate, suffers greater relative CDV, however the absolute CDV values, in terms of cell slot times, are virtually the same for both traffic types using the non-FCFS policies.



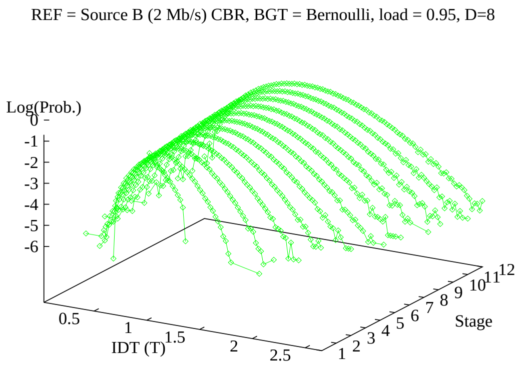
(a) FCFS



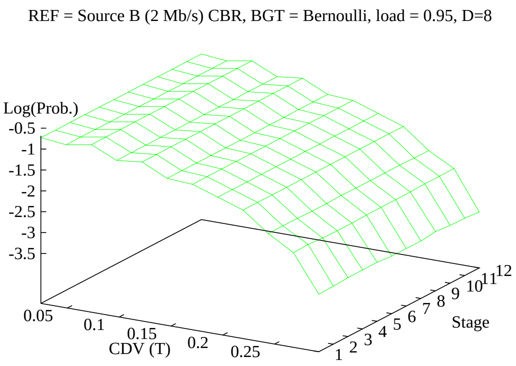
(b) FCFS



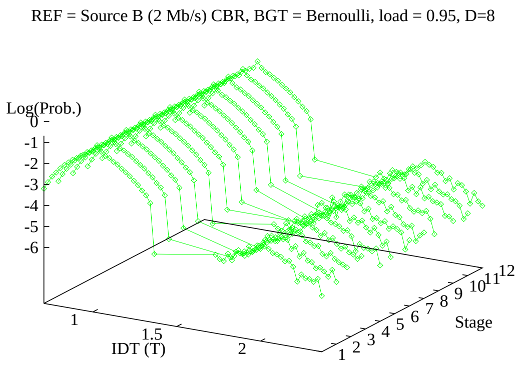
(c) SP



(d) SP

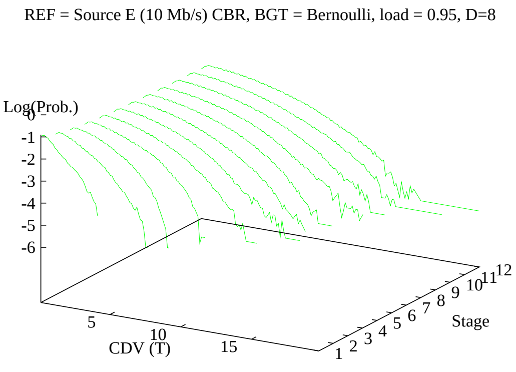


(e) RB

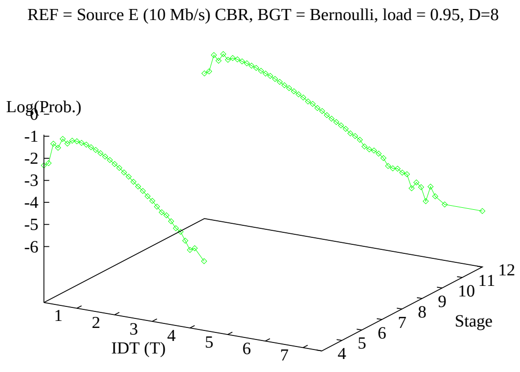


(f) RB

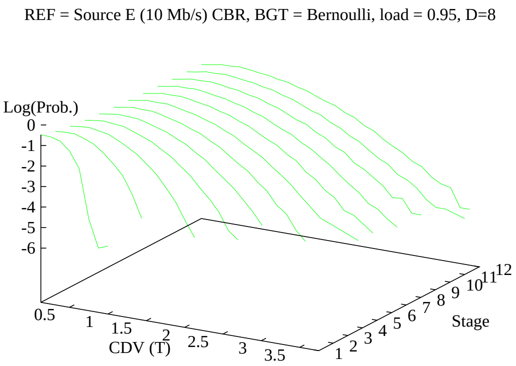
Figure 8.4: Comparison of CDV and inter-cell departure time distributions for source B using FCFS (a,b), SP (c,d) and RB (e,f) policies, with Bernoulli background traffic. Note the difference in CDV scales, required to make all graphs visible



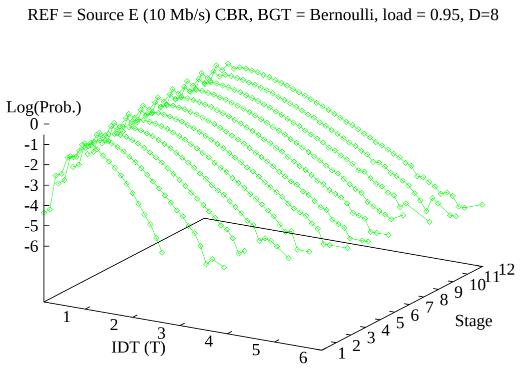
(a) FCFS



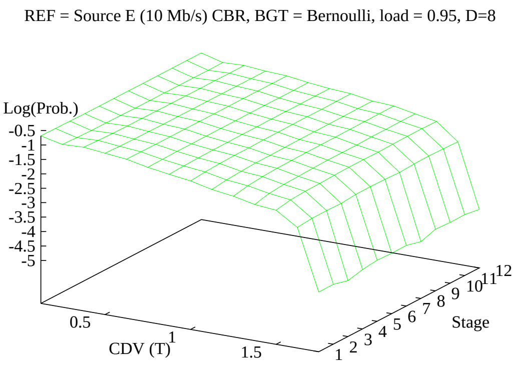
(b) FCFS



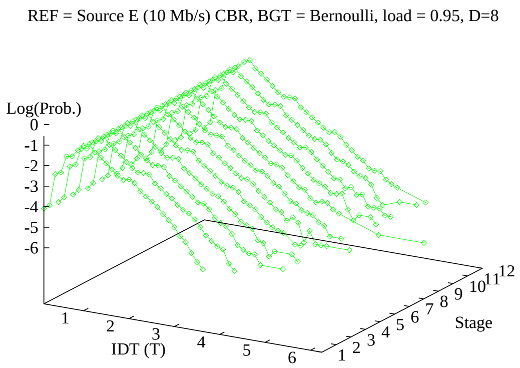
(c) SP



(d) SP



(e) RB



(f) RB

Figure 8.5: Comparison of CDV and inter-cell departure time distributions for source E using FCFS (a,b), SP (c,d) and RB (e,f) policies, with Bernoulli background traffic. Note the difference in CDV scales, required to make all graphs visible

Effect of the Hardware Transmit Queue

The independence of the CDV of the source rate, together with the difference between the expected and measured performance suggests that cause of the CDV is the fixed size FCFS output queueing mechanism, described in Section 8.4.1 above.

Table 8.1 compares the maximum expected transmit queueing delay (TXQ delay) given the queueing structure of the Fairisle port controller shown in Figure 8.2 with the maximum observed inter-cell departure times after 1 stage of switching for SP and RB, for each source. The worst case delay experienced by a reference cell scheduled for transmission, under either policy, will occur if (a) the output FIFO has just accepted a cell which takes it over its half full mark, and (b) the hardware transmit queue has just accepted a cell and become full. The output FIFO thus contains 5.7 cells, and the hardware transmit queue 8 cells. These will take 13.7 cell times to transmit, equivalent to a delay of $0.33T$ for reference source B, and $1.72T$ for source E.

Source Name	Period T	Expected TXQ delay	Measured TXQ delay	Measured IDT (SP)	Measured IDT (RB)
B	41	0.33	0.365	1.365	2.365
E	8	1.72	1.625	2.625	3.625

Table 8.1: Theoretical and measured worst case transmission queueing delay and measured tail inter-cell times after 1 switching stage (units of T)

The table shows that the estimated delay matches well with measured values – the discrepancy is approximately 1.5 cell times, which could be due to interrupt latency, and the byte-wise drain from the output FIFO. The measured inter-cell departure times were taken for a periodic reference stream. The measured IDT under SP is precisely as expected, with the initial period T increased to $1.365T$ by the queueing delay (for source B). Similarly under RB, the measured maximum IDT is $2.365T$ because of the mechanism shown in Figure 8.3.

The source period T is crucial in determining the performance of the system under each policy. For source E the TXQ delay is greater than T, whereas for source B it much less than T. Figure 8.6 shows how the properties of the originally periodic reference source B are degraded under SP. Both cells 2 and 3 experience TXQ delay, causing cell 4, already "early" on arrival, to be transmitted back-to-back after cell 3. The resulting cell stream exhibits both clumping and separation.

Figure 8.7 shows the effects of the TXQ delay on the periodic departure sequence from the RB scheduler, for source B. Although cell 4 arrives "early" due

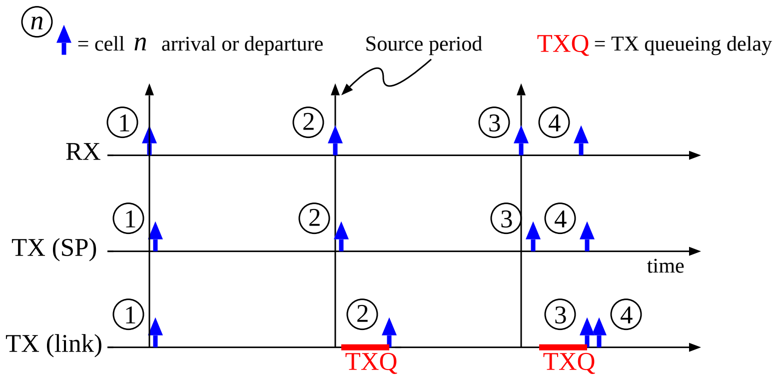


Figure 8.6: Effect of hardware transmit queue on inter-cell times under SP, source B

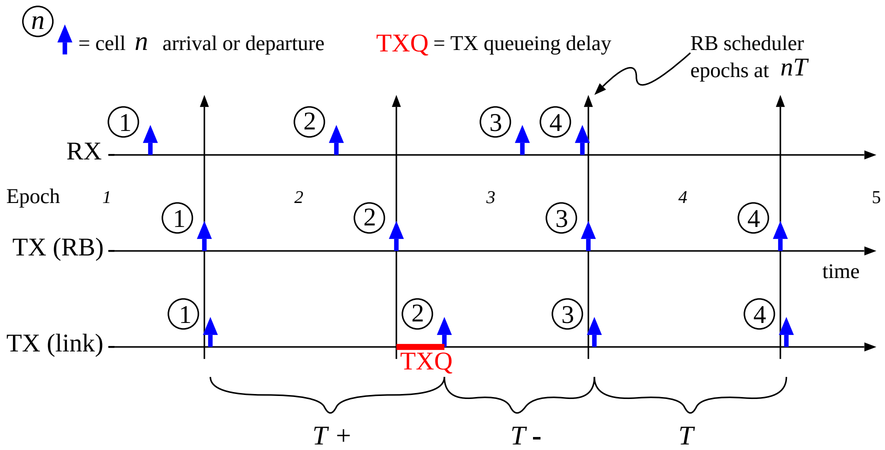


Figure 8.7: Effect of hardware transmit queue on inter-cell times under RB, source B

to queueing at the previous switch, the RB scheduler corrects this and transmits the cell on time. Cell 2, on departing the RB scheduler, finds a full hardware transmit queue and is delayed. The resulting output sequence from the switch has cells at intervals T^+, T^- and T, where $^{+,-}$ indicate a positive or negative delta. At the output measurement point an IDT distribution is expected to show a major peak at T, a smaller peak at $2T$ and a slight spread about the peaks due to the effects of the transmission queue. Figure 8.8 shows the same mechanism, but for source E. A tiny increase in the inter-arrival times between cells 1 and 2 can cause a huge difference in IDT because the TXQ delay is larger than T. It can also cause cells to depart back-to-back, causing higher CDV.

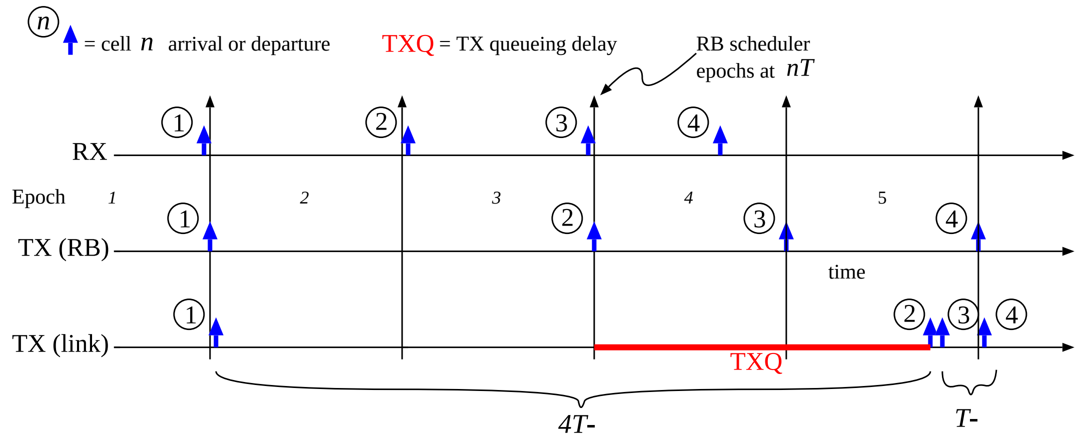


Figure 8.8: Effect of hardware transmit queue on inter-cell times RB, source E

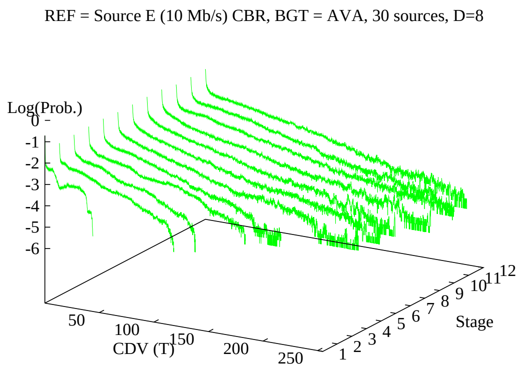
Fortunately under RB the effects of the FCFS queueing are corrected at each output by the strict periodicity of the scheduler, so the CDV does not degrade with additional stages (though the IDT will). Under SP, however, there is no such correction, and the effects are cumulative, resulting in higher CDV.

Source Name	Theoretical Max CDV (RB)	Measured Max CDV (RB)
B	0.28	0.29
E	1.745	1.75

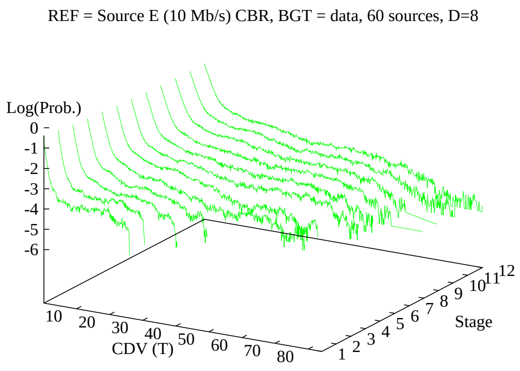
Table 8.2: Predicted worst case CDV under RB (units of T)

Figures 8.7 and 8.8 show how the maximum CDV arises for each source, under RB. For source B the closest two cells can move together is when the first experiences the maximum TXQ delay, and the second none. For source E the worst case CDV under RB occurs as shown in Figure 8.8 for cells 2, 3 and 4: Two cells depart back-to-back, followed by a third an interval $0.28T$ later. The resulting expected and measured CDV values are shown Table 8.2. The extremely close match lends confidence to the analysis of the TXQ delay.

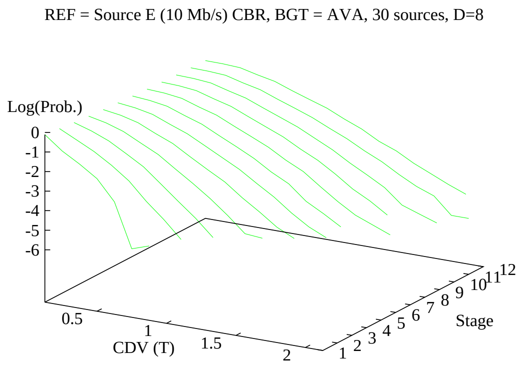
Finally, note that Figure 8.5 (f) shows 3 peaks at multiples of T. The transmit queueing delay of $1.72T$ will cause many cells to be widely separated as shown in Figure 8.8, so the peaks are not unexpected.



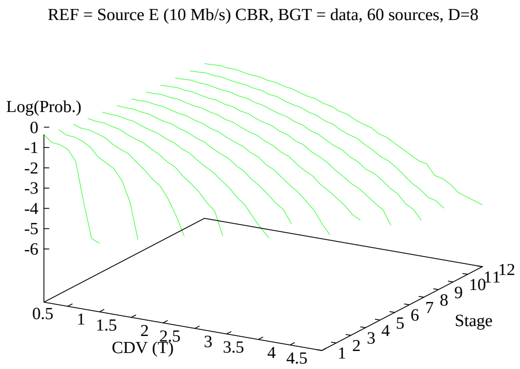
(a) FCFS



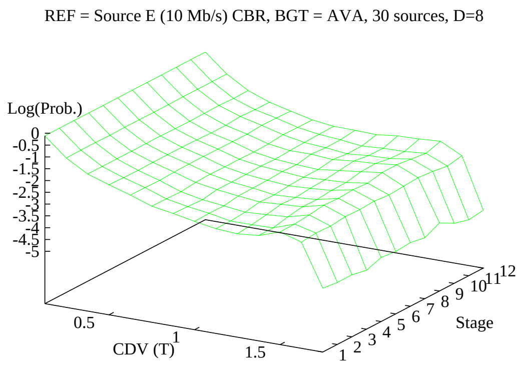
(b) FCFS



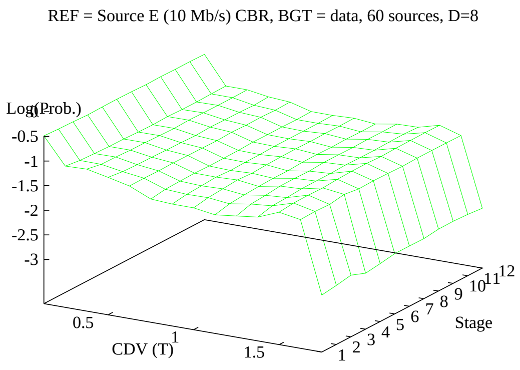
(c) SP



(d) SP



(e) RB



(f) RB

Figure 8.9: Comparison of CDV distributions for source E using FCFS (a,b), SP (c,d) and RB (e,f) policies, with AVA and data background traffic. Note the difference in CDV scales, required to make all graphs visible

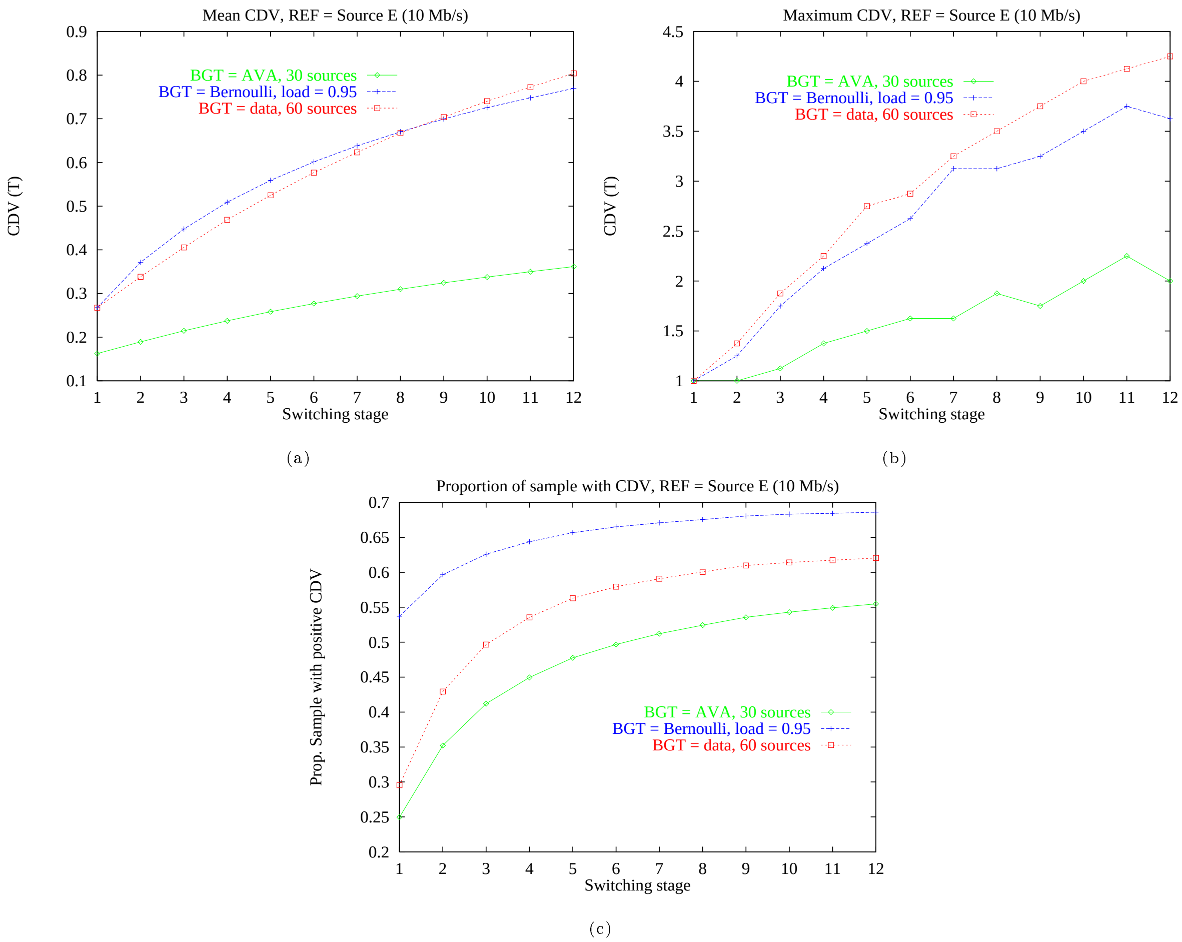


Figure 8.10: Mean (a) and maximum (b) CDV values, and proportion of sample with positive CDV (c), CBR source E under SP

Bursty Background Traffic

When the reference connection is multiplexed with bursty background traffic the two non-FCFS policies show a clear advantage over FCFS queueing. Figure 8.9 shows for source E the CDV under each policy with AVA and data background traffic. The maximum observed CDV is greatly reduced using SP, whilst under RB it is held constant as for Bernoulli traffic.

