Dynamic Provisioning of Resource-Assured and Programmable Virtual Private Networks

Rebecca Isaacs

Darwin College
University of Cambridge

A dissertation submitted for the degree of
Doctor of Philosophy

December 2000

Summary

Virtual Private Networks (VPNs) provide dedicated connectivity to a closed group of users on a shared network. VPNs have traditionally been deployed for reasons of economy of scale, but have either been statically defined, requiring manual configuration, or else unable to offer any quality of service (QoS) guarantees.

This dissertation describes VServ, a service offering dynamic and resource-assured VPNs that can be acquired and modified on demand. In VServ, a VPN is both a subset of physical resources, such as bandwidth and label space, together with the means to perform fine-grained management of those resources. This network programmability, combined with QoS guarantees, enables the multiservice network—a single universal network that can support all types of service and thus be efficient, cost-effective and flexible.

VServ is deployed over a network control framework known as Tempest. The Tempest explicitly distinguishes between inter- and intra-VPN resource management mechanisms. This makes the dynamic resource reallocation capabilities of VServ viable, whilst handling highly dynamic VPNs or a large number of VPNs. Extensions to the original implementation of the Tempest to support dynamically reconfigurable QoS are detailed.

A key part of a dynamic and responsive VPN service is fully automated VPN provisioning. A notation for VPN specification is described, together with mechanisms for incorporating policies of the service provider and the current resource availability in the network into the design process. The search for a suitable VPN topology can be expressed as an optimisation problem that is not computationally tractable except for very small networks. This dissertation describes how the search is made practical by tailoring it according to the characteristics of the desired VPN.

Availability of VServ is addressed with a proposal for distributed VPN creation. A resource revocation protocol exploits the dynamic resource management capabilities of VServ to allow adaptation in the control plane on a per-VPN basis. Managed resource revocation supports highly flexible resource allocation and reallocation policies, allowing VServ to efficiently provision for short-lived or highly dynamic VPNs.

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 my 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 [Isaacs00] and [Isaacs01].

Acknowledgements

I would like to thank my supervisor, Ian Leslie, for his advice and encouragement. I am also grateful for the help and insights offered by members of the Systems Research Group, in particular Herbert Bos, Richard Mortier, Steve Hand, Sean Rooney and Kobus van der Merwe.

For reading and commenting on earlier drafts of this dissertation, I am indebted to Timothy Roscoe, Sean Rooney, Richard Mortier, Ian Leslie, Steve Hand, Herbert Bos and Paul Barham.

I would also like to thank all the members of the IP Group at Sprint Advanced Technology Labs where I spent an extremely enjoyable and rewarding summer as an intern.

Contents

List of Figures

Glossary

AAL	ATM Adaptation Layer
ABR	Available Bit Rate
API	Application Programming Interface
AS	Autonomous System
ATM	Asynchronous Transfer Mode
BB	Bandwidth Broker
BGP	Border Gateway Protocol
BGRP	Border Gateway Reservation Protocol
BISDN	Broadband Integrated Services Digital Network
BICA	Bootstrap IP Control Architecture
BNF	Backus-Naur Form
CAC	Connection Admission Control
CBR	Constant Bit Rate
COPS	Common Open Policy Service
CORBA	Common Object Request Broker Architecture
CRACA	CoRny Adaptive Control Architecture
DPE	Distributed Processing Environment
EE	Execution Environment
FEC	Forwarding Equivalence Class
GARP	Generic Attributes Registration Protocol
GCAC	Generic CAC
GIOP	General Inter-ORB Protocol
GNU	GNU's not Unix
GVRP	GARP VLAN Registration Protocol
GSMP	General Switch Management Protocol
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB Protocol
IN	Intelligent Network
INCA	Intelligent Network Control Architecture
IP	Internet Protocol
ISP	Internet Service Provider
ITU	International Telecommunications Union

JPEG	Joint Photographic Experts Group
L2TP	Layer Two Tunnelling Protocol
LAN	Local Area Network
LDP	Label Distribution Protocol
LSP	Label Switched Path
LSR	Label Switching Router
MAC	Media Access Control
MPEG	Moving Picture Experts Group
MPLS	Multiprotocol Label Switching
MST	Minimum Spanning Tree
NNI	Network-Network Interface
NRT-VBR	Non-Real-Time Variable Bit Rate
ORB	Object Request Broker
OSPF	Open Shortest Path First
PCR	Peak Cell Rate
PDU	Protocol Data Unit
PHB	Per-Hop Forwarding Behaviour
PNNI	Private NNI
PPP	Point to Point Protocol
PPTP	Point to Point Tunnelling Protocol
PSTN	Public Switched Telephone Network
PVC	Permanent Virtual Circuit
QoS	Quality of Service
RM	Resource Management
RSVP	Resource ReSerVation Protocol
RT-VBR	Real-Time Variable Bit Rate
SCR	Sustainable Cell Rate
SCTF	Selective Closest Terminal First
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SMT	Steiner Minimal Tree
SVC	Switched Virtual Circuit
SS7	Signalling System No. 7
TCP	Transmission Control Protocol
UBR	Unspecified Bit Rate
UDP	User Datagram Protocol
UNI	User-Network Interface
UPC	Usage Parameter Control
VACANT	Video Adaptive Control with A kNown Topology

VAN	Virtual Active Network
VC	Virtual Circuit
VCI	Virtual Channel Identifier
VL	Videoman Light
VLAN	Virtual Local Area Network
VPI	Virtual Path Identifier
VPN	Virtual Private Network
VSI	Virtual Switch Interface
VSP	VPN Service Provider
XTI	X/Open Transport Interface

Chapter 1

Introduction

A Virtual Private Network (VPN) uses a share of the physical resources of a real network to provide dedicated and secure connectivity. When VPNs can be created and deleted dynamically, and when this share of resources is guaranteed, then it is desirable to automate VPN management and to optimise inter-VPN resource allocation. This dissertation describes *VServ*, a network service that offers dynamic and resource-assured VPNs. The service allows each VPN to have customised control of its resource allocation, and VPN creation and modification operations can take place on short timescales. VServ exploits the capabilities of a multiservice network in an open, programmable and resource-controlled manner.

1.1 Motivation

The motivation for a dynamic and resource-assured VPN service is three-fold. Firstly, VPNs have traditionally been deployed for reasons of economy of scale [Hall94], but have either been statically defined, requiring manual configuration, or else unable to offer any *quality of service* (QoS) guarantees. The VPNs offered by VServ are acquired 'on demand' and have a guaranteed and explicit resource allocation.

Secondly, previous research on open network control [van der Merwe97b, Rooney98a] provides the mechanisms for realising a *multiservice* network which can support many diverse and sophisticated network services. VServ capitalises upon this work and extends it to support the exploitation of multiservice capabilities.

Finally, the domain-wide resource management problems considered in this dissertation are relevant to ongoing work on the provision of QoS in the Internet [Berson98, Nichols99, Bernet00]. The approach taken in VServ allows resource management within the network at a range of granularities, and in so doing, addresses some of the concerns of other proposals.

The following sections examine each of these points in more detail.

1.1.1 A Better VPN

VPNs are usually created through the provision of private connectivity on public network infrastructure. For many users, privately owned and controlled networks are prohibitively expensive, creating a market in which network owners or operators provide network access to third parties through VPNs. A VPN user may obtain their connectivity directly