8.4.5.2 Performance of SP

Comparing Figure 8.5 (c), and Figures 8.9 (c) and (d) shows that under SP the performance is worst with data traffic, in spite of its lower load. This is because of its extreme burstiness, which causes reference cells to experience greater variance in delay at the output port. Figures 8.10 (a) and (b) display the mean and

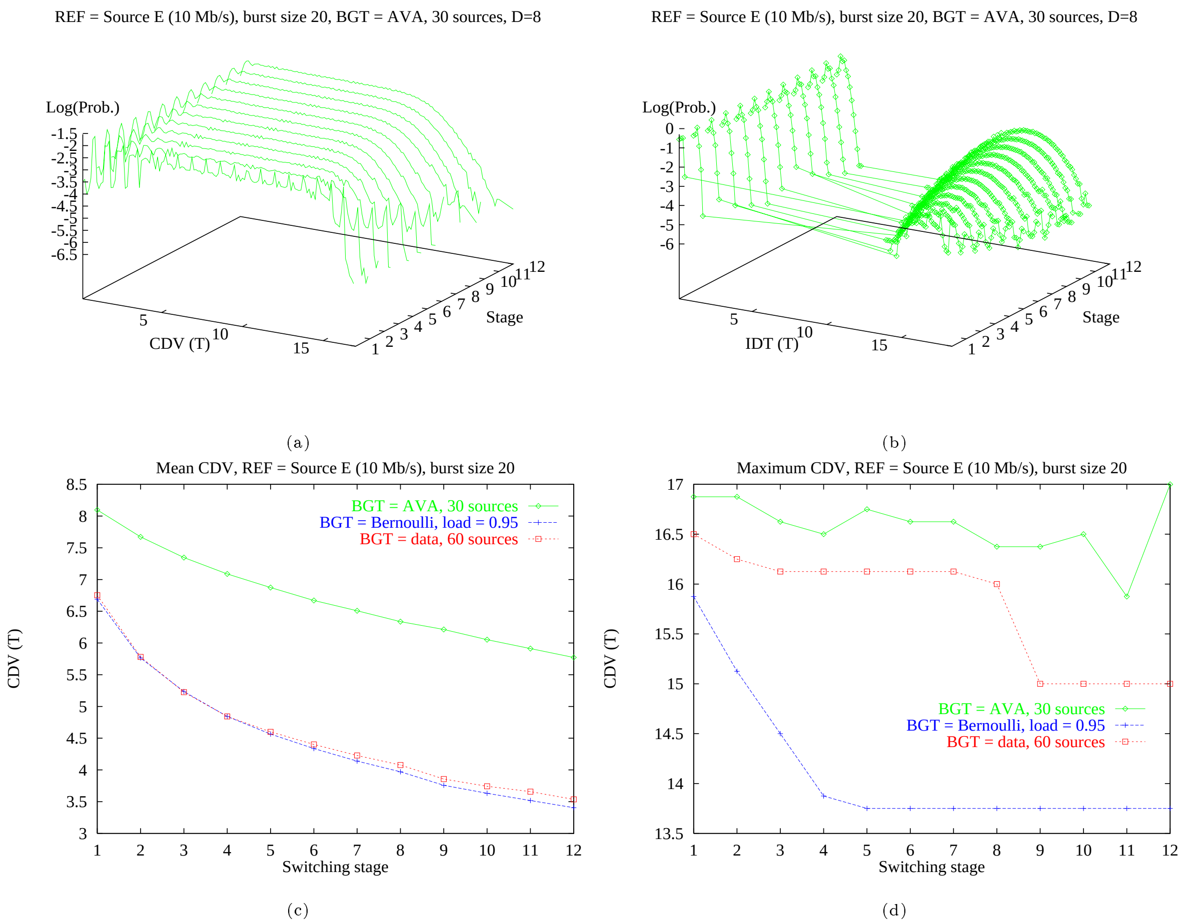


Figure 8.11: CDV (a) and inter-cell departure time (b) distributions, and comparison of mean (c) and maximum (d) CDV under SP for bursty reference source E

maximum CDV for source E under SP. The maximum CDV appears to grow in a bounded fashion, as would be expected given the results of Chapter 6. The proportion of the experimental sample experiencing positive CDV is plotted in Figure 8.10 (c).

Whilst SP performs well in comparison to FCFS, particularly with bursty background traffic, it fails to protect the network from badly behaved users. Figure 8.11 shows, by way of example, the CDV and IDT distributions measured using bursty reference source E, with AVA video traffic. Here the priority mechanism works against the network, preserving the bursts. Measurements indicate that burst expansion is more rapid using data traffic, followed by Bernoulli and AVA video, contrary to expectation given the total loads. We might well ask why there is any burst expansion at all. Bursts of priority cells arriving back-to-back from the reference connection should be received and transmitted back-to-back.

The answer lies in the mixing of cells in the non-blocking switch fabric – this tends to break up large bursts with background traffic. Further, the output queueing mechanism can delay a burst of priority traffic because the hardware transmit queue may be full when the burst arrives. Overall both the mean and maximum CDV decrease over the 12 switching stages, as shown in Figure 8.11 (e) and (f). Similar results were found using bursty source B.

8.4.5.3 Performance of RB

A comparison of Figure 8.5 (e), and Figure 8.9 (e) and (f) shows that the difference in CDV using different background traffic types is due to a slight change in the shape of the distribution. For Bernoulli traffic the distribution is log linear, suggesting that a departing rate-guaranteed cell experiences an exponentially distributed TXQ delay. With bursty background traffic the CDV distributions have two peaks, corresponding to almost no delay and a very large delay. The distributions are the same for each type of traffic even when the reference source used is bursty. This is confirmed, by way of example, in Figure 8.12 which compares the CDV and IDT distributions for the CBR and bursty reference sources.

The difference between the inter-cell departure time distributions of the CBR and bursty source experiments is due to a slight rate mismatch between the source and RB scheduler. Each burst contains 20 cells back-to-back in a period of $20T$. Once spaced out by the scheduler a gap slightly in excess of T remains between cell 20 and the first cell of the next burst. These cells tend to become increasingly widely separated over successive stages.

One final remark is in order with regard to the IDT distributions in Figure 8.12: in all cases they exhibit the expected peaks at multiples of T, but traffic is not evenly distributed around the peaks, tending to be on the "early" side, particularly with bursty background traffic. Figure 8.13 shows how the bursty background traffic can cause this phenomenon: cells 1, 2 and 3 experience TXQ delay, whilst cell 4 finds the transmit queue empty. Using bursty background traffic the transmit queue will be non-empty for long periods and then empty for long periods. Because cell 4 arrives slightly late it is only scheduled at the end of epoch 5. The arrival inter-cell gap between cells 3 and 4 of T^+ is thus changed to $2T^-$. These dynamics result in an IDT distribution with peaks at multiples of T, and a higher proportion of cells arriving "early" relative to the peaks than late, explaining the sharp drop in the IDT distribution at values of nT^+.

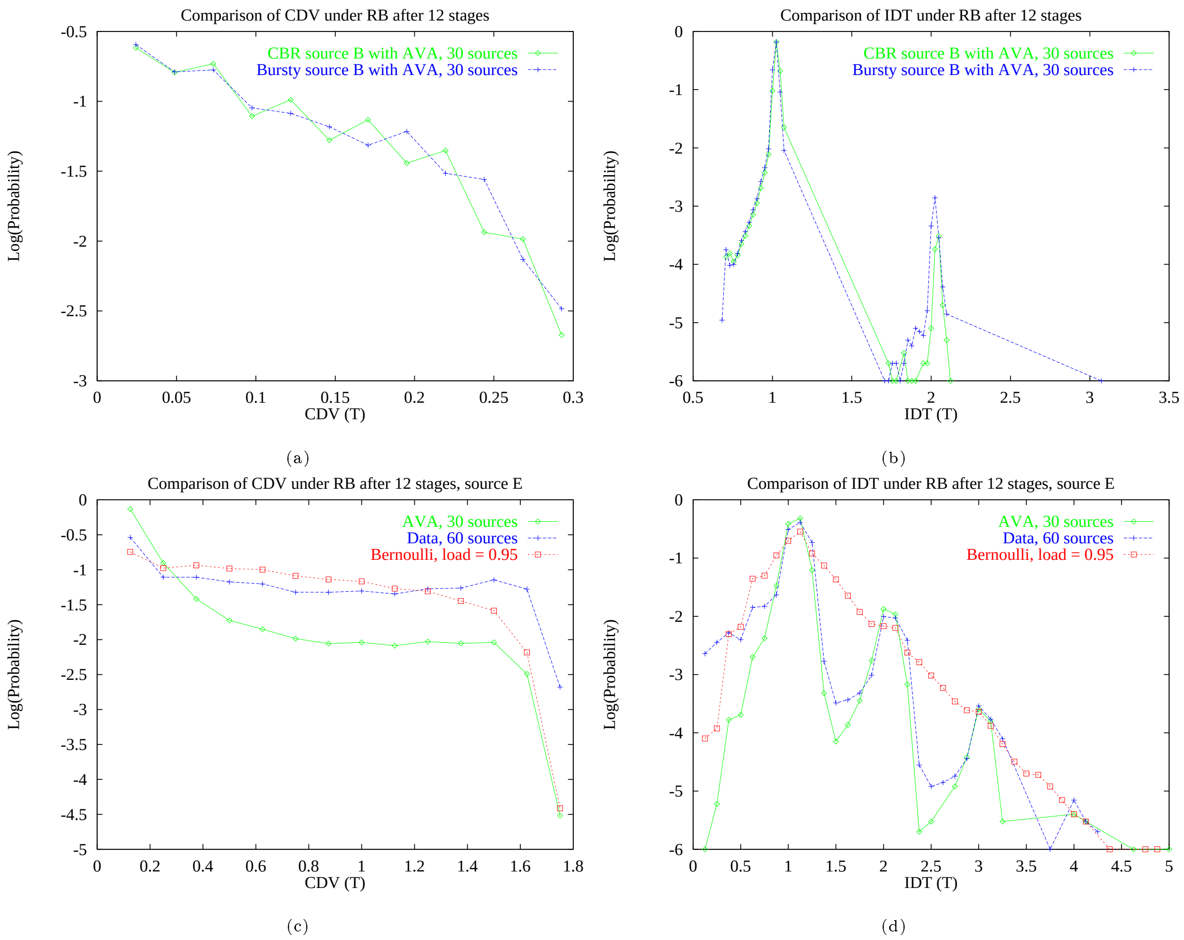


Figure 8.12: Comparison of CDV and inter-cell departure time distributions under RB after 12 switching stages, for CBR and bursty source B (a,b), and CBR source E (c,d)

8.4.6 Summary

Work conserving mechanisms can improve the performance of high priority types, but cannot prevent the occurrence of CDV. Non-work conserving policies are able to deliver guaranteed performance to different classes and can bound the CDV throughout the network. Whilst SP is attractive because of its simplicity, difficulty arises in allocating the traffic to the different priorities. Rate based schemes, particularly if positioned at network entry points, offer significant advantages, however they are more complex to implement. Given a RB controller at the network input, the CDV can be bounded within the network, as shown in earlier chapters for CBR traffic.

Analytical models of scheduling disciplines fail to take into account the effects

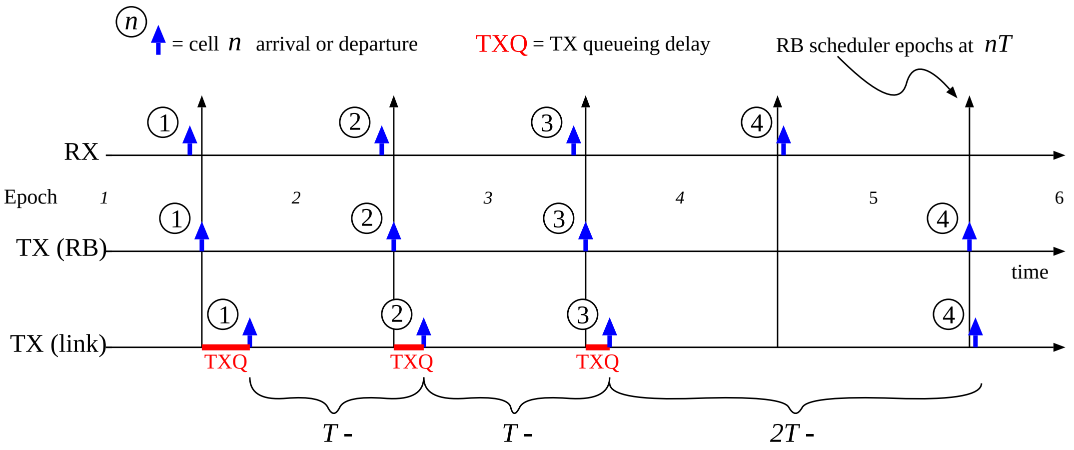


Figure 8.13: Effect of bursty background traffic on inter-cell times for the RB policy

of hardware properties, such as FIFOs used for rate adaptation. In the case of Fairisle these cause significant performance differences from the theoretical predictions. Whilst it might be tempting to suggest that this is a Fairisle-specific problem, it should be clear that all ATM equipment will incorporate small buffers to permit rate adaptation. If the queueing delay these cause constitutes a significant proportion of the source rate, then they may seriously affect the performance delivered.

Finally, with small buffer sizes, FCFS queueing can deliver adequate performance throughout the network. With large buffer sizes, however, and un-shaped traffic sources, it is necessary to employ a more intelligent queueing policy to deliver adequate performance.

8.5 Allocation of Buffer Space

The previous sections have studied the advantages which non-FCFS queueing policies confer on delay-sensitive traffic. Large output buffers were used to ensure that the experiments were lossless. In all ATM systems, irrespective of the scheduling policy, some mechanism is required to allocate buffer space to different connections: to achieve maximum utilization of the available bandwidth, the queue should be kept non-empty, without excessively increasing the delay. Fairness requires that each connection is allocated sufficient buffer space for its needs, hence domination of the buffer by one class should be avoided.

A naïve approach would be to partition the buffer between the connections. This is similar to peak rate allocation of bandwidth, and requires large buffer sizes [Kröner91]. Partial Buffer Sharing (PBS) [Hébuterne89, Petr90, Bonomi90] discriminates by class, assigning a maximum queue length to the delay sensitive traffic – an arriving cell which finds its share of the buffer space full is discarded. There is no threshold associated with the loss sensitive traffic. Threshold based discard performs significantly better than naïve discard from the tail, but choice of the thresholds is critical [Petr90]. Alternatively, loss sensitive traffic may be given Pushout Priority (PP): an arriving loss sensitive cell may discard a delay sensitive cell when the buffer is full, if there is one. The cell to be discarded may be selected from either the head or the tail of the queue, or randomly. Discarding from the head may be preferable to discarding from the tail: from the network's point of view, traffic which has already consumed resources is more valuable, however delay-sensitive cells at the front of a queue have already been delayed, and have a higher chance of discard at the receiver [Yin90]. Discarding from the tail, however, has a lower cell buffer requirement for the same total cell loss [Lucantoni90]. Random discard performs better than tail dropping, but is more difficult to implement [Yuan89]. It is also unsuitable for traffic with a higher layer frame structure, such as packet data. Finally, buffer allocation can be combined with service priority to accommodate both delay and loss sensitive traffic [Gravey90, Roberts92].

To investigate the performance of the different buffer allocation policies, three were implemented on the Fairisle testbed. The following subsections describe the algorithms, their implementation details and the performance under each.

8.5.1 Choice of Policies

The following buffer allocation policies were implemented:

- **Instant (or tail) discard**: this policy is the obvious baseline against which to compare more sophisticated discard policies – an arriving cell is discarded immediately, irrespective of its class, if it finds the buffer space to be full.

- **Partial Buffer Sharing**: Class 1 has High service Priority (HP) whilst class 2 has loss priority. An arriving class 1 cell is discarded if it finds that the queue length of class 2 cells exceeds a threshold, or if there is no buffer space. A class 2 cell joins the queue if there is buffer space.

- **Pushout**: Class 2 has pushout priority: an arriving class 2 cell which finds the buffer full discards a class 1 cell if there is one, otherwise it is lost. An arriving class 1 cell joins the queue if there is buffer space. Class 1 has service priority.

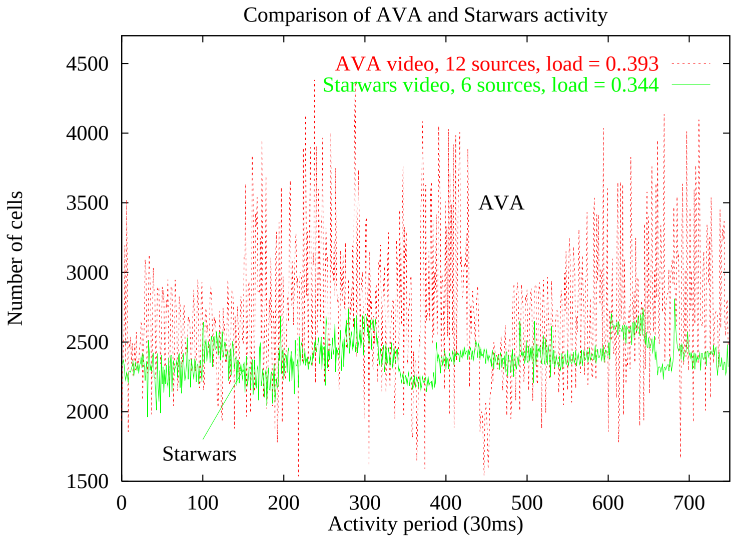


Figure 8.14: Comparison of AVA and Starwars video activity for traffic mix A

8.5.2 Traffic Characteristics

Two VBR video sources were used as the two classes of traffic, namely an AVA video trace, and a trace of the "Starwars" movie.

8.5.2.1 The Starwars Data

The Starwars data set was produced by Garrett and Vetterli at Bellcore, and has been studied in some detail, for example [Garrett94, Chong94, Crosby95]. It comprises the information content in bytes per frame for about 2 hours of the film, as transmitted by a DCT based codec similar to JPEG. The byte count per frame is broken down into "slices" of 16 video lines. With 30 slices per frame and 24 frames per second, one slice represents the information transmitted in about 1.4 milliseconds.

The traces used in these experiments were constructed by encoding each slice as a single AAL5 PDU; cells from each PDU are transmitted evenly spaced throughout the slice time. This reduces the peak rate of the source from the line rate (100 Mb/s) to about 24.2 Mb/s for the worst slice. The mean rate is about 5.3 Mb/s.

Figure 8.14 compares the activity of the two sources for traffic mix A. The AVA traffic is far more bursty in spite of having double the number of sources. Two traffic mixes were used, with measured parameters shown in Table 8.3. Each source ran a trace of 1.5 minutes (AVA) or 2 minutes (Starwars) duration, with random subsampling to remove correlation.

Traffic mix	AVA			Starwars			Total load
	No.	Load	Prop. total	No.	Load	Prop. total	
A	12	0.393	0.534	6	0.345	0.466	0.737
B	6	0.196	0.276	9	0.522	0.724	0.718

Table 8.3: Experimental traffic mixes

8.5.3 Experimental Method and Implementation Issues

The experiments were carried out using a single output buffered switch. In each experiment all sources were multiplexed into one output queue, whose length and discard policy were chosen to permit comparison of the different mechanisms. In each experiment approximately 5×10^7 cells were transmitted (variation was caused by the random seeding of the traffic generators), composed of the two traffic types in proportion to their fraction of the total load. In each experiment the absolute number of cells was recorded to enable accurate assessment of the loss probability.

Two features of the Fairisle equipment influenced the implementation of the discard policies, and are discussed in the following subsections.

8.5.3.1 Cell Discard Mechanism

On the FPC3 a cell which is to be discarded is enqueued for transmission by placing it on the hardware transmit queue and setting a *discard* bit. When the cell buffer reaches the head of the queue the Xilinx returns it directly to the free pool. This operation takes a full cell time and causes two problems: first, it is impossible to discard a cell from the tail of a queue when the system is busy (i.e. when the hardware transmit queue is full). Second, discarding takes a cell time, reducing throughput and affecting observations of the queue length. The first problem was overcome by enqueueing cells to be discarded in the usual way, but with a modified retry count. When a cell marked in this way is transmitted by the hardware it is discarded. Care must be taken when recording the queue length, and to ensure that even with a high loss rate sufficient buffers will remain in the free pool to accommodate arriving traffic.

8.5.3.2 Fairisle Queue Structure

A second problem results from the choice of a singly linked list as the data structure representing a queue. This structure is supported by the hardware. Under the pushout policy, an arriving loss sensitive cell should discard the delay sensitive cell at the tail of its queue. To do so involves processing the list representing the delay sensitive queue, and is time consuming. Instead the pushout mechanism implemented selected the cell at the *head* of the queue for discard, scheduling that queue for immediate transmission to discard the selected cell as quickly as possible.

8.5.4 Results and Discussion

Figure 8.15 (a) investigates tail discard for two different queueing policies, namely FCFS and SP, showing the cell loss ratio (CLR) for each traffic type in mix A, for increasing buffer sizes. Irrespective of the policy, the AVA traffic sustains greater loss than the Starwars traffic. This is confirmed in Figure 8.15 (b) which shows that the proportion of the total loss sustained by each type is at odds with the ratio of their loads: the AVA traffic suffers in excess of 70% of the loss but represents 53% of the load. This surprising result is contrary to expectation: intuitively one might expect the loss to be apportioned according to the load ratio. It occurs because the AVA traffic is far more bursty than the Starwars traffic. An arriving burst of AVA cells will find the buffer partially occupied by other traffic, and will frequently be unable to fit into the remaining space, causing high loss. Phrased differently, the loss probability for an arriving cell is a function of the burst size distribution of its traffic type. This means that bursty sources experience greater loss for the same load. Frater [Frater94] observed as a result of an analytical study that loss is determined by the weight of the tail of the queue length distribution for each class. In this case the AVA traffic has a far heavier tail, and suffers as a result.

Returning to Figure 8.15 (a), under SP, the CLR decreases more rapidly than under FCFS in both cases. The total CLR (not shown) decreases in the order SP with HP Starwars, FCFS, and SP with HP AVA. Clearly there is an advantage to giving the bursty AVA traffic service priority.

Investigating PBS and tail discard with service priorities, Figure 8.15 (c) shows the effect of introducing a threshold of 35 cells for the service priority class, on the CLR of the AVA traffic. Figure 8.15 (d) shows the same results expressed as the proportion of AVA traffic lost. When AVA has service priority, it suffers lower cell loss than when Starwars has priority, irrespective of the threshold. With the threshold, at a low buffer size the buffer size is the limiting constraint on the AVA loss, but for larger buffer sizes the threshold is the constraint. When Starwars is given service priority the AVA traffic suffers significantly greater loss, particularly when the Starwars traffic is not limited by the threshold, demonstrating that the threshold can protect the non-service priority class.

Figure 8.15 (e) shows the tail of the queue length probability distribution when AVA is given service priority. Notice the raised probability just preceding each buffer limit. An arriving cell has a higher probability of seeing a queue

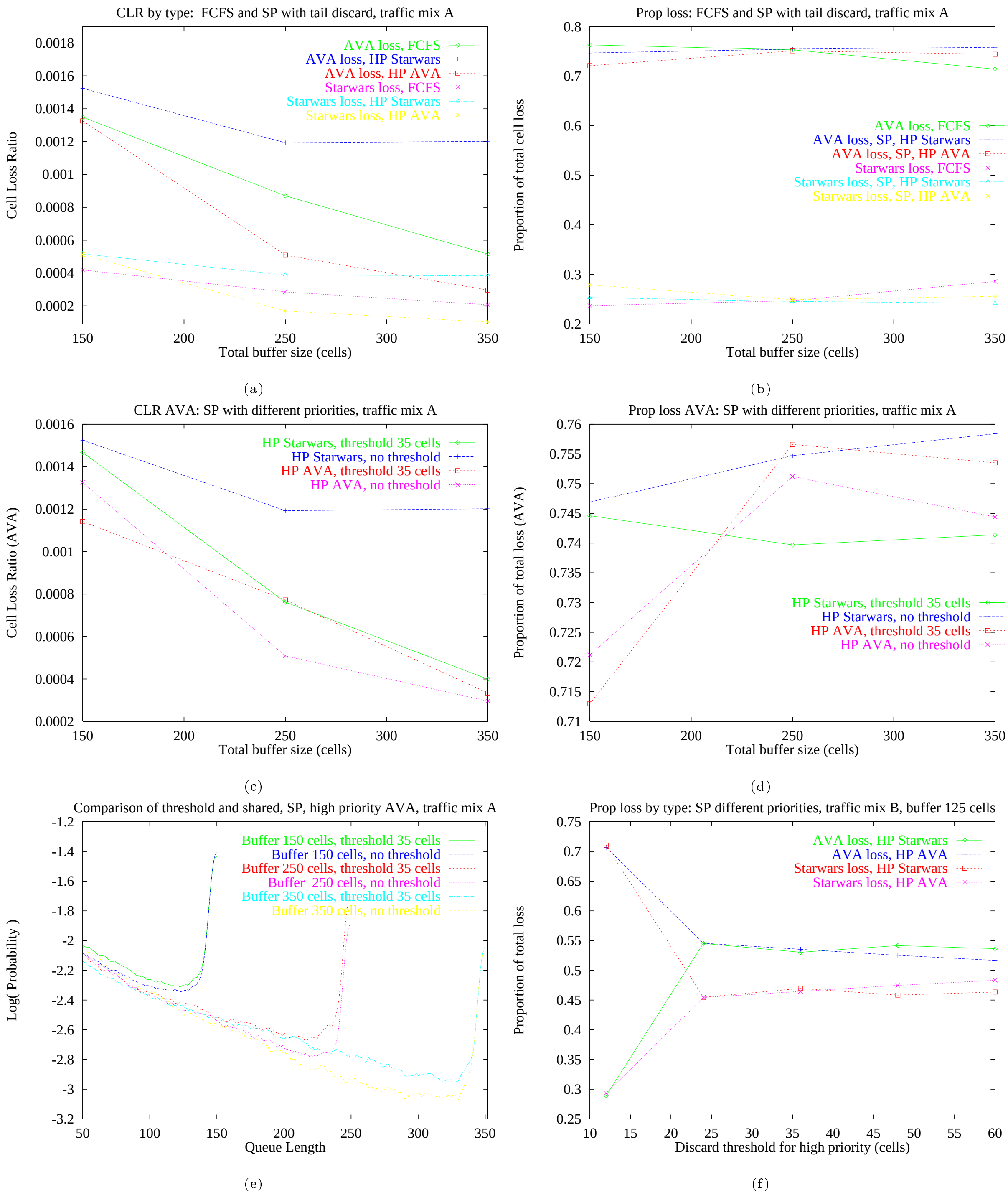


Figure 8.15: Results for FCFS and SP queueing with thresholds

whose length is close to the limit than significantly below the limit because the queue is often close to full. Using thresholds the total CLR is higher in all cases, because the threshold discards a significant proportion of the AVA traffic.

Figure 8.15 (f) and Figure 8.16 (a-d) investigate PBS using traffic mix B. Figure 8.15 (f) plots the proportion of loss sustained by each traffic type, for a range of threshold values when each type is given service priority, and a total buffer size of 125. With a threshold of 12 the division follows the ratio of the two loads, but with larger thresholds the proportion of loss becomes more evenly distributed. The threshold gives good control when it is extremely tight. As the thresholds increase, however, an interesting change occurs: irrespective of the threshold, approximately 55% of the discarded traffic is AVA traffic. Figure 8.16 (a) shows the tail of the queue length distribution for the same system but giving AVA high priority. It gives an indication of the total CLR under the different thresholds, for the fixed buffer size. Contrary to expectation the loss is greater with higher thresholds, and this explains the convergence of the proportions of loss: for larger thresholds the AVA traffic will occupy a greater proportion of the buffer, increasing the loss of Starwars traffic more than it decreases the loss of the AVA traffic.

Holding the threshold constant at 12 cells in Figure 8.16 (b), and varying the total buffer size, the proportion of loss of the high priority traffic increases, because there is more space for the low priority traffic. This can also be seen in Figure 8.16 (c) which shows the CLR in each case, for the same experiment. The total CLR decreases with increasing buffer size, as expected; however, the proportion of loss of the high priority traffic increases at the same time.

Finally, again varying the thresholds, but showing results for AVA high priority only, Figure 8.16 (d) shows for three different buffer sizes, the proportion of loss of each traffic type. With a low buffer size of 25 cells, and threshold of 12 cells, the buffer size is the dominating constraint and the threshold is unable to adequately separate the two classes. A buffer of 75 cells provides more room for the low priority class (Starwars) and for the same threshold gives it better protection. A threshold of 25 exhibits the same proportion of loss irrespective of the buffer size. For a buffer of 125 cells, the low threshold splits the traffic according to their proportions of the total load, with higher thresholds tending to apportion loss more evenly. For the buffers of size 75 and 125, the lowest proportion of the high priority traffic is lost with a threshold equal to half of the total buffer size.

Figure 8.16 (e) contrasts PBS with pushout for mix A, showing the proportion of loss sustained by AVA traffic when it has service priority under three discard policies, namely tail discard, PBS with a threshold of 35, and with pushout priority for Starwars. Under pushout virtually all loss is sustained by the AVA traffic, and none by Starwars, demonstrating that pushout can completely protect

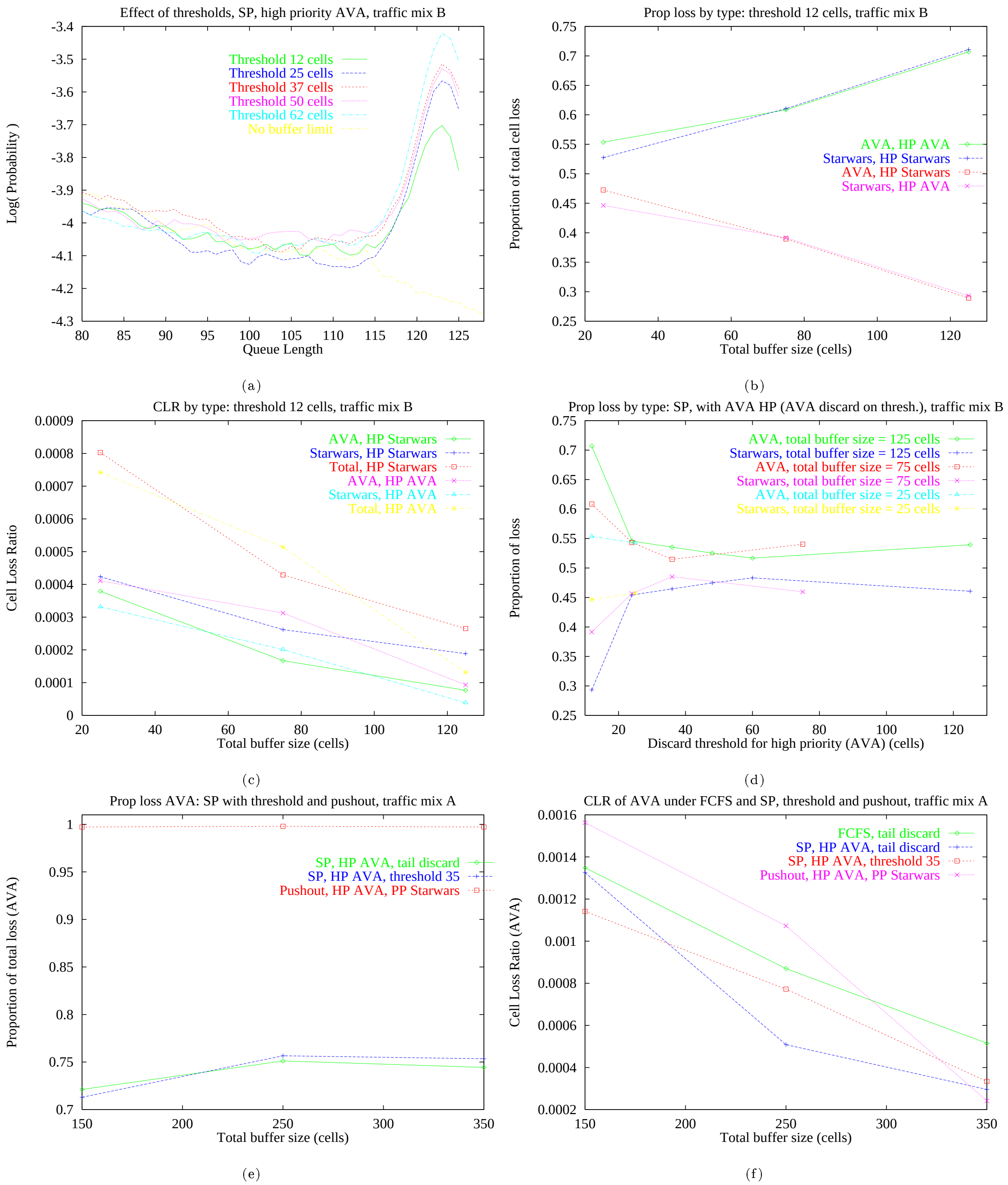


Figure 8.16: Results for SP with thresholds and pushout

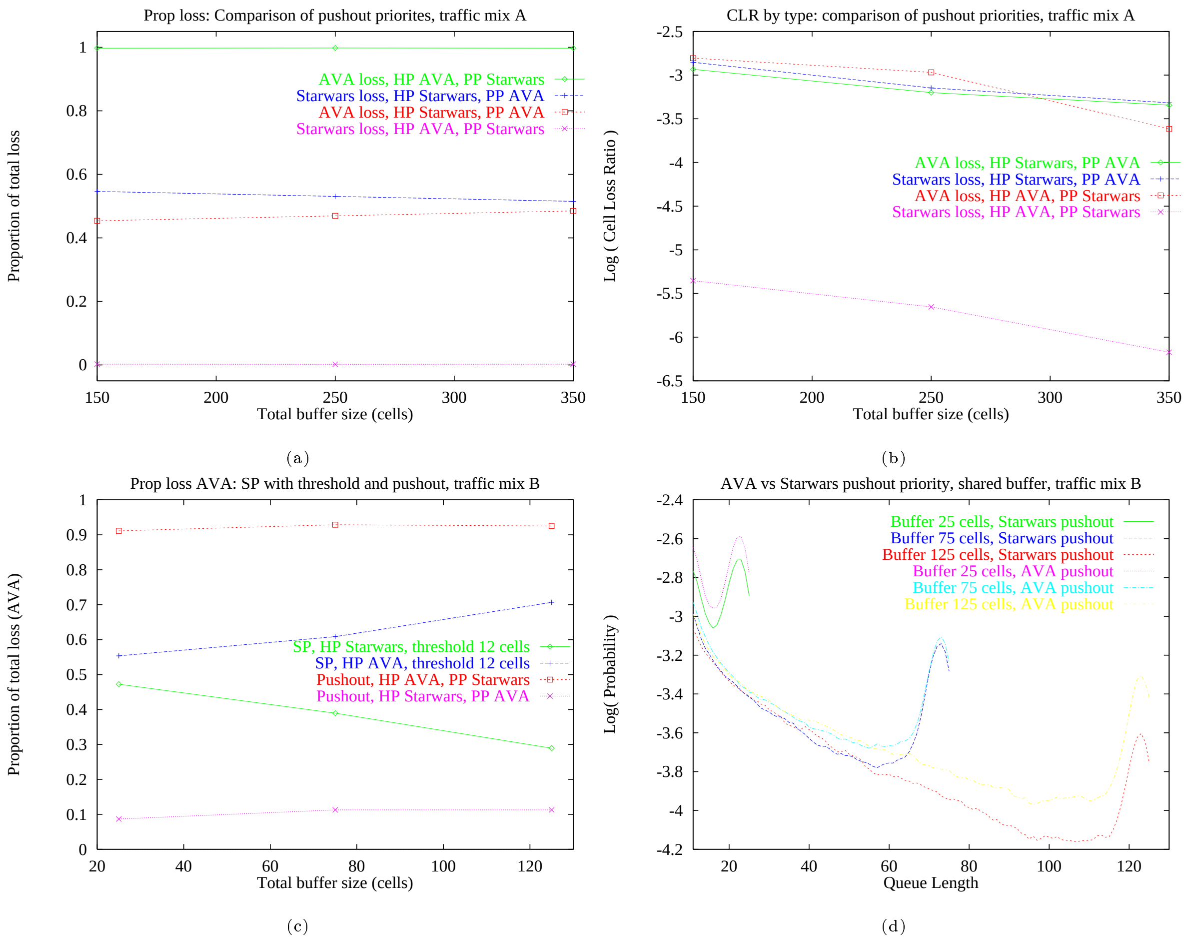


Figure 8.17: Comparison of pushout priorities

the loss sensitive traffic in this case. In spite of this, the total CLR with pushout is roughly the same as under the other schemes, as shown in Figure 8.16 (f). With a smaller buffer size the AVA traffic sustains higher CLR under pushout than using the threshold or tail discard. Here the buffer size is the limiting constraint, and not the threshold. With a buffer size of 350, the pushout mechanism appears to deliver a slightly lower total CLR.

Surprisingly, when the service priority is reversed, the pushout mechanism cannot adequately protect the low priority traffic. Comparing the two cases, Figure 8.17 (a) shows that when Starwars has pushout priority, a Starwars cell arriving to a full buffer always finds queued AVA traffic to discard. When AVA has pushout priority, however, the mechanism fails to partition the two types. At a low buffer size the split between the two is roughly in proportion to their load contribution. At higher buffer sizes the pushout mechanism tends to allocate the loss evenly. Why is this? It seems that the highly bursty nature of the AVA traffic is the cause. Since it is so bursty, it is the major contributor to the tail of the queue length distribution. Thus, an AVA cell arriving to a full buffer is likely to find most of the buffer occupied by AVA cells. A burst of AVA will discard all queued Starwars traffic, but if the burst is large enough, insufficient Starwars traffic will be present, resulting in AVA loss. This, coupled with the fact that the AVA traffic constitutes a greater portion of the total load, explains why the pushout mechanism fails to work in this case. Figure 8.17 (b) shows the corresponding CLR in each case. When the pushout mechanism functions as intended, the CLR for Starwars is tiny, however with AVA pushout, it is unable to separate the two classes.

Figure 8.17 (c) compares pushout and PBS for mix B, again showing the proportion of AVA loss. Reversing the pushout priority approximately reverses the proportion of loss of the AVA traffic. Here, too, the pushout mechanism fails to totally isolate the pushout priority traffic from the service priority traffic, nevertheless it easily outperforms the threshold mechanism, even with an extremely tight threshold and large buffer sizes. This leads to the conclusion that the pushout mechanism is to be preferred when greater protection for the non service priority class is desired. The burstiness of the pushout class is extremely important – it can still sustain heavy loss if the bursts are too large and if it represents a significant proportion of the total load.

Finally, Figure 8.17 (d) shows the difference in the queue length distributions when each class is given pushout priority. In all cases the distribution for Starwars pushout is lower than for AVA pushout, for the same buffer size. This corresponds to a higher total CLR when AVA has pushout, and results, once again, from its greater burstiness.

8.5.5 Summary

The results of the study can be summarised as follows:

- Of the three mechanisms, pushout priority performs best in separating the loss of the two classes, followed by threshold mechanisms with tight thresholds on the service priority class. Tail discard functions poorly.
- The burstiness of the traffic types multiplexed is critical in determining the portion of loss sustained by each, under all buffer allocation policies. More bursty traffic suffers loss which is disproportionate to its portion of the load.
- When the more bursty traffic has service priority the total CLR is lower than when it has low priority and under FCFS.
- Under PBS the determination of the appropriate thresholds is difficult, because the proportion of loss sustained by each class is a function both of

the threshold value and the total buffer size.

- Under PBS tight thresholds offer the best control, and decrease the total CLR. There appears to be a crossover point between the buffer size and threshold as the dominant constraint when apportioning loss. Its value depends on the load ratio of the traffic types. For mix B the crossover occurs with a threshold of about half the total buffer size. For mix A, even a tight threshold of 35 cells cannot protect the AVA traffic when Starwars has service priority, because the AVA traffic contributes over half of the total load.

- The proportion of the load contributed by each type is critical, irrespective of the buffer allocation policy. For mix B the pushout mechanism can protect the bursty AVA traffic because it represents only 28% of the load. For mix A, however, even the pushout mechanism fails to protect the AVA traffic. This observation concurs with [Roberts92] which suggests that pushout traffic should be less than 0.2 of the total load.

- Irrespective of the service discipline and buffer allocation policy the total CLR remains roughly the same under all mechanisms.

Experiments with these buffer allocation policies and using a RB scheduler were attempted, however CPU limitations on the FPC3 limited the ability of the system to keep track of loss and queue lengths whilst executing the rate based schedule.

8.6 Concluding Remarks

This chapter has considered the problem of cell level scheduling and buffer allocation at input and output buffered switches. Experimentation with both work conserving and non-work conserving policies demonstrates that the latter offer better performance at a higher implementation cost. However, given the findings of previous chapters, such policies are only required at the network boundaries: the CDV will be bounded within the network.

Buffer allocation is a difficult problem. This chapter has illustrated that the choice of service priority affects the loss of different classes, and that the selection of thresholds and pushout priorities is sensitive to the properties of traffic concerned and their contribution to the total load. Analytical modelling reports to date have failed to take these variables into account.

Chapter 9

Conclusions and Scope for Future Work

In ATM networks control must be exercised on different time scales, and different solutions are appropriate in each case. On the call time scale an out of band control plane which supports diverse communication services is required. On the cell and burst time scales, dynamic control algorithms must be located in band to successfully control multiplexed traffic. This dissertation has investigated mechanisms on both time scales, contributing to debates on both architectural and performance issues in ATM networks.

9.1 Review

Chapter 3 identified several drawbacks of the current approach to the development of control architectures for telecommunication networks. These include their monolithic design and complexity which result from the aim of having a single signalling protocol which is capable of expressing all service needs. It also identified several failings of a simple metasignalling protocol which makes naive assumptions about network behaviour. A novel open, flexible architecture, the MSN-CMA, was proposed which facilitates the construction of complex call components using basic service primitives. The architecture enables network control functions to be located outside the network: each entity cedes control to a designated master, implemented as a set of distributed programs. The architecture enables the incorporation of different equipment types, and "dumb" ATM devices.

Chapter 4 described the MSN-CMA user-network signalling protocol in detail, stressing importance of its robustness in the face of errors. The protocol is RPC-based; since IDLs do not permit a detailed specification of the operational semantics of an interface, the protocol was formally specified using TLT. Successful verification of the protocol gives greater confidence in its correct operation.

Chapter 5 described the implementation of the MSN-CMA. On Unix the connection management functions were encapsulated in a single, user space daemon which communicates with the kernel via a special control socket. This novel approach has several benefits, including reduced kernel complexity and code size, ease of development and portability of the management code. Performance measurements showed that the implementation performs sufficiently well for use in a lightweight-connection environment.

An implementation of the entity-specific control interfaces within the Wanda micro-kernel was described; user space control functions communicate with the kernel using a dedicated control protocol. Key to the success of the implementation was the flexibility and efficiency of the cell level scheduler on the Fairisle port controller, which arbitrates access to the fabric and cell buffers. This permitted the development of a range of cell scheduling and buffer allocation policies and sophisticated measurement facilities for ATM connections. Chapters 6, 7 and 8 documented use of these facilities in a series of experiments which assessed the performance delivered to ATM connections both in an unregulated network and under control policies which aim to deliver guaranteed performance.

Chapter 6 compared the results of two analytical models of the CDV experienced by CBR ATM traffic over several switching stages with experimental results using typical LAN ATM traffic. To date no measurements from multi-stage ATM networks have been reported in the literature. The experimental work confirms the broad conclusions of the analytical models: the CDV grows in a bounded fashion and is a function of the types of traffic multiplexed, the total multiplexed load, and the proportion of the load contributed by the CBR connection. The analytical models fail to bound the CDV experienced using highly bursty background traffic. The results suggest that unless traffic is shaped on entry to the network, its performance will degrade badly.

Chapter 7 continued the study of network performance, investigating two analytical models of burst expansion, and showing that as bursts of traffic proceed through an ATM network they expand at a rate which is dependent on the rate of the reference connection, the total load and the background traffic type. Measurements using Bernoulli background traffic confirmed the accuracy of two analytical approximations.

Finally, Chapter 8 investigated several proposed mechanisms for guaranteed performance. Static priority and rate based schedulers significantly improve the performance of delay sensitive traffic over FCFS queueing. Priority scheduling is attractive because of its low cost, but allocation of traffic to the priority levels is difficult. Rate based queueing performs best, providing guaranteed low CDV. It has a high implementation cost unless supported by the hardware. Delay sensitive traffic experiences CDV irrespective of the policy, due to hardware FIFO buffers which implement rate adaptation. All ATM equipment will incorporate such features and both designers and modellers should take their effects into account.

Using two VBR video sources of different burstiness, and under a combination of scheduling rules and buffer allocation policies, experiments showed that the performance of the buffer allocation mechanisms is sensitive to the traffic types and their relative loads. Bursty traffic experiences disproportionately high loss compared to smoother traffic of the same load. Using threshold-based discard, choice of the thresholds is critical to performance; these will be difficult to determine in practice. Performance of pushout priority is sensitive to the load ratio and cannot protect the pushout class if it is too bursty or represents a significant portion of the load.

9.2 Future Work

The MSN-CMA design incorporates only simple connection primitives. Multicast must be implemented on top of the architecture – this is the subject of continued work at the Computer Laboratory. The design does not incorporate features for multi-connection calls. This should be added to enable full support of multimedia applications. Requirements for an ATM API, and for additional control features should be analysed; the DCAN project is building on the MSN-CMA to develop a fully fledged implementation of a control environment for multimedia applications.

The performance studies of the CDV in ATM networks, burst expansion and the use of non-FCFS scheduling policies and buffer allocation mechanisms are, in a sense, preliminary. Further work should be performed to develop improved analytical models of CDV, which reflect the observations. Further, the effects of shaped background traffic on the reference connection should also be investigated. Similar comments apply to the study of burst expansion.

Finally, the experiments with non-FCFS scheduling policies were based on simple implementations of typical policies. An in-depth investigation of more sophisticated mechanisms should be undertaken, comparing the performance delivered to multiple connections, the buffer requirements of each and the implementation complexity. Further work is required also on buffer allocation policies. Those used fail to adequately protect the different traffic classes under certain circumstances. Additional work should aim to establish procedures for dynamically parameterising the algorithms in response to changes in the multiplexed load. Further use of the Fairisle CLS in a study of CAC algorithms is in progress; some results are reported in [Crosby95].

Bibliography

[Aawater91] G.A. Aawater and F.C Schoute. *Optimal Queueing Policies for Fast Packet Switching of Mixed Traffic*. IEEE JSAC, 9(3):458–466, April 1991. (p 145)

[Accetta86] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, and M. Young. *Mach: A New Kernel Foundation for UNIX Development*. Technical Report, School of Computer Science, Carnegie Mellon University, August 1986. (p 32)

[Ahmadi90] Hamid Ahmadi and Roch Guérin. *Bandwidth Allocation in High Speed Networks Based on the Concept of Equivalent Capacity*. In Broadband Technologies, New Jersey, October 1990. Proceedings Seventh ITC Specialist Seminar. (p 46)

[Anderson92] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P Thacker. *High Speed Switch Scheduling for Local Area Networks*. In Proceedings ASPLOS-V, October 1992. (p 142)

[ANSA89] ANSA. *The ANSA Reference Manual*. Architecture Projects Management, release 01.00 edition, March 1989. (pp 7, 22, 61, 68)

[ANSA92] ANSA. *ANSAware Version 4.1 Manual*. Architecture Projects Management, 1992. (p 7)

[ATMF94] ATM Forum. *ATM User-Network Interface Specification Version 3.1*, 1994. (pp 27, 65)

[Bae91] J.J. Bae and T. Suda. *Survey of Traffic Control Schemes and Protocols in ATM Networks*. Proceedings IEEE, 79(2):170–189, February 1991. (pp 104, 143)

[Barnard95] Dieter Barnard and Simon Crosby. *The Specification and Verification of an Experimental ATM Signalling Protocol*. In Proceedings Fifteenth International Symposium on Protocol Specification, Testing and Verification (PSTV '95), Warsaw, Poland, June 1995. (pp i, 64)

[Birrell84] A. Birrell and G. Nelson. *Implementing RPC.* ACM Transactions on Computer Systems, 2(1), February 1984. (p 23)

[Birrell94] Andrew Birrell, Greg Nelson, Susan Owicki, and Edward Wobber. *Network Objects.* Technical Report 115, Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, California, 1994. (p 22)

[Black94a] Richard Black and Simon Crosby. *Experience and Results from the Implementation of an ATM Socket Family.* In Proceedings USENIX Winter Technical Conference, San Francisco, January 1994. (pp i, 76, 80)

[Black94b] Richard Black, Ian Leslie, and Derek McAuley. *Experiences of building an ATM switch for the Local Area.* In Proceedings ACM SIGCOMM, volume 24(4), September 1994. (pp 14, 111, 142)

[Black95] Richard Black. *Explicit Network Scheduling.* Technical Report 361, Cambridge University Computer Laboratory, February 1995. Ph.D. dissertation. (pp 23, 89, 93)

[Bonomi90] F. Bonomi, L. Fratta, S. Montagna, and R. Paglino. *Priority on Cell Service and on Cell Loss in ATM Switching.* In Broadband Technologies. Proceedings Seventh ITC Specialist Seminar, October 1990. (p 162)

[Boyer90] P. Boyer. *A Congestion Control for the ATM.* In Broadband Technologies, New Jersey, October 1990. Proceedings Seventh ITC Specialist Seminar. (pp 10, 57)

[Brinkmann94] Axel Brinkmann, Robin Macfadyen, Andrew Atkinson, and Eric Scharf. *Validation of Delay and Cell Loss Models for ATM Switches.* In Proceedings RACE EXPLOIT Traffic Workshop, Basel, Switzerland, September 1994. (p 96)

[Bubenick91] Rick Bubenick, John De Hart, and Mike Gaddis. *Multipoint Connection Management in High Speed Networks.* In Proceedings IEEE INFOCOM, pages 59–68, Bal Harbor, Florida, April 1991. (pp 8, 70)

[Bubenick92] Rick Bubenick, John De Hart, and Mike Gaddis. *Connection Management Access Protocol (CMAP) Specification, Version 2.11.* Technical Report WUCS-92-01, Washington University at St. Louis, St. Louis, Missouri, USA, May 1992. (pp 57, 70)

[Burrows93] M. Burrows. *A Prototype Elapsed Time Profiler for Alpha OSF/1 and MIPS Ultrix.* Technical Report, Digital Equipment Corporation, Systems Research Center, 1993. To appear. (p 82)

[Chen89] T.M. Chen, J. Walrand, and D.G. Messerschmitt. *Dynamic Priority Protocols for Packet Voice.* IEEE JSAC, 7(5):632–643, June 1989. (p 144)

[Chipalkatti89] Renu Chipalkatti, James F. Kurose, and Don Towsley. *Scheduling Policies for Real-Time and Non-Real-Time Traffic in a Statistical Multiplexer.* In Proceedings IEEE INFOCOM, 1989. (p 145)

[Chong94] Song Chong, San-qi Li, and Joydeep Ghosh. *Dynamic Bandwidth Allocation for Efficient Transport of Real-Time VBR Video over ATM.* In Proceedings IEEE INFOCOM, pages 81–90, Toronto, Ontario, Canada, June 1994. (p 163)

[Chuang95] Shaw Chuang. *Security and Control of ATM Networks.* PhD thesis, Cambridge University Computer Laboratory, 1995. (in Preparation). (p 58)

[Cidon93] Israel Cidon, Inder Gopal, and Adrian Segall. *Connection Establishment in High Speed Networks.* IEEE/ACM Trans. on Networking, 1(4):469 – 481, August 1993. (p 73)

[Clark92] D. Clark, S. Shenker, and L. Zhang. *Supporting Real Time Applications in an Integrated Services Packet Network: Architecture and Mechanism.* In Proceedings ACM SIGCOMM, volume 22, Baltimore, 1992. ACM. (p 145)

[Comer91] Douglas Comer. *Internetworking with TCP/IP*, volume 1; Principles, Protocols and Architecture. Prentice-Hall International, Englewood Cliffs, New Jersey, 1991. (pp 4, 49)

[Courcoubetis91] C. Courcoubetis, G. Kesidis, A. Ridder, J. Walrand, and R. Weber. *Admission Control and Routing in ATM Networks using Inferences from Measured Buffer Occupancy.* Technical Report, University of California, Berkley, EECS Department, UCB, California, CA94720, 1991. (p 10)

[Crosby93] Simon Crosby. *In-Call Renegotiation of Traffic Parameters.* In Proceedings IEEE INFOCOM, San Francisco, March 1993. (pp i, 10, 57)

[Crosby94] Simon Crosby, Ian Leslie, Kobus vd Merwe, Andy Atkinson, Rhodri Griffiths, and Peter Key. *CDV in ATM networks – Performance Results from the Fairisle ATM Testbed.* In Proceedings RACE EXPLOIT Traffic Workshop, Basel, Switzerland, September 1994. (pp i, 94)

[Crosby95] Simon Crosby, Ian Leslie, John Lewis, Neil O'Connell, Raymond Russell, and Fergal Toomey. *Bypassing Modelling: an Investigation of Entropy as a Traffic Descriptor in the Fairisle ATM Network.* In Proceedings Twelfth UK Teletraffic Symposium, Windsor, England, March 1995. IEE. (pp 10, 12, 46, 163, 174)

[Cuéllar94a] J. R. Cuéllar, I. Wildgruber, and D. Barnard. *Combining the Design of Industrial Systems with Effective Verification Techniques.* In M. Naftalin, T. Denvir, and M. Betran, editors, *Proceedings FME'94*, volume 873 of *LNCS*, pages 639–658, Barcelona, Spain, October 1994. Springer-Verlag. (p 62)

[Cuéllar94b] Jorge Cuéllar, Dieter Barnard, and Simon Crosby. *A Tutorial Introduction to TLT. Part III: Case Study - an ATM signalling protocol.* Technical Report, Siemens AG, Corporate Research and Development, ZFE T SE 1, D-81730 Munich, Germany, 1994. Internal report. (pp 64, 189)

[de Cleene94] Brian de Cleene, Pramod Pancha, Magda El Zarki, and Henrik Sorensen. *Comparison of Priority Partition Methods for VBR MPEG.* In Proceedings IEEE INFOCOM, pages 689–696, Toronto, Ontario, Canada, June 1994. (p 10)

[Demers89] Alan Demers, Srinivasan Keshav, and Scott Shenker. *Analysis and Simulation of a Fair Queueing Algorithm.* Proceedings ACM SIGCOMM, pages 3–12, 1989. (p 144)

[Devault88] Michel Devault, Jean-Yves Cochennec, and Michel Servel. *The Prelude ATD experiment: assessments and future prospects.* IEEE JSAC, 6(9):1528–1537, December 1988. (p 3)

[Dixon92] M. J. Dixon. *System Support for Multi-Service Traffic.* Technical Report 245, Cambridge University Computer Laboratory, January 1992. Ph.D. dissertation. (pp 21, 32, 82)

[Doar93] Matthew Doar. *Multicast in the Asynchronous Transfer Mode Environment.* Technical Report 298, Cambridge University Computer Laboratory, January 1993. Ph.D. dissertation. (pp 18, 57)

[Doshi90] B.T. Doshi and S. Dravida. *Congestion Control in High Speed Wide Area Packet Networks: In-call Parameter Negotiations.* In Broadband Technologies. Proceedings Seventh ITC Specialist Seminar, October 1990. (p 57)

[Duffield95] N. G. Duffield, J. T. Lewis, N. O'Connell, R. Russell, and F. Toomey. *Entropy of ATM Traffic Streams.* To appear in Special Issue of IEEE JSAC on Advances in the Fundamentals of Networking; available as ftp://stp01.stp.dias.ie/DAPG/dapg9430.ps, 1995. (p 12)

[Ferrari89] Domenico Ferrari and Dinesh Verma. *A Scheme for Real-Time Channel Establishment in Wide-Area Networks.* IEEE JSAC, 8(3):368–379, April 1989. (p 144)

[FORE95] FORE Systems Inc., 174 Thorn Hill Rd, Warrendale PA, USA. *SPANS UNI: Simple Protocol for ATM Network Signaling*, 1995. Release 3.0. (p 73)

[Fraser93] Sandy Fraser. *Early Experiments with Asynchronous Time Division Networks.* IEEE Network, 7(1):12–26, January 1993. (p 3)

[Frater94] M. Frater, J.F. Arnold, and P. Tan. *A New Statistical Model for Traffic Generated by VBR Coders for Television in the Broadband ISDN.* Proceedings IEEE CSVT, December 1994. (pp 11, 165)

[French92] L. French, D. Greaves, and D. McAuley. *Private ATM Networks and Protocol and Interface for ATM LANs.* Technical Report 258, Computer Laboratory, May 1992. (p 84)

[Garrett94] M. W. Garrett and W. Willinger. *Analysis, Modeling and Generation of Self-Similar VBR Video Traffic.* In Proceedings ACM SIGCOMM 94, pages 269–280, London, UK, August 1994. (pp 9, 163)

[Gibbens95] R.J. Gibbens, F.P. Kelly, and P.B. Key. *A Decision Theoretic Approach to Call Admission Control in ATM Networks.* IEEE JSAC, 1995. To appear in the special issue on Advances in the Fundamentals of Networking. (pp 10, 46)

[Golestani90] S.J. Golestani. *Congestion-Free Communication in Broadband Packet Networks.* In Proceedings ICC, 1990. (p 145)

[Gravey90] A. Gravey and G. Hebuterne. *Analysis of a Priority Queue with Delay and/or Loss Sensitive Customers.* In Broadband Technologies, New Jersey, October 1990. Proceedings Seventh ITC Specialist Seminar. (p 162)

[Gravey94] Annie Gravey and Soren Blaabjerg, editors. *Cell Delay Variation in ATM Networks.* COST 242 Mid-term Seminar Interim Report, August 1994. (pp 95, 100)

[Greaves88] D.J. Greaves and A. Hopper. *The Cambridge Backbone Network.* In European Fibre Optic Conference (EFOC / LAN 88), Amsterdam, June 1988. (p 14)

[Hayter93] Mark Hayter. *A Workstation Architecture to Support Multimedia.* Technical Report 319, Cambridge University Computer Laboratory, October 1993. Ph.D. dissertation. (pp 18, 112)

[Hébuterne89] G. Hébuterne and A. Gravey. *A Space Priority Queueing Mechanism for Multiplexing ATM channels.* In ITC Specialist Seminar, Adelaide, Australia, September 1989. (p 162)

[Heijenk94] Geert J. Heijenk, Xinli Hou, and Ignas G Niemegeers. *Communication Systems Supporting Multimedia Multi-user Applications.* IEEE Network Magazine, January 1994. (pp 8, 58)

[Hluchyj88] M.G. Hluchyj and M.J. Karol. *Queueing in High Performance Packet Switching.* IEEE JSAC, 6(9), December 1988. (pp 141, 142)

[Holtzmann91] Gerard J. Holtzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International, London, 1991. (pp 53, 61)

[Hopper78] Andy Hopper. *Local Area Computer Communication Networks.* Technical Report 7, Cambridge University Computer Laboratory, 1978. (p 3)

[Hopper86] A. Hopper and R.M. Needham. *The Cambridge Fast Ring Networking System.* Technical Report 90, Unversity of Cambridge Computer Laboratory, June 1986. (p 14)

[Hui88] Joseph Y. Hui. *Resource Allocation for Broadband Networks.* IEEE JSAC, 6(9):1598–1608, December 1988. (p 10)

[Hyden94] Eoin A. Hyden. *Operating System Support for Quality of Service.* Technical Report, Cambridge University Computer Laboratory, February 1994. (p 23)

[IEEE92] IEEE Computer Society Draft Standard. *POSIX Extension for Portable Operating Systems P1003.4a / D6*, 1992. (pp 22, 80)

[ISO91] *ISO Standard, IS 9595, Common Management Information Service Definition*, February 1991. (p 33)

[ITU-T88] ITU-T. *Blue Book, Recommendation Z.100: Functional SDL. Criteria for Using Formal Description Techniques.* ITU, Geneva, 1988. (p 61)

[ITU-T92a] ITU-T. *Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model.* ITU, Boulder, Colorado, November 1992. (p 68)

[ITU-T92b] ITU-T. *Recommendation M.3010, Principals for a Telecommunications Management Network.* ITU, 1992. (p 66)

[ITU-T92c] ITU-T. *Recommendation M.3100, Generic Network Information Model.* ITU, 1992. (p 66)

[ITU-T93a] ITU-T. *Draft recommendation Q.650, B-ISDN Interworking between SS No. 7 B-ISUP and DSS 2.* ITU, Geneva, Switzerland, December 1993. (p 6)

[ITU-T93b] ITU-T. *Recommendation I.371 – Traffic Control and Congestion Control in B-ISDN.* ITU, Helsinki, March 1993. (pp 11, 95)

[ITU-T93c] ITU-T. *Recommendation Q.1400, Architecture Framework for the Development of Signalling and OA&M Protocols Using OSI Concepts.* ITU, Helsinki, Finland, March 1993. (p 6)

[ITU-T94a] ITU-T. *Draft Recommendation Q.2931, Broadband Integrated Services Digital Network (B-ISDN) Digital Subscriber Signalling System No. 2, User-Network Interface Layer 3 Specification for Basic Call/Connection Control.* ITU, Geneva, Switzerland, November 1994. (pp 6, 27, 65)

[ITU-T94b] ITU-T. *Recommendation Q.931, Digital Subscriber Signalling System No. 1, ISDN User-Network Interface Layer 3 Specification for Basic Call Control.* ITU, Geneva, Switzerland, November 1994. (p 65)

[Jabbari91] Bijan Jabbari. *Common Channel Signalling System Number 7 for ISDN and Intelligent Networks.* Proceedings IEEE, 79(2):155–169, 1991. (p 5)

[Jackson60] J.R. Jackson. *Some Problems in Queueing with Dynamic Priorities.* Naval Res. Logist. Quarterly, 7:235–249, 1960. (p 144)

[Kalmanek90] Charles Kalmanek, Hemant Kanakia, and Srinivasan Keshav. *Rate Controlled Service for Very High Speed Networks.* In Proceedings IEEE GLOBECOM, San Diego, California, December 1990. (p 146)

[Kelly91] F.P. Kelly. *Effective Bandwidths at Multi-Class Queues.* Queueing Systems, 9:5–16, 1991. (pp 10, 46)

[Kelly94] F.P. Kelly and P.B. Key. *Dimensioning Playout Buffers from an ATM Network.* In Proceedings UK Teletraffic Congress. IEE, 1994. (pp 95, 102)

[Kröner91] H. Kröner, G. Hébuterne, P. Boyer, and A. Gravey. *Priority Management in ATM Queueing Systems.* IEEE JSAC, 9(3):418–427, April 1991. (p 162)

[Kung94] H. T. Kung, T. Blackwell, and A. Chapman. *Credit Update Protocol for Flow-Controlled ATM Networks: Statistical Multiplexing and Adaptive Credit Allocation.* In Proceedings ACM SIGCOMM 94, pages 101–114, London, UK, August 1994. (p 10)

[Laubach94] Mark Laubach. *RFC 1577, Classical IP and ARP over ATM*, January 1994. (p 78)

[Lazar94] A.A. Lazar, S. Bhonsle, and K.S. Lim. *A Binding Architecture for Multimedia Networks.* In Proceedings Multimedia Transport and Teleservices, Vienna, Austria, November 1994. (p 71)

[Leffler89] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.3 BSD UNIX Operating System.* Addison-Wesley Publishing Company, Reading, Massachusetts, November 1989. (pp 20, 32, 38)

[Leland93] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. *On the Self-Similar Nature of Ethernet Traffic.* In Proceedings ACM SIGCOMM, 1993. (pp 9, 106)

[Leslie83] I. M. Leslie. *Extending the Local Area Network.* PhD thesis, University of Cambridge Computer Laboratory, Pembroke Street, Cambridge, CB2 3QG, February 1983. (p 18)

[Leslie91] I.M. Leslie and D. McAuley. *Fairisle: An ATM Network for the Local Area.* In Proceedings ACM SIGCOMM, Zürich, September 1991. (p 14)

[Leslie93] I. M. Leslie, D. R. McAuley, and S. J. Mullender. *Pegasus — Operating System Support for Distributed Multimedia Systems.* Operating Systems Review, 27(1):69–78, January 1993. Also available as Pegasus Paper 92–2. (p 23)

[Li95] Guangxing Li. *ANSAware/RT Version 1.0: Programming and System Overview.* Draft request for comments APM.1207.00.01, Architecture Projects Management Limited, March 1995. (p 73)

[Lim88] Y. Lim and J. Kobza. *Analysis of a Delay Dependent Priority Discipline in a Multi-Class Traffic Packet Switching Node.* In Proceedings IEEE GLOBECOM, 1988. (p 144)

[Louis94] Stéphane Louis. *Cell Delay Variation in ATM Networks: Causes and Magnitudes.* In Proceedings RACE EXPLOIT Traffic Workshop, Basel, Switzerland, September 1994. (p 96)

[Lucantoni90] D.M. Lucantoni and S.P. Parekh. *Selective Cell Discard Mechanisms for a BISDN Congestion Control Architecture.* In Broadband Technologies. Proceedings Seventh ITC Specialist Seminar, October 1990. (pp 10, 162)

[Malis92] A. Malis, D. Robinson, and R. Ullman. *RFC 1356, Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode*, August 1992. (p 80)

[Matragi94] Wassim Matragi, Khosrow Sohraby, and Chatschik Bisdikian. *Jitter Calculus in ATM Networks: Multiple Node Case.* In Proceedings IEEE INFOCOM, pages 242–251, Toronto, Ontario, Canada, June 1994. (pp 95, 102, 121, 124)

[McAuley90] Derek McAuley. *Protocol Design for High Speed Networks.* PhD thesis, University of Cambridge Computer Laboratory, Pembroke Street, Cambridge, CB2 3QG, January 1990. (pp 18, 29, 32)

[Miller92a] Patrick A Miller and Petre N Turcu. *Generic Signalling Protocol: Architecture, Model and Services.* IEEE Trans. Commun., 40(5):957–966, May 1992. (p 66)

[Miller92b] Patrick A Miller and Petre N Turcu. *Generic Signalling Protocol: Switching, Networking and Interworking.* IEEE Trans. Commun., 40(5):967–979, May 1992. (pp 65, 66)

[Minzer91] Steven Minzer. *A Signaling Protocol for Complex Multimedia Services.* IEEE JSAC, 9(9):1383–1394, December 1991. (pp 8, 58, 71)

[Molnár94] Sándor Molnár and Soren Blaabjerg. *The Effect of Multiplexing CDV Affected CBR Cell Streams.* Technical Report, COST 242 TD(94), May 1994. (p 95)

[Moore94] Simon Moore. *Multithreaded Processor Design.* Technical Report 358, Cambridge University Computer Laboratory, October 1994. Ph.D. dissertation. (p 144)

[Nellas94] V. Nellas, N. Lykouropoulos, G. Konstantoulakis, and N. Mitrou. *Experiments with ON/OFF PC-produced Traffic with Arbitrary, Programmable ON and OFF Sojourn Time Distributions.* In Proceedings RACE EXPLOIT Traffic Workshop, Basel, Switzerland, September 1994. (p 96)

[Newman89] Peter Newman. *Fast Packet Switching for Integrated Services.* PhD thesis, Unversity of Cambridge Computer Laboratory, Pembroke Street, Cambridge, CB2 3QG, March 1989. (p 14)

[Nicolaou91] Cosmos Nicolaou. *A Distributed Architecture for Multimedia Communication Systems.* Technical Report 220, Cambridge University Computer Laboratory, December 1991. Ph.D. dissertation. (pp 7, 28, 50, 72)

[Nilsson95] Gunnar Nilsson, Fabrice Dupuy, and Martin Chapman. *An Overview of the Telecommunications Information Networking Architecture.* In Proceedings TINA Conference, Melbourne, Australia, February 1995. (p 68)

[Oie89] Y. Oie, M. Murata, K. Kubota, and H. Miyahara. *Effect of Speedup in Nonblocking Packet Switch.* In Proceedings ICC, 1989. (p 142)

[OMG91] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, Draft 10th December 1991. Document Number 91.12.1, revision 1.1. (pp 61, 68, 71)

[OSF92] Open Software Foundation. *OSF DCE 1.0 Application Development Guide*, 1992. Volume 2. (pp 61, 68)

[Ozveren94] C. Ozveren, R. Simcoe, and G. Varghese. *Reliable and Efficient Hop-by-Hop Flow Control.* In Proceedings ACM SIGCOMM 94, pages 89–100, London, UK, August 1994. (p 10)

[Pacifici95] G. Pacifici and R. Stadler. *An Architecture for Performance Management in Multimedia Networks.* In Proceedings IEEE INFOCOM, Boston, April 1995. (p 45)

[Panwar88] S.S. Panwar, D. Towsley, and J.K. Wolf. *Optimal Scheduling Policies for a Class of Queues with Customer Deadlines to the Start of Service.* Jnl. ACM, 35(4):832–844, October 1988. (p 144)

[Perez95] M. Perez, F. A. Mankin, E. Hoffman, G. Grossman, and A. Malis. *RFC 1755, ATM Signaling Support for IP over ATM*, February 1995. (p 78)

[Petr90] David Petr and Victor Frost. *Optimal Threshold-Based Discarding for Queue Overload Control.* In Broadband Technologies, New Jersey, October 1990. Proceedings Seventh ITC Specialist Seminar. (p 162)

[Rathgeb90] Erwin P. Rathgeb. *Policing Mechanisms for ATM Networks – Modelling and Performance Comparison.* In Broadband Technologies. Proceedings Seventh ITC Specialist Seminar, 1990. (p 11)

[RFC90] Internet Network Working Group. *RFC 1190, Experimental Internet Stream Protocol, ST-II*, 1990. Version 2. (p 4)

[Roberts92] J. W. Roberts, editor. *Performance evaluation and design of multiservice networks – COST 224 Final Report.* Commission of the European Communities, Directorate-General, Telecommunications, Information Industries and Innovation, L-2920 Luxembourg, 1992. (pp 11, 57, 95, 125, 126, 162, 171, 211)

[Roosma94] André Roosma and W. Marieke ten Wolde. *A New Definition of Cell Delay Variation.* Technical Report, Royal PTT Netherland submission to ITU-T Study Group 13, March 1994. (p 100)

[Roscoe94] Timothy Roscoe, Simon Crosby, and Richard Hayton. *The MSRPC2 User Manual* . In SRG [SRG94], chapter 16. SRG Technical Note available as: http://www.cl.cam.ac.uk/Research/SRG/bluebook.html. (p 21)

[Roscoe95] Timothy Roscoe. *The Structure of a Multi-Service Operating System.* PhD thesis, Cambridge University Computer Laboratory, April 1995. (pp 23, 73)

[Rose91] Marshall T. Rose. *The Simple Book.* Prentice-Hall International, Englewood Cliffs, New Jersey, 1991. (pp 7, 66)

[Saito90] Hiroshi Saito. *Optimal Queueing Discipline for Real Time Traffic at ATM Switching Nodes.* IEEE Trans. Commun., 38(12):2131–2136, December 1990. (p 144)

[Schroeder90] Michael D. Schroeder, Andrew D. Burrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. *Autonet: A High-speed, Self-configuring Local Area Network Using Point to Point Links.* Technical Report 59, Digital Systems Research Centre, 130 Lytton Ave, Palo Alto, California, April 1990. (p 48)

[Schulzrinne94] Henning Schulzrinne, Jim Kurose, and Don Towsley. *An Evaluation of Scheduling Mechanisms for Providing Best-Effort, Real Time Communication in Wide Area Networks.* In Proceedings IEEE INFOCOM, Toronto, Ontario, Canada, June 1994. (p 145)

[Sharma94] R. Sharma and S. Keshav. *Signaling and Operating System Support for Native-Mode ATM Applications.* In Proceedings ACM SIGCOMM 94, pages 149–157, London, UK, August 1994. (p 84)

[Sreenan92] Cormac J. Sreenan. *Synchronisation Services for Digital Continuous Media.* Technical Report 292, Cambridge University Computer Laboratory, October 1992. (p 7)

[SRG94] *ATM Document Collection.* Systems Research Group, University of Cambridge Computer Laboratory, 3rd (The Blue Book) edition, March 1994. SRG Technical Note available as: http://www.cl.cam.ac.uk/Research/SRG/bluebook.html. (pp 14, 32, 92, 105, 186)

[Sriram86] Kotikalapudi Sriram and Ward Whitt. *Characterizing Superposition Arrival Processes for Packet Multiplexers for Voice and Data.* IEEE JSAC, 4(6):833–846, September 1986. (pp 123, 211)

[Stallings89] William Stallings. *Data and Computer Communications.* Maxwell MacMillan International, Singapore, 1989. (pp 4, 5, 58)

[Stiller95] Burkhard Stiller. *A Survey on UNI Signalling Systems and Protocols for ATM Networks.* Cambridge University Computer Laboratory, January 1995. (p 65)

[Sun87] Sun Microsystems. *XDR: External Data Representation Standard*, June 1987. (pp 22, 73)

[Sun90] Sun Microsystems. *SunNet X.25 System Administration Manual*, October 1990. Version 7.0 beta. (p 80)

[Tennenhouse89] D. L. Tennenhouse. *Layered Multiplexing Considered Harmful.* In Rudin and Williamson, editors, *Protocols for High Speed Networks.* Elsevier, 1989. (p 7)

[Tutufor90] Kos Tutufor. *On Admission Control and Policing in an ATM Based Network.* In Broadband Technologies, New Jersey, October 1990. Proceedings Seventh ITC Specialist Seminar. (p 11)

[van den Berg92] J. L. van den Berg and J. A. C. Resing. *The Change of Traffic Characteristics in ATM Networks.* Technical Report, COST 242 TD(92)040, 1992. (pp 95, 101, 125, 133)

[Verma91] Dinesh Verma, Hui Zhang, and Domenico Ferrari. *Guaranteeing Delay Jitter Bounds in Packet Switching Networks.* In Proceedings Tricomm 91, Chapel Hill, North Carolina, April 1991. (p 146)

[Yin90] N. Yin and M. G. Hluchyj. *Implication of Dropping Packets from the Front of a Queue.* In Broadband Technologies. Proceedings Seventh ITC Specialist Seminar, October 1990. (p 162)

[Yuan89] Chin Yuan and John A. Silvester. *Queueing Analysis of Delay Constrained Voice Traffic in a Packet Switching System.* IEEE JSAC, 7(5):729–738, June 1989. (p 162)

[Zhang90] L. Zhang. *Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks.* In Proceedings ACM SIGCOMM, volume 20, Philadelphia, 1990. ACM. (p 144)

[Zhang91] Hui Zhang and Srinivasan Keshav. *Comparison of Rate-Based Service Disciplines.* Proceedings ACM SIGCOMM, pages 113 – 121, 1991. (pp 146, 150)

[Zhang93a] Hui Zhang and Domenico Ferrari. *Rate-Controlled Static Priority Queueing.* In Proceedings IEEE INFOCOM, pages 227–234, 1993. (p 146)

[Zhang93b] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. *RSVP: A New Resource ReSerVation Protocol.* IEEE Network Magazine, September 1993. (p 4)

Appendix A

Verification of the User Master Interface

By: Dieter Barnard[1]

This appendix describes the procedures involved in verifying the TLT design of the UMI protocol presented in Chapter 4.

A.1 TLT Specification of the UMI

The design consists of two U-TLT *programs*, the UUMI and MUMI, representing a user and the MSNL Master respectively. In addition to a default *interface*[2] to the *environment* surrounding it, each program also has three dedicated interfaces: the UUMI has interfaces to the PSI, EXC and UMI, while the MUMI has interfaces to the UMI, EXC (exception interface) and MMI. Each interface contains input and output buffers which model RPC operations with asynchronous message passing. The UUMI and MUMI programs are composed to model the whole protocol using a *connector*. The following subsections discuss the design of the MUMI in detail. The design of the UUMI is similar and can be found in [Cuéllar94b].

A.1.1 The MUMI TLT Program

The U-TLT specification of the MUMI is given in Figure A.1. The program is structured into *header*, *declaration*, *interface*, *link-constraints*, *initially* and *instruction* sections. The program header contains a *rigid* variable *time_out_max*,

[1]Siemens AG, ZFE T SE 1, D-81730 Munich, Germany. Tel: +49 89 636-49601. Fax: +49 89 636-42284. E-Mail: Dieter.Barnard@zfe.siemens.de

[2]Unfortunately the word *interface* has fixed but different meanings in the MSRPC2 IDL and TLT. Here a U-TLT interface represents a particular view of the environment from the perspective of a program.

PROGRAM *MUMI_Protocol* (RIGID *time_out_max*: Natural)

DECLARATIONS
VAR
  LOCAL *call*: { *state_u_final, state_u_idle, state_u_sap_bound, state_u_sap_offered, state_n_connect, state_u_accepted, state_n_complete, state_u_connect, state_n_accepted, state_u_disconnect, state_n_disconnect, state_u_fin_pend, state_u_aborting, state_n_aborting, state_aborting* };
  LOCAL *sap*: { *null, bound, active, passive* };
  LOCAL *time_out_ctr*: Range(*0, time_out_max*);
  LOCAL *umi_link, mmi_link*: { *online, offline* };
  GLOBAL *sap_manager, qos_manager*: { *ok, not_ok* }

INTERFACES
  INCLUDE *MUMI_Interface*;
  INCLUDE *MMI_Interface*;
  INCLUDE *EXC_Interface*

LINK-CONSTRAINTS
  $[rx\_umi]\ \neg(rx\_umi' = u\_null) \Rightarrow (rx\_mmi' = n\_null \wedge rx\_exc' = e\_null)$
  $[rx\_mmi]\ \neg(rx\_mmi' = n\_null) \Rightarrow (rx\_umi' = u\_null \wedge rx\_exc' = e\_null)$
  $[rx\_exc]\ \neg(rx\_exc' = e\_null) \Rightarrow (rx\_umi' = u\_null \wedge rx\_mmi' = n\_null)$

INITIALLY
  $call = state\_u\_final \wedge sap = null \wedge time\_out\_ctr = 0$
  $\wedge\ umi\_link = online \wedge mmi\_link = online$
  $\wedge\ qos\_manager = ok \wedge sap\_manager = ok$

INSTRUCTIONS

```
% ----------------------------------------------------------------------
% Call state = STATE_U_FINAL and UMI input = U_SAP_REQUEST
```
  [] $call = state\_u\_final \wedge rx\_umi = u\_sap\_req \longrightarrow$
    (IF ($sap\_manager = ok$) THEN
      ($call' = state\_u\_sap\_bound \parallel rx\_umi' = u\_null \parallel tx\_umi' = u\_sap\_bound$
      $\parallel sap' = bound \parallel tx\_exc' = e\_set\_timer \parallel time\_out\_ctr' = 0$)
    ELSE
      ($call' = state\_u\_final \parallel rx\_umi' = u\_null \parallel tx\_umi' = u\_sap\_denied$))

```
% All other MUMI instructions modelled in a similar way.
```
END.

Figure A.1: The MUMI Program

which specifies the number of time-out exceptions before taking alternative action. Rigid variables act like constants and are instantiated before execution. The declaration section defines the program variables, consisting of the call-specific state variables *call* and *sap*, link status variables *umi_link* and *mmi_link*, and manager status variables *sap_manager* and *qos_manager*. The call-specific and link status variables are *local* (i.e. not visible to the environment), whereas the managers are *global* (i.e. visible and modifiable by the environment).

The MUMI further contains three *dedicated* interfaces to the UMI, EXC and MMI (explained in the next section), which define additional variables (eg. the communication buffers). The link-constraints section allows a designer to make explicit assumptions about the environment in which a program will be executed (this is in addition to the implicit link-constraints imposed by the environment classes of the variables). Here it is used as a simple scheduler, restricting[3] the input buffers of the three interfaces to contain only one message at a time. A logical predicate in the *initially* section represents a predetermined set of initial states from which execution starts. The initial values of the MUMI program variables are straightforward.

The main part of the MUMI specification consists of ca. 160 instructions glued together by the [] operator. The full protocol specification is presented in Appendix B. U-TLT instructions are written as guarded-commands ($\longrightarrow$), using IF-THEN-ELSE for internal decisions and $\|$ for parallel execution within each command. The atomic expressions of each instruction are logical predicates, where the unprimed program variables (*call, sap*, etc.) refer to their current value and the primed variables (*call', sap'*, etc.) to the values in the next execution state. The example in Figure A.1. should now be easy to follow: if the call is state *state_u_final* (i.e. no state) and a *u_sap_req* is received via the UMI, then if the SAP-manager indicates *ok*, the SAP is granted, otherwise it is refused.

The combined behaviour of a U-TLT program and its environment is known as its *closure*, and can be defined intuitively as follows: if any of the program instructions can be executed (i.e. their guards evaluate to true in a given execution state), then one is selected non-deterministically and executed. Otherwise, if no program instruction can be executed, a special environment instruction satisfying all implicit and explicit link-constraints is executed. This execution loop is repeated ad infinitum. This type of semantics will be called *reactive*, since the program continuously processes inputs from the environment, based on the underlying assumption that it is fast enough to do so.

[3]This restriction does not influence the correctness of the design, but collapses its state space markedly. A more detailed level of design could include a more complex scheduling mechanism.

A.1.2 The MUMI Exception Interface

The U-TLT dedicated interfaces consist of an *execution* section, a *declaration* section, an *always* section, an *initially* section and a *link-constraint* section. Only the *execution* and *always* sections are new: the *execution* section specifies the way the program and the environment will interact via the given interface, and the *always* section contains logical predicates which holds throughout the execution of the program. As an example of a dedicated interface, the MUMI exception interface is listed in Figure A.2. It is the biggest of the three MUMI interfaces, since the exception handler is a trusted application responsible for timers, link status, etc. This knowledge about the exception handler is included as assumptions[4] in the *link-constraint* section of the interface and subsequently exploited for verification purposes.

A.1.2.1 RPC, Exceptions and Message Passing

The three interfaces of the MUMI each contains an input and an output buffer to model RPC communication. In the example Figure A.2 the exception interface contains two *global* variables, *rx_exc* and *tx_exc*, as asynchronous buffers of unit-size. Raised exceptions are modelled as input messages, and timer actions by the protocol as output messages. For the other two interfaces of the MUMI, the request and reply nature of RPC is implemented as message-passing by the program instructions themselves. The protocol specification includes an instruction for every call state and input message combination, starting with messages expected for particular call state, then unexpected but harmless messages which can be ignored, and finally, unexpected messages indicating an inconsistent call state at the other side.

A.1.2.2 Timers

Both the UUMI and MUMI make extensive use of timers to react to communication failures. To model timers in U-TLT, a *history* variable *time_out_history* and an *always* section which records the status of the (external) timer have been added to the exception interface. History variables are used to model the behaviour of an environment without complicating the program (*time_out_history* only appears in this interface). *Time_out_history* is needed in the *fairtime* link-constraint, where the environment is required to be *weak fair* (WF): if the timer is set (*e_set_timer*) and stays set indefinitely, then a time-out must occur as part of any infinite sequence of messages. The combination of a history variable and this

[4]The assumptions made in link-constraints about the environment of a program should be justified in terms of the real world; alternatively, if the environment is another U-TLT program, they become proof obligations.

```
INTERFACE EXC_Interface
EXECUTION Reactive
DECLARATIONS
VAR
    GLOBAL rx_exc: { e_time_out, e_idle, e_u_data, e_u_failed, e_n_failed,
        e_u_recovered, e_n_recovered, e_null };
    GLOBAL tx_exc: { e_set_timer, e_reset_timer, e_cancel_timer, e_null }
HISTORY
    LOCAL    time_out_history: { on, off }
ALWAYS
    (tx_exc' = e_set_timer) ⇒ (time_out_history' = on)
    (tx_exc' = e_cancel_timer) ⇒ (time_out_history' = off)
INITIALLY
    (rx_exp = e_null) ∧ (tx_exp = e_null) ∧ (time_out_history = off)
LINK-CONSTRAINTS
    [tx_exc]      ¬(tx_exc' = tx_exc) ⇒ (tx_exc' = e_null)
    [time]        (time_out_his = off) ⇒ ¬ (rx_exc' = e_time_out)
    [umi1link]    (umi_link = online) ⇒ ¬ (rx_exc' = e_u_recovered)
    [umilink2]    (umi_link = offline) ⇒ ¬ (rx_exc' = e_u_failed)
    [mmilink1]    (mmi_link = online) ⇒ ¬ (rx_exc' = e_n_recovered)
    [mmilink2]    (mmi_link = offline) ⇒ ¬ (rx_exc' = e_n_failed)
    [fairtime]    WF(time_out_history = on, rx_exc = e_time_out)
    [fairfail1]   WF(¬(call = state_u_aborting ∧ rx_exc = e_u_failed))
    [fairfail2]   WF(¬(call = state_n_aborting ∧ rx_exc = e_n_failed))
END.
```

Figure A.2: The MUMI Exception Interface

fairness assumption captures the essence of time-outs in a simple but effective way.

A.1.2.3 Cyclic Behaviour

Certain RPC requests, eg. withdrawing an offered SAP, and certain exceptions, eg. the failure and recovery of a link, are of a cyclic nature. Such sequences can (correctly) happen an infinite number of times, but from a modelling point of view they are difficult to distinguish from other *non-progress cycles*, which are real errors in the protocol. Rather than modifying the MUMI for this purpose (eg. allowing a link to fail once only), appropriate link constraints were added to prohibit such infinite behaviour. The MUMI exception interface contains two examples, *fairfail1* and *fairfail2*, which restrict link failure messages using fairness: formulated negatively, the *exc_u_failed* message (link failed) may not be received infinitely often when the call is in state *state_u_aborting* (similarly for *exc_n_failed* and *state_n_aborting*). In total 7 such fairness link-constraints were added to the MUMI interfaces.

A.1.2.4 Call Acceptance Control

The modelling of CAC was achieved quite simply. The environment class of the *sap_manager* and *qos_manager* variables is global. Initially their values are *ok*, but since no other instruction or link-constraint make reference to them as primed variables, they can change their values arbitrarily and independently from *ok* to *not_ok* and back again. While this would of course not be the case for the final implementation of the protocol, it suffices to specify and verify the protocols at this level of abstraction: all possible behaviours are modelled this way, and all the instructions of the MUMI program are deterministic in terms of call state and input message.

A.2 Verifying the UMI Design

The strength of TLT lies in the smooth integration of design and verification within a single framework, and not just providing verification as an ad-hoc feature. Verification techniques for TLT can be divided into two main groups, based on whether the underlying datatypes used for modelling are finite (as is the case for the UMI), or not. The reason for this is that finite domains can always be encoded as Boolean domains, which can be implemented very efficiently as Binary Decision Diagrams (BDDs). The state space of a TLT program is thus represented symbolically rather than explicitly, with the advantage that very large state spaces can be verified this way.

A.2.1 The TLT Toolbox

The TLT toolbox for programs using finite datatypes is displayed in Figure A.3. The *Consistency Checker* inputs a given U-TLT program and checks a number of consistency criteria for a given program, eg. that it is indeed possible to execute each U-TLT instruction. These criteria ensure that the A-TLT formula corresponding to the program does indeed have a semantic interpretation. The *Simulator* allows a user to step through a TLT program and examine its behaviour. Finally, the *Model Checker* inputs a U-TLT program and an A-TLT formula representing a property and automatically determines whether the program does indeed satisfy this requirement. If not, the *Debugger* is capable of generating counterexamples demonstrating why this is not the case. All three tools were used extensively throughout this case study.

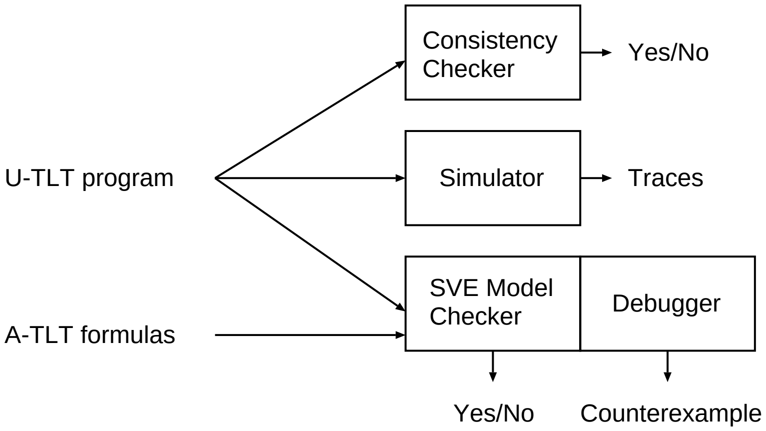


Figure A.3: The TLT Toolbox

Although the model checker does not require the *a priori* calculation of the reachable state space of a U-TLT program, the model checker tool provides this feature optionally (and subsequently uses the result to optimise further calculations). The reachability statistics do however provide insight into the size of the verification task. The first step is translate the *initially* and *instruction* sections of a program closure to an appropriate BDD representation. The set of reachable states can then be computed symbolically by forward iteration starting from the set of initial states. The statistics for this case study are listed in Figure A.4.

Model	#BDD Nodes	#Reachable States	Fairness Time (s)	Reach. Time (s)
UUMI (without fairness)	906	5429	n.a.	4.13
UUMI (with fairness)	595	3172419	7.48	25.93
MUMI (without fairness)	492	9109	n.a.	5.88
MUMI (with fairness)	640	145644723	16.58	45.38

Figure A.4: Reachability Statistics

Four models are listed, two for the UUMI closure and two for the MUMI closure, depending on whether the fairness assumptions in the corresponding link-constraints were included. Without fairness assumptions, the reachable state space of both programs consist of less than 10^4 states. This was computed in less than 6 seconds on a SUN SPARC2. The 7 fairness assumptions alter this picture radically, because the state space needs to be extended to keep track of eventualities which influence infinite behaviour. The reachable state space of the MUMI program now contains 145 million states. Nevertheless, the combined fairness and reachability calculation still took only 61 seconds due to the succinct BDD size[5].

A.2.2 Verification of the MUMI

The consistency checker only provides the guarantee that a U-TLT program does not contain any logical inconsistencies (i.e. contradictions). The correctness of the programs in terms of user requirements however, is the task of the model checker.

The A-TLT Language

User requirements are expressed in TLT as A-TLT formulas. The subset of A-TLT which will be used below consists of the following:

$$\phi := \phi \wedge \phi \mid \phi \vee \phi \mid \mathsf{ALWAYS}(\psi) \mid \psi \mapsto \psi$$

where ψ is a propositional predicate containing unprimed and primed program variables. Intuitively $\mathsf{ALWAYS}(\psi)$ holds, if ψ holds in every pair[6] of reachable states of the closure. It is a safety operator, i.e. expresses that "nothing bad should happen". The $\mapsto$ operator is called leads-to, and $p_1 \mapsto p_2$ holds if all fair execution paths starting from pairs of states satisfying p_1 lead to pairs of states satisfying p_2. It is a liveness operator, i.e. that "something good should happen".

MUMI Properties

Since the TLT specification of the MUMI includes an instruction for every possible call state and input message, the program cannot deadlock and there is no need to verify deadlock freedom. The A-TLT logic was however used to formulate 23 safety and liveness properties that the MUMI design should fulfill. Representative examples are listed in Figure A.5, and discussed below:

- The most important property the MUMI should satisfy is *nolivelock*, i.e. that the program does not contain any nonprogress cycles. This is expressed in A-TLT as $\mathsf{true} \mapsto (\mathrm{call} = \mathrm{state\_u\_final})$, i.e. all fair execution paths starting from any state lead to a state where the call state is null. With the help of the model checker and debugger all valid cyclic behaviour

[5] The model checker uses dynamic BDD variable reordering to keep the BDD representations optimal

[6] The semantics of A-TLT operators are not defined formally here. Note that since propositional predicates can contain both unprimed and primed variables, the semantics is defined locally over pairs (η_1, η_2) of states instead of just a single state (η). This also explains the absence of an explicit next operator.

PROPERTIES

$[\mathit{nolivelock}]$ $\mathsf{true} \mapsto (\mathit{call} = \mathit{state\_u\_final})$

$[\mathit{getsap}]$ $(\mathit{call} = \mathit{state\_u\_final}) \wedge (\mathit{rx} = \mathit{u\_sap\_req}) \wedge (\mathit{sap\_manager} = \mathit{ok}) \mapsto (\mathit{call} = \mathit{state\_u\_sap\_bound})$

$[\mathit{umifail}]$ $\mathsf{ALWAYS}((\mathit{rx\_exc} = \mathit{e\_u\_failed}) \Rightarrow (\mathit{umi\_link'} = \mathit{offline}))$

$[\mathit{sapboth}]$ $\mathsf{ALWAYS}((\mathit{call} = \mathit{state\_n\_complete}) \Rightarrow (\mathit{sap} = \mathit{active} \vee \mathit{sap} = \mathit{passive}))$

$[\mathit{noleakage}]$ $\mathsf{true} \mapsto (\mathit{sap} = \mathit{null})$

END.

Figure A.5: Examples of MUMI Properties

mentioned in Section A.1.2.3 was identified, and more important, two non-progress cycles in the abort phase were found. These two subtle errors were corrected in subsequent versions of the specification.

- The *getsap* property describes the functionality of SAP acquisition, i.e. if a call is in state *state_u_final*, receives a *u_sap_req* request and the SAP manager has sufficient resources, then the SAP is always granted. Although this trivially matches the instruction example earlier in Figure A.1, it should also remain valid if the specification is refined to allocate a SAP in more than one step. A number of similar properties were successfully checked for each protocol phase.

- The *umifail* property is a safety property that requires the MUMI to handle link failure correctly, i.e. if an *exc_u_failed* exception is raised, then the status of the link will be updated in *umi_link* accordingly. Similar properties were checked for the *exc_n_failed* exception and for link recovery. The model checker/debugger found 2 situations where such updates were neglected. The specification was corrected accordingly.

- *Sapboth* is another safety property, which is concerned with the consistency of SAP status and call state. It requires that when data transfer takes place, the user's SAP must be either passive (receiving) or active (sending), and not just bound, or even worse, null. This and similar properties for other call states uncovered 3 mistakes in the management of SAPs by the MUMI. They were fixed.

- Finally, *noleakage* is a liveness property which requires that SAPs will eventually be freed (no resource leakage), i.e. all fair execution paths starting from any state lead to a state where the SAP state is null. It is similar in spirit to the *nolivelock* property in that it would also be fouled by non-progress cycles if any are present, but the two properties do not imply each other.

Further properties of the MUMI can of course be formulated in A-TLT, but the collection above is representative of the requirements for the UMI protocol described in Section 2. The final verification results are summarised in Figure A.6.

Property	Fairness Assumptions	Result	Time(s)
nolivelock	All	true	119.43
getsap Similar for other phase properties.	None	true	2.88
umifail Similar for other link properties.	None	true	0.33
sapboth Similar for other SAP properties.	None	true	0.39
noleakage	All	true	59.79

Figure A.6: Verification Benchmarks

Verification Times

The model checker needed between 0.39 seconds and 119.43 seconds on a SUN SPARC2 to successfully check the properties listed in Figure A.5. The high variance in the benchmarks can be explained as follows: firstly, ALWAYS properties (eg. link and SAP) can be checked immediately if the reachable state space has been computed. Secondly, for *getsap* there was no need to include the include fairness assumptions, and it could thus be checked using the small reachability model. The *nolivelock* and *noleakage* properties are however more complex and in addition rely on the fairness assumptions, which is why the model checker needed 119.43 seconds and 59.79 seconds to check them respectively. This is still acceptable. In our experience the verification times for a monolithic version of the protocol (i.e. UUMI and MUMI with fairness) would be at least an order of an magnitude greater.

Appendix B

TLT Specification of the MUMI

```
PROGRAM protocol(RIGID time_out_max: nat)

TYPES
   call_type := {state_u_final, state_u_idle, state_u_sap_bound, state_u_sap_offered, state_n_connect,
      state_u_accepted, state_n_complete, state_u_connect, state_n_accepted, state_u_disconnect,
      state_n_disconnect, state_u_fin_pend, state_u_aborting, state_n_aborting, state_aborting};
   sap_status_type := { null, bound, active, passive };

   time_out_ctr_type := unsigned_range(0, time_out_max);
   time_out_his_type := { on, off };
   link_status_type  := { online, offline };
   manager_type      := { ok, not_ok };

   rx_umi_type := {u_abort, u_aborted, u_accepted, u_complete, u_connect, u_disconnect,
      u_final, u_idle, u_rejected, u_sap_bound, u_sap_denied, u_sap_offer, u_sap_offered,
      u_sap_req, u_sap_withdraw, u_null};
   rx_mmi_type := {n_accepted, n_complete, n_connect, n_disconnect, n_rejected,
      n_release, n_null};
   rx_exc_type := {e_time_out, e_idle, e_u_data, e_u_failed, e_n_failed, e_u_recovered,
      e_n_recovered, e_null};
   tx_umi_type := {u_abort, u_aborted, u_accepted, u_complete, u_connect, u_disconnect,
      u_final, u_idle, u_null, u_rejected, u_sap_bound, u_sap_denied, u_sap_offer,
      u_sap_offered, u_sap_req, u_sap_withdraw};
   tx_mmi_type := {n_accepted, n_complete, n_connect, n_disconnect, n_null, n_rejected,
      n_release};
   tx_exc_type := {e_null, e_set_timer, e_reset_timer, e_cancel_timer}

DECLARATIONS
VAR
   LOCAL   call: call_type;
   LOCAL   sap: sap_status_type;

   LOCAL   time_out_ctr: time_out_ctr_type;
   LOCAL   time_out_his: time_out_his_type;
   LOCAL   umi_link: link_status_type;
   LOCAL   mmi_link: link_status_type;
   LOCAL   sap_manager: manager_type;
   LOCAL   qos_manager: manager_type;

   GLOBAL  rx_umi: rx_umi_type;
   GLOBAL  rx_mmi: rx_mmi_type;
   GLOBAL  rx_exc: rx_exc_type;
   GLOBAL  tx_umi: tx_umi_type;
   GLOBAL  tx_mmi: tx_mmi_type;
   GLOBAL  tx_exc: tx_exc_type

LINK-CONSTRAINTS
   [time] (time_out_his = off) => ~(rx_exc' = e_time_out)
   [umi1] (umi_link = online)  => ~(rx_exc' = e_u_recovered)
   [umi2] (umi_link = offline) => ~(rx_exc' = e_u_failed)
   [umi3] (umi_link = offline) =>  (rx_umi' = u_null)
   [mmi1] (mmi_link = online)  => ~(rx_exc' = e_n_recovered)
   [mmi2] (mmi_link = offline) => ~(rx_exc' = e_n_failed)
   [mmi3] (mmi_link = offline) =>  (rx_mmi' = n_null)
   [my_rx_umi] ~(rx_umi' = u_null) => (rx_mmi' = n_null /\ rx_exc' = e_null)
   [my_rx_mmi] ~(rx_mmi' = n_null) => (rx_umi' = u_null /\ rx_exc' = e_null)
   [my_rx_exc] ~(rx_exc' = e_null) => (rx_umi' = u_null /\ rx_mmi' = n_null)
   [my_tx_umi]  (tx_umi  = u_null) => (tx_umi' = tx_umi)
```

```
   [my_tx_mmi]  (tx_mmi  = n_null) => (tx_mmi' = tx_mmi)
   [my_tx_exc]  (tx_exc  = e_null) => (tx_exc' = tx_exc)
   [empty_tx_umi] ~(tx_umi  = u_null) => (tx_umi' = u_null)
   [empty_tx_mmi] ~(tx_mmi  = n_null) => (tx_mmi' = n_null)
   [empty_tx_exc] ~(tx_exc  = e_null) => (tx_exc' = e_null)

INVARIANT
   true

INITIALLY
   call = state_u_final /\ sap = null
      /\ time_out_ctr = 0  /\ time_out_his = off
      /\ umi_link = online /\ mmi_link = online
      /\ sap_manager = ok /\ qos_manager = ok
      /\ rx_umi = u_null /\ rx_mmi = n_null /\ rx_exc = e_null
      /\ tx_umi = u_null /\ tx_mmi = n_null /\ tx_exc = e_null

INSTRUCTIONS

% -------------------------------------------------------------------------

[]  call  = state_u_final /\ rx_umi = u_sap_req -->
    IF (sap_manager = ok) THEN
       (call' = state_u_sap_bound || rx_umi' = u_null || tx_umi' = u_sap_bound
       || sap' = bound || tx_exc' = e_set_timer || time_out_his' = on
       || time_out_ctr' = 0)
    ELSE
       (call' = state_u_final || rx_umi' = u_null || tx_umi' = u_sap_denied)

[]  call  = state_u_final /\ rx_umi = u_connect -->
    IF (mmi_link = online) THEN
       (IF (sap_manager = ok) THEN
          (IF (qos_manager = ok) THEN
             (call' = state_u_connect || rx_umi' = u_null || tx_mmi' = n_connect
             || sap' = active || tx_exc' = e_set_timer || time_out_his' = on
             || time_out_ctr' = 0)
          ELSE
             (call' = state_u_final || rx_umi' = u_null || tx_umi' = u_rejected))
       ELSE
          (call' = state_u_final || rx_umi' = u_null || tx_umi' = u_sap_denied))
    ELSE
       (call' = state_u_final || rx_umi' = u_null || tx_umi' = u_abort)

[]  call  = state_u_final /\ rx_umi = u_final -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_final

[]  call  = state_u_final /\ rx_umi = u_aborted -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_u_final /\ rx_mmi = n_disconnect -->
    call' = state_u_final || rx_mmi' = n_null || tx_mmi' = n_release

[]  call  = state_u_final /\ rx_mmi = n_connect -->
    call' = state_u_final || rx_mmi' = n_null || tx_mmi' = n_rejected

[]  call  = state_u_final /\ rx_mmi = n_release -->
    call' = state_u_final || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_u_final /\ ~(rx_umi = u_aborted \/ rx_umi = u_final \/ rx_umi = u_connect
    \/ rx_umi = u_sap_req \/ rx_umi = u_null) /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_abort

[]  call  = state_u_final /\ rx_umi = u_null /\ ~(rx_mmi = n_release \/ rx_mmi = n_connect
    \/ rx_mmi = n_disconnect \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_u_final || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_u_final /\ rx_exc = e_u_data -->
    call' = state_u_final || rx_exc' = e_null || tx_mmi' = n_disconnect

[]  call  = state_u_final /\ rx_exc = e_n_failed -->
    call' = state_u_final || rx_exc' = e_null || mmi_link' = offline

[]  call  = state_u_final /\ rx_exc = e_u_failed -->
    call' = state_u_final || rx_exc' = e_null || umi_link' = offline

[]  call  = state_u_final /\ rx_exc = e_n_recovered -->
    call' = state_u_final || rx_exc' = e_null || mmi_link' = online

[]  call  = state_u_final /\ rx_exc = e_u_recovered -->
    call' = state_u_final || rx_exc' = e_null || umi_link' = online

[]  call  = state_u_final /\ rx_umi = u_null /\ rx_mmi = n_null /\ ~(rx_exc = e_u_recovered
    \/ rx_exc = e_n_recovered \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
```

```
    \/ rx_exc = e_u_data \/ rx_exc = e_null) -->
    call' = state_u_final || rx_exc' = e_null || tx_exc' = e_null

% --------------------------------------------------------------------------

[]  call  = state_u_idle /\ rx_umi = u_final -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_final
    || tx_exc' = e_cancel_timer || time_out_his' = off || sap' = null

[]  call  = state_u_idle /\ rx_umi = u_connect -->
    call' = state_u_idle || rx_umi' = u_null || tx_umi' = u_rejected

[]  call  = state_u_idle /\ rx_umi = u_disconnect -->
    call' = state_u_idle || rx_umi' = u_null || tx_umi' = u_idle

[]  call  = state_u_idle /\ rx_mmi = n_disconnect -->
    call' = state_u_idle || rx_mmi' = n_null || tx_mmi' = n_release

[]  call  = state_u_idle /\ rx_mmi = n_release -->
    call' = state_u_idle || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_u_idle /\ rx_mmi = n_accepted -->
    call' = state_u_idle || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_u_idle /\ ~(rx_umi = u_disconnect \/ rx_umi = u_connect \/ rx_umi = u_final
    \/ rx_umi = u_null) /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_u_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_idle /\ rx_umi = u_null /\ ~(rx_mmi = n_accepted \/ rx_mmi = n_release
    \/ rx_mmi = n_disconnect \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_u_idle || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_u_idle /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline || tx_umi' = u_abort
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_idle /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_idle /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_idle || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1)
    ELSE
       (call' = state_u_aborting || rx_exc' = e_null || tx_umi' = u_abort
       || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_u_idle /\ rx_umi = u_null /\ rx_mmi = n_null /\ ~(rx_exc = e_time_out
    \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed \/ rx_exc = e_null) -->
    call' = state_u_idle || rx_exc' = e_null || tx_exc' = e_null

% --------------------------------------------------------------------------

[]  call  = state_u_sap_bound /\ rx_umi = u_sap_offer -->
    IF (qos_manager = ok) THEN
       (call' = state_u_sap_offered || rx_umi' = u_null || tx_umi' = u_sap_offered
       || tx_exc' = e_reset_timer || time_out_ctr' = 0 || sap' = passive)
    ELSE
       (call' = state_u_sap_bound || rx_umi' = u_null || tx_umi' = u_sap_denied)

[]  call  = state_u_sap_bound /\ rx_umi = u_sap_req -->
    call' = state_u_sap_bound || rx_umi' = u_null || tx_umi' = u_sap_bound

[]  call  = state_u_sap_bound /\ rx_umi = u_sap_withdraw -->
    call' = state_u_sap_bound || rx_umi' = u_null || tx_umi' = u_sap_bound

[]  call  = state_u_sap_bound /\ rx_umi = u_connect -->
    IF (qos_manager = ok) THEN
       (call' = state_u_connect || rx_umi' = u_null || tx_mmi' = n_connect
       || tx_exc' = e_reset_timer || time_out_ctr' = 0 || sap' = active)
    ELSE
       (call' = state_u_idle || rx_umi' = u_null || tx_umi' = u_rejected
       || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_u_sap_bound /\ rx_umi = u_final -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_final
    || tx_exc' = e_cancel_timer || time_out_his' = off || sap' = null

[]  call  = state_u_sap_bound /\ rx_mmi = n_disconnect -->
    call' = state_u_sap_bound || rx_mmi' = n_null || tx_mmi' = n_release

[]  call  = state_u_sap_bound /\ rx_mmi = n_release -->
```

```
    call' = state_u_sap_bound || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_u_sap_bound /\ ~(rx_umi = u_final \/ rx_umi = u_connect
    \/ rx_umi = u_sap_withdraw \/ rx_umi = u_sap_req \/ rx_umi = u_sap_offer
    \/ rx_umi = u_null) /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_u_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_sap_bound /\ rx_umi = u_null /\ ~(rx_mmi = n_release
    \/ rx_mmi = n_disconnect \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_u_sap_bound || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_u_sap_bound /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_sap_bound /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_sap_bound /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_sap_bound || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_null || tx_mmi' = n_null)
    ELSE
       (call' = state_aborting || rx_exc' = e_null || tx_umi' = u_abort
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_u_sap_bound /\ rx_umi = u_null /\ rx_mmi = n_null /\ ~(rx_exc = e_time_out
    \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed \/ rx_exc = e_null) -->
    call' = state_u_sap_bound || rx_exc' = e_null || tx_exc' = e_null

% ---------------------------------------------------------------------------

[]  call  = state_u_sap_offered /\ rx_umi = u_sap_withdraw -->
    call' = state_u_sap_bound || rx_umi' = u_null || tx_umi' = u_sap_bound
    || tx_exc' = e_reset_timer || time_out_ctr' = 0 || sap' = bound

[]  call  = state_u_sap_offered /\ rx_umi = u_final -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_final
    || tx_exc' = e_cancel_timer || time_out_his' = off || sap' = null

[]  call  = state_u_sap_offered /\ rx_umi = u_sap_offer -->
    call' = state_u_sap_offered || rx_umi' = u_null || tx_umi' = u_sap_offered

[]  call  = state_u_sap_offered /\ rx_umi = u_rejected -->
    call' = state_u_sap_offered || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_u_sap_offered /\ rx_mmi = n_connect -->
    call' = state_n_connect || rx_mmi' = n_null || tx_umi' = u_connect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_sap_offered /\ rx_mmi = n_connect -->
    call' = state_u_sap_offered || rx_mmi' = n_null || tx_mmi' = n_rejected

[]  call  = state_u_sap_offered /\ rx_mmi = n_disconnect -->
    call' = state_u_sap_offered || rx_mmi' = n_null || tx_mmi' = n_release

[]  call  = state_u_sap_offered /\ rx_mmi = n_release -->
    call' = state_u_sap_offered || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_u_sap_offered /\ ~(rx_umi = u_rejected \/ rx_umi = u_sap_offer
    \/ rx_umi = u_final \/ rx_umi = u_sap_withdraw \/ rx_umi = u_null)
    /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_u_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_sap_offered /\ rx_umi = u_null /\ ~(rx_mmi = n_release \/
    rx_mmi = n_disconnect \/ rx_mmi = n_connect \/ rx_mmi = n_null)
    /\ rx_exc = e_null -->
    call' = state_u_sap_offered || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_u_sap_offered /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_sap_offered /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_sap_offered /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_sap_offered || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
```

```
      || tx_umi' = u_null || tx_mmi' = n_null)
   ELSE
      (call' = state_aborting || rx_exc' = e_null || tx_umi' = u_abort
      || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_u_sap_offered /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_u_sap_offered || rx_exc' = e_null || tx_exc' = e_null

% -------------------------------------------------------------------------

[]  call  = state_n_connect /\ rx_umi = u_sap_withdraw -->
    call' = state_u_sap_bound || rx_umi' = u_null || tx_umi' = u_sap_bound
    || tx_mmi' = n_rejected || tx_exc' = e_reset_timer || time_out_ctr' = 0
    || sap' = bound

[]  call  = state_n_connect /\ rx_umi = u_final -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_final
    || tx_mmi' = n_rejected || tx_exc' = e_cancel_timer || time_out_his' = off
    || sap' = null

[]  call  = state_n_connect /\ rx_umi = u_accepted -->
    IF (qos_manager = ok) THEN
       (call' = state_u_accepted || rx_umi' = u_null || tx_mmi' = n_accepted
       || tx_exc' = e_reset_timer || time_out_ctr' = 0)
    ELSE
       (call' = state_n_disconnect || rx_umi' = u_null || tx_umi' = u_disconnect
       || tx_mmi' = n_rejected || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_n_connect /\ rx_umi = u_rejected -->
    call' = state_u_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_mmi' = n_rejected || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_connect /\ rx_umi = u_sap_offer -->
    call' = state_n_connect || rx_umi' = u_null || tx_umi' = u_connect

[]  call  = state_n_connect /\ rx_mmi = n_disconnect -->
    call' = state_n_disconnect || rx_mmi' = n_null || tx_umi' = u_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_connect /\ rx_mmi = n_connect -->
    call' = state_n_connect || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_n_connect /\ ~(rx_umi = u_sap_offer \/ rx_umi = u_rejected
    \/ rx_umi = u_accepted \/ rx_umi = u_final \/ rx_umi = u_sap_withdraw
    \/ rx_umi = u_null) /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_u_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_mmi' = n_rejected || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_connect /\ rx_umi = u_null /\ ~(rx_mmi = n_connect
    \/ rx_mmi = n_disconnect \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_aborting || rx_mmi' = n_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_connect /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_connect /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_connect /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_n_connect || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_connect || tx_mmi' = n_null)
    ELSE
       (call' = state_n_disconnect || rx_exc' = e_null || tx_umi' = u_disconnect
       || tx_mmi' = n_rejected || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_n_connect /\ rx_umi = u_null /\ rx_mmi = n_null /\ ~(rx_exc = e_time_out
    \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed \/ rx_exc = e_null) -->
    call' = state_n_connect || rx_exc' = e_null || tx_exc' = e_null

% -------------------------------------------------------------------------

[]  call  = state_u_accepted /\ rx_umi = u_disconnect -->
    call' = state_u_disconnect || rx_umi' = u_null || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_umi = u_final -->
    call' = state_u_fin_pend || rx_umi' = u_null || tx_mmi' = n_disconnect
```

```
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_umi = u_accepted -->
    call' = state_u_accepted || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_u_accepted /\ rx_mmi = n_complete -->
    call' = state_n_complete || rx_mmi' = n_null || tx_umi' = u_complete
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_mmi = n_disconnect -->
    call' = state_n_disconnect || rx_mmi' = n_null || tx_umi' = u_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_mmi = n_connect -->
    call' = state_u_accepted || rx_mmi' = n_null || tx_mmi' = n_accepted

[]  call  = state_u_accepted /\ ~(rx_umi = u_accepted \/ rx_umi = u_final
    \/ rx_umi = u_disconnect \/ rx_umi = u_null) /\ rx_mmi = n_null
    /\ rx_exc = e_null -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_umi = u_null /\ ~(rx_mmi = n_connect
    \/ rx_mmi = n_disconnect \/ rx_mmi = n_complete \/ rx_mmi = n_null)
    /\ rx_exc = e_null -->
    call' = state_aborting || rx_mmi' = n_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_exc = e_u_data -->
    call' = state_n_complete || rx_exc' = e_null || tx_umi' = u_complete
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_accepted /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_accepted || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_null || tx_mmi' = n_null)
    ELSE
       (call' = state_aborting || rx_exc' = e_null || tx_umi' = u_abort
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_u_accepted /\ rx_umi = u_null /\ rx_mmi = n_null
    /\ ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_u_data \/ rx_exc = e_null) -->
    call' = state_u_accepted || rx_exc' = e_null || tx_exc' = e_null

% ---------------------------------------------------------------------------

[]  call  = state_n_complete /\ rx_umi = u_disconnect -->
    call' = state_u_disconnect || rx_umi' = u_null || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_umi = u_final -->
    call' = state_u_fin_pend || rx_umi' = u_null || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_umi = u_accepted -->
    call' = state_n_complete || rx_umi' = u_null || tx_umi' = u_complete

[]  call  = state_n_complete /\ rx_umi = u_complete -->
    call' = state_n_complete || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_n_complete /\ rx_mmi = n_disconnect -->
    call' = state_n_disconnect || rx_mmi' = n_null || tx_umi' = u_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_mmi = n_complete -->
    call' = state_n_complete || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_n_complete /\ rx_mmi = n_accepted -->
    call' = state_n_complete || rx_mmi' = n_null || tx_mmi' = n_complete

[]  call  = state_n_complete /\ ~(rx_umi = u_complete \/ rx_umi = u_accepted
    \/ rx_umi = u_final \/ rx_umi = u_disconnect \/ rx_umi = u_null)
    /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_abort
```

```
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_umi = u_null /\ ~(rx_mmi = n_accepted
    \/ rx_mmi = n_complete \/ rx_mmi = n_disconnect \/ rx_mmi = n_null)
    /\ rx_exc = e_null -->
    call' = state_aborting || rx_mmi' = n_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_exc = e_idle -->
    call' = state_aborting || rx_exc' = e_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_complete /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_n_complete || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_null || tx_mmi' = n_null)
    ELSE
       (call' = state_aborting || rx_exc' = e_null || tx_umi' = u_abort
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_n_complete /\ rx_umi = u_null /\ rx_mmi = n_null
    /\ ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_idle \/ rx_exc = e_null) -->
    call' = state_n_complete || rx_exc' = e_null || tx_exc' = e_null

% --------------------------------------------------------------------------

[]  call  = state_u_connect /\ rx_umi = u_disconnect -->
    call' = state_u_disconnect || rx_umi' = u_null || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_umi = u_final -->
    call' = state_u_fin_pend || rx_umi' = u_null || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_umi = u_connect -->
    call' = state_u_connect || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_u_connect /\ rx_mmi = n_rejected -->
    call' = state_u_idle || rx_mmi' = n_null || tx_umi' = u_rejected
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_mmi = n_accepted -->
    call' = state_n_accepted || rx_mmi' = n_null || tx_umi' = u_accepted
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_mmi = n_accepted -->
    call' = state_n_aborting || rx_mmi' = n_null || tx_umi' = u_rejected
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_mmi = n_disconnect -->
    call' = state_u_idle || rx_mmi' = n_null || tx_umi' = u_rejected
    || tx_mmi' = n_release || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ ~(rx_umi = u_connect \/ rx_umi = u_final
    \/ rx_umi = u_disconnect \/ rx_umi = u_null) /\ rx_mmi = n_null
    /\ rx_exc = e_null -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_umi = u_null /\ ~(rx_mmi = n_disconnect
    \/ rx_mmi = n_accepted \/ rx_mmi = n_rejected \/ rx_mmi = n_null)
    /\ rx_exc = e_null -->
    call' = state_aborting || rx_mmi' = n_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_connect /\ rx_exc = e_time_out -->
```

```
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_connect || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_null || tx_mmi' = n_connect)
    ELSE
       (call' = state_u_idle || rx_exc' = e_null || tx_umi' = u_rejected
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_u_connect /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_u_connect || rx_exc' = e_null || tx_exc' = e_null

% ------------------------------------------------------------------------

[]  call  = state_n_accepted /\ rx_umi = u_complete -->
    call' = state_n_complete || rx_umi' = u_null || tx_mmi' = n_complete
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_umi = u_connect -->
    call' = state_n_accepted || rx_umi' = u_null || tx_umi' = u_accepted

[]  call  = state_n_accepted /\ rx_umi = u_disconnect -->
    call' = state_u_disconnect || rx_umi' = u_null || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_umi = u_final -->
    call' = state_u_fin_pend || rx_umi' = u_null || tx_mmi' = n_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_mmi = n_disconnect -->
    call' = state_n_disconnect || rx_mmi' = n_null || tx_umi' = u_disconnect
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_mmi = n_accepted -->
    call' = state_n_accepted || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_n_accepted /\ ~(rx_umi = u_final \/ rx_umi = u_disconnect
    \/ rx_umi = u_connect \/ rx_umi = u_complete \/ rx_umi = u_null)
    /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_umi = u_null /\ ~(rx_mmi = n_accepted
    \/ rx_mmi = n_disconnect \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_aborting || rx_mmi' = n_null || tx_umi' = u_abort
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_accepted /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_n_accepted || rx_exc' = e_null
       || time_out_ctr' = time_out_ctr + 1 || tx_umi' = u_accepted || tx_mmi' = n_null)
    ELSE
       (call' = state_aborting || rx_exc' = e_null || tx_umi' = u_abort
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_n_accepted /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_n_accepted || rx_exc' = e_null || tx_exc' = e_null

% ------------------------------------------------------------------------

[]  call  = state_u_disconnect /\ rx_umi = u_final -->
    call' = state_u_fin_pend || rx_umi' = u_null || tx_umi' = u_null
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_disconnect /\ rx_umi = u_disconnect -->
    call' = state_u_disconnect || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_u_disconnect /\ rx_mmi = n_release -->
    call' = state_u_idle || rx_mmi' = n_null || tx_umi' = u_idle
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_disconnect /\ rx_mmi = n_disconnect -->
    call' = state_u_disconnect || rx_mmi' = n_null || tx_mmi' = n_release
```

```
[]  call  = state_u_disconnect /\ ~(rx_umi = u_disconnect \/ rx_umi = u_final
    \/ rx_umi = u_null) /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_disconnect /\ rx_umi = u_null /\ ~(rx_mmi = n_disconnect
    \/ rx_mmi = n_release \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_u_disconnect || rx_mmi' = n_null || tx_mmi' = n_null

[]  call  = state_u_disconnect /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_disconnect /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_disconnect /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_disconnect || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_null || tx_mmi' = n_disconnect)
    ELSE
       (call' = state_u_idle || rx_exc' = e_null || tx_umi' = u_idle
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_u_disconnect /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_u_disconnect || rx_exc' = e_null || tx_exc' = e_null

% --------------------------------------------------------------------------

[]  call  = state_n_disconnect /\ rx_umi = u_idle -->
    call' = state_u_idle || rx_umi' = u_null || tx_mmi' = n_release
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_disconnect /\ rx_umi = u_final -->
    call' = state_u_final || rx_umi' = u_null || tx_mmi' = n_release
    || tx_umi' = u_final || tx_exc' = e_cancel_timer || time_out_his' = off
    || sap' = null

[]  call  = state_n_disconnect /\ rx_umi = u_disconnect -->
    call' = state_u_idle || rx_umi' = u_null || tx_mmi' = n_release
    || tx_umi' = u_idle || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_disconnect /\ rx_mmi = n_disconnect -->
    call' = state_u_idle || rx_mmi' = n_null || tx_mmi' = n_release
    || tx_umi' = u_idle || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_disconnect /\ ~(rx_umi = u_disconnect \/ rx_umi = u_final
    \/ rx_umi = u_idle \/ rx_umi = u_null) /\ rx_mmi = n_null
    /\ rx_exc = e_null -->
    call' = state_n_disconnect || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_n_disconnect /\ rx_umi = u_null /\ ~(rx_mmi = n_disconnect
    \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_n_disconnect || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_n_disconnect /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_disconnect /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_disconnect /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_n_disconnect || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_disconnect || tx_mmi' = n_null)
    ELSE
       (call' = state_u_aborting || rx_exc' = e_null || tx_umi' = u_abort
       || tx_mmi' = n_release || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_n_disconnect /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_n_disconnect || rx_exc' = e_null || tx_exc' = e_null

% --------------------------------------------------------------------------

[]  call  = state_u_fin_pend /\ rx_umi = u_final -->
```

```
    call' = state_u_fin_pend || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_u_fin_pend /\ rx_mmi = n_release -->
    call' = state_u_final || rx_mmi' = n_null || tx_umi' = u_final
    || tx_exc' = e_cancel_timer || time_out_his' = off || sap' = null

[]  call  = state_u_fin_pend /\ rx_mmi = n_disconnect -->
    call' = state_u_final || rx_mmi' = n_null || tx_umi' = u_final
    || tx_mmi' = n_release || tx_exc' = e_cancel_timer || time_out_his' = off
    || sap' = null

[]  call  = state_u_fin_pend /\ ~(rx_umi = u_final \/ rx_umi = u_null)
    /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_abort
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_fin_pend /\ rx_umi = u_null /\ ~(rx_mmi = n_disconnect
    \/ rx_mmi = n_release \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_u_fin_pend || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_u_fin_pend /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_fin_pend /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_u_fin_pend /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_fin_pend || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_null || tx_mmi' = n_disconnect)
    ELSE
       (call' = state_u_final || rx_exc' = e_null || tx_umi' = u_final || sap' = null
       || tx_mmi' = n_disconnect || tx_exc' = e_cancel_timer || time_out_his' = off)

[]  call  = state_u_fin_pend /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_u_fin_pend || rx_exc' = e_null || tx_exc' = e_null

% --------------------------------------------------------------------------

[]  call  = state_u_aborting /\ rx_umi = u_aborted -->
    call' = state_u_final || rx_umi' = u_null || tx_umi' = u_null || sap' = null
    || tx_exc' = e_cancel_timer || time_out_his' = off

[]  call  = state_u_aborting /\ ~(rx_umi = u_aborted \/ rx_umi = u_null)
    /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_u_aborting || rx_umi' = u_null || tx_umi' = u_abort

[]  call  = state_u_aborting /\ rx_umi = u_null /\ ~(rx_mmi = n_null)
    /\ rx_exc = e_null -->
    call' = state_u_aborting || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_u_aborting /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort

[]  call  = state_u_aborting /\ rx_exc = e_u_failed -->
    IF (mmi_link = online) THEN
       (call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)
    ELSE
       (call' = state_u_aborting || rx_exc' = e_null || umi_link' = offline
       || tx_mmi' = n_null)

[]  call  = state_u_aborting /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_u_aborting || rx_exc' = e_null ||
       time_out_ctr' = time_out_ctr + 1 || tx_umi' = u_abort)
    ELSE
       (call' = state_u_final || rx_exc' = e_null || tx_umi' = u_abort
       || tx_exc' = e_cancel_timer || time_out_his' = off || sap' = null)

[]  call  = state_u_aborting /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_u_aborting || rx_exc' = e_null || tx_exc' = e_null

% --------------------------------------------------------------------------

[]  call  = state_n_aborting /\ rx_mmi = n_release -->
```

```
    call' = state_u_final || rx_mmi' = n_null || tx_mmi' = n_null
    || tx_exc' = e_cancel_timer || time_out_his' = off || sap' = null

[]  call  = state_n_aborting /\ ~(rx_umi = u_null) /\ rx_mmi = n_null
    /\ rx_exc = e_null -->
    call' = state_n_aborting || rx_umi' = u_null || tx_umi' = u_abort

[]  call  = state_n_aborting /\ rx_umi = u_null /\ ~(rx_mmi = n_release
    \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_n_aborting || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_n_aborting /\ rx_exc = e_n_failed /\ (umi_link = online) -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_n_aborting /\ rx_exc = e_n_failed /\ (umi_link = offline) -->
    call' = state_n_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_null

[]  call  = state_n_aborting /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect

[]  call  = state_n_aborting /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_n_aborting || rx_exc' = e_null
       || time_out_ctr' = time_out_ctr + 1 || tx_mmi' = n_disconnect)
    ELSE
       (call' = state_u_final || rx_exc' = e_null || tx_mmi' = n_disconnect
       || tx_exc' = e_cancel_timer || time_out_his' = off || sap' = null)

[]  call  = state_n_aborting /\ rx_umi = u_null /\ rx_mmi = n_null /\
    ~(rx_exc = e_time_out \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed
    \/ rx_exc = e_null) -->
    call' = state_n_aborting || rx_exc' = e_null || tx_exc' = e_null

% ---------------------------------------------------------------------------

[]  call  = state_aborting /\ rx_mmi = n_release -->
    call' = state_u_aborting || rx_mmi' = n_null || tx_mmi' = n_null
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_aborting /\ rx_umi = u_aborted -->
    call' = state_n_aborting || rx_umi' = u_null || tx_umi' = u_null
    || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_aborting /\ rx_mmi = n_disconnect -->
    call' = state_aborting || rx_mmi' = n_null || tx_mmi' = n_release

[]  call  = state_aborting /\ rx_umi = u_final -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_null

[]  call  = state_aborting /\ ~(rx_umi = u_final \/ rx_umi = u_aborted
    \/ rx_umi = u_null) /\ rx_mmi = n_null /\ rx_exc = e_null -->
    call' = state_aborting || rx_umi' = u_null || tx_umi' = u_abort

[]  call  = state_aborting /\ rx_umi = u_null /\ ~(rx_mmi = n_disconnect
    \/ rx_mmi = n_release \/ rx_mmi = n_null) /\ rx_exc = e_null -->
    call' = state_aborting || rx_mmi' = n_null || tx_mmi' = n_disconnect

[]  call  = state_aborting /\ rx_exc = e_n_failed -->
    call' = state_u_aborting || rx_exc' = e_null || mmi_link' = offline
    || tx_umi' = u_abort || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_aborting /\ rx_exc = e_u_failed -->
    call' = state_n_aborting || rx_exc' = e_null || umi_link' = offline
    || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0

[]  call  = state_aborting /\ rx_exc = e_time_out -->
    IF (time_out_ctr < time_out_max) THEN
       (call' = state_aborting || rx_exc' = e_null || time_out_ctr' = time_out_ctr + 1
       || tx_umi' = u_abort || tx_mmi' = n_disconnect)
    ELSE
       (call' = state_u_aborting || rx_exc' = e_null || tx_umi' = u_abort
       || tx_mmi' = n_disconnect || tx_exc' = e_reset_timer || time_out_ctr' = 0)

[]  call  = state_aborting /\ rx_umi = u_null /\ rx_mmi = n_null /\ ~(rx_exc = e_time_out
    \/ rx_exc = e_u_failed \/ rx_exc = e_n_failed \/ rx_exc = e_null) -->
    call' = state_aborting || rx_exc' = e_null || tx_exc' = e_null

% ---------------------------------------------------------------------------

FAIRNESS
```

```
   [fairtime] WF(time_out_his = on, rx_exc = e_time_out)

PROPERTIES
   [nolivelock] MCL("ag(af(call = state_u_final))")

   [time]       MCL("ag((rx_exc = e_time_out) ==> (time_out_his = on))")
   [umifail]    MCL("ag((rx_exc = e_u_failed) ==> ax(umi_link = offline))")
   [mmifail]    MCL("ag((rx_exc = e_n_failed) ==> ax(mmi_link = offline))")
   [umirec]     MCL("ag((rx_exc = e_u_recovered) ==> ax(umi_link = online))")
   [mmirec]     MCL("ag((rx_exc = e_n_recovered) ==> ax(mmi_link = online))")

   [sapbound]   MCL("ag((call = state_u_sap_bound) ==> (sap = bound))")
   [sappassive] MCL("ag((call = state_u_sap_offered \/ call = state_n_connect
                       \/ state_u_accepted) ==> (sap = passive))")
   [sapactive]  MCL("ag((call = state_u_connect \/ call = state_n_accepted)
                       ==> (sap = passive))")

END.
```

Appendix C

Correlation in the Reference Stream

This appendix presents results which indicate that a CBR reference stream becomes correlated as a result of multiplexing, and which confirm the burst expansion results of Chapter 7.

C.1 The Index of Dispersion for Counts

The Index of Dispersion for Counts (IDC) is a valuable measure of the dependence between successive cell arrival times. It has been used in studies of multiplexed packet voice [Sriram86] and on-off ATM sources [Roberts92]. Let $N(t)$ denote the number of cell arrivals in an interval t. The IDC is defined by

$$I(t) = \frac{\mathrm{Var}(N(t))}{\mathbb{E}(N(t))}, \text{ for } t > 0. \tag{C.1}$$

The IDC is related to the Index of Dispersion for Intervals (IDI), which measures the cumulative covariance between groups of k interarrival times, and as such is a more general form of the inter-cell time covariance results presented in Section 6.6.2.1. Examining values of $I(t), t > 0$ reveals the correlation in a traffic stream, on different time scales. For a Poisson process $I(t) = 1, \forall t > 0$; thus $I(t)$ measures the degree of deviation of the stream from a renewal process.

Values of $I(t)$ can be used to parameterise analytical models of (superposed) bursty sources. Two candidates include $I(\infty)$ and $\max_t I(t)$. In [Roberts92], results are given which show that three parameters fitted to analytical models yield the closest approximation to the measured queueing delay of real traffic:

- the mean cell rate m (i.e. the load),
- the value of $I(1/m)$, and

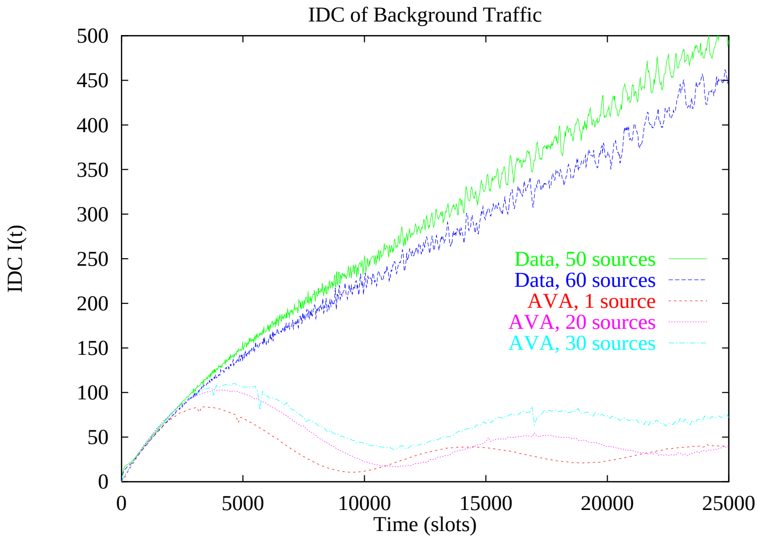


Figure C.1: IDC of the bursty background traffic types

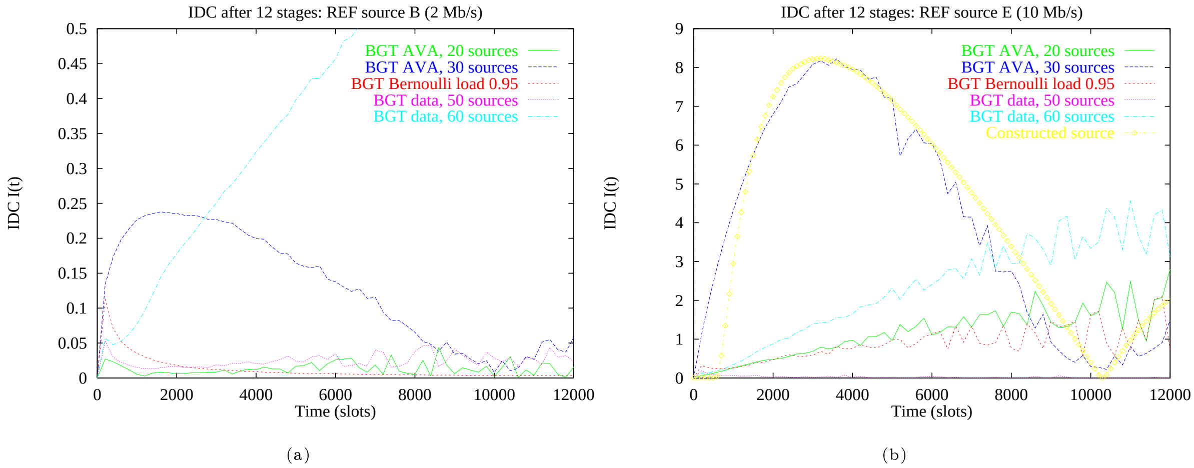


Figure C.2: IDC of the CBR reference sources B and E after 12 stages

- the value of $\max_{t>0} I(t)$.

Two other useful properties to note are that the slope of the IDC can be used as a measure of the burstiness of the traffic, and that if the IDC increases monotonically, the traffic exhibits signs of "long range dependence". In this brief investigation the IDC is used simply to better characterise the background traffic processes used, and to illustrate their effects on streams with which they are multiplexed.

Figure C.1 shows the IDC of the bursty background traffic types used in the experiments of Chapters 6 and 7, and a single AVA traffic source. The Bernoulli background traffic used has an IDC of 1; in comparison the bursty background traffic types have extremely high values of $I(t)$ and, in the case of the AVA traffic, exhibit correlations (periodicity) on the time scale of a video frame. On the time scale of multiple frames these correlations are less significant, however the traffic is still extremely bursty. The value of $\max_{t>0} I(t)$ is 100 for 20 AVA streams, and 110 for 30 AVA streams, but for both Bellcore data loads, the IDC increases (approximately linearly) and appears to reach a bound of about 5000, however this is not clear due to the limited data set length.

C.2 CBR Reference Streams

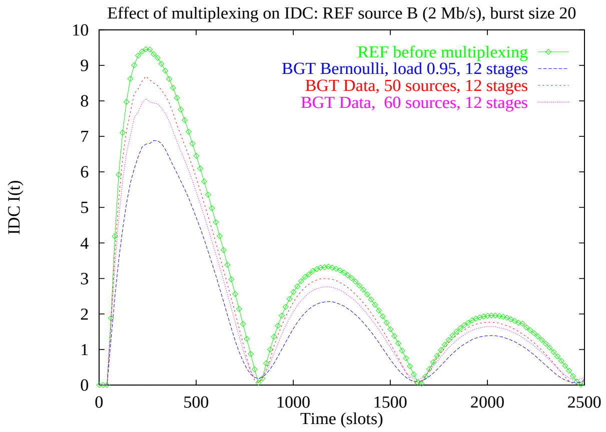
Figures C.2 (a) and (b) show the IDC of the reference sources B and E after 12 stages of multiplexing with each of the background traffic types. A CBR stream has an IDC close to 0. In both cases the effects of multiplexing are as follows:

- $I(t)$ has a lower value for source B than source E, irrespective of the background traffic.
- With Bernoulli traffic, the value of $I(t)$ is close to 1, more so for source E (as expected, due to the more rapid convergence).
- For source B, $\max_{t>0} I(t)$ using 60 data sources is about 2.5, and with 30 AVA sources about 0.25.
- For source E, $\max_{t>0} I(t)$ using 60 data sources is about 7, and with 30 AVA sources about 8.
- The reference stream becomes correlated in the same way as the background traffic: for example with AVA traffic, it exhibits a periodicity on the video frame time scale.
- In the case of source E, the perturbation of the reference stream is particularly bad, and the correlation explains why, in spite of their similar loads, using 30 AVA sources the inter-cell time variance is far lower than using Bernoulli traffic, in Figure 6.12.
- Figure C.2 (b) shows also a trivially constructed deterministic burst-silence source with the same period as the AVA multiplex and a burst size of 16 cells back-to-back. The constructed source is a good approximation of the perturbed reference stream, on the long time scale.

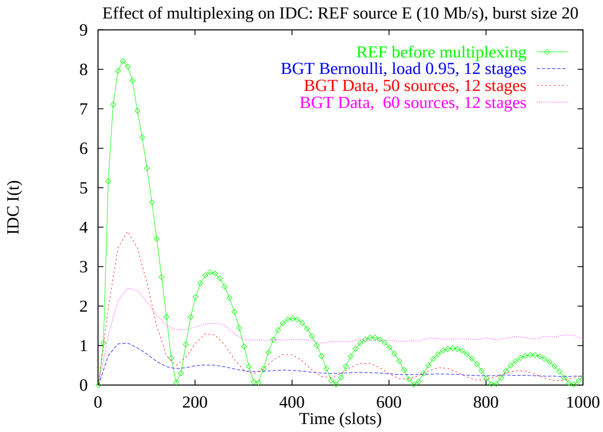
Further work should enable the fitting of accurate statistical models to these parameters.

C.3 Bursty Reference Streams

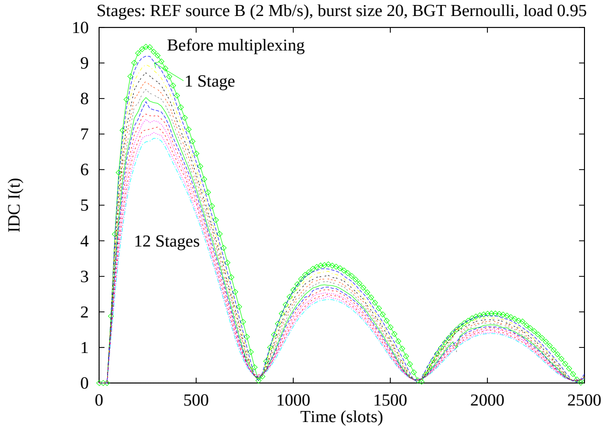
The IDC can be used to measure the degree to which the bursty reference sources become more like renewal processes as they proceed through the network. Figures C.3 (a) and (b) show for each source the effect on the IDC after 12 stages of multiplexing with two different loads of data background traffic. In the case of source E, the burst expansion is more rapid and therefore the reference traffic is far closer to being a renewal process after 12 stages. This is particularly true using the high load of data traffic. Figures C.3 (c) and (d) show the cumulative effects on the IDC over successive stages, using Bernoulli traffic, and the same results are shown in Figures C.3 (e) and (f). After 12 stages source E is far less bursty, and is close to exhibiting the renewal property.



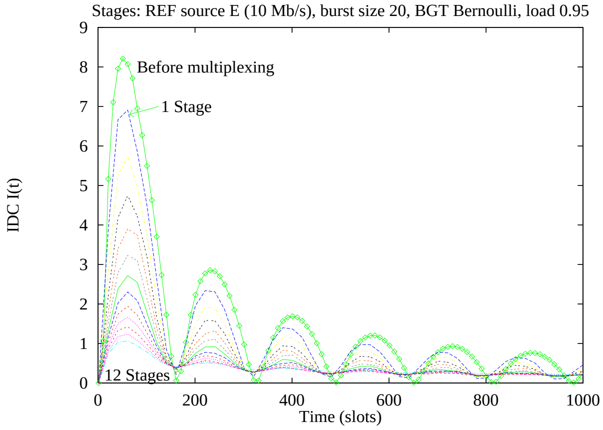
(a)



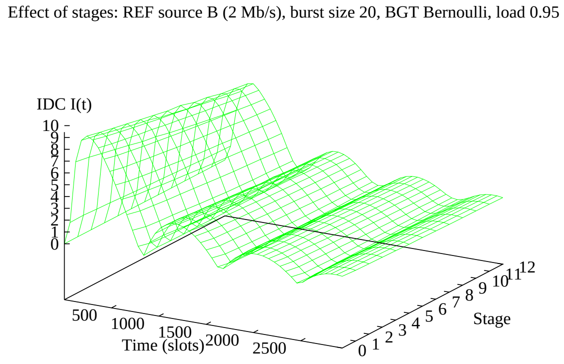
(b)



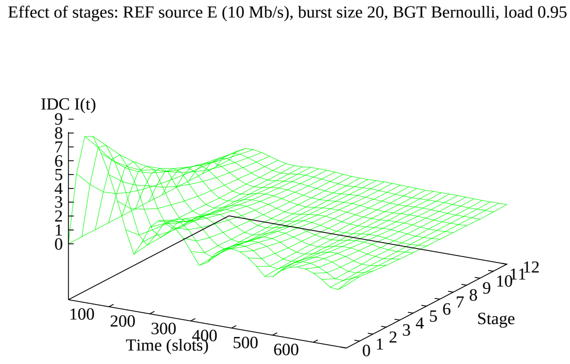
(c)



(d)



(e)



(f)

Figure C.3: IDC after 12 stages (a,b), and IDC with increasing stages shown in two (c,d) and three (e,f) dimensions, bursty reference sources B and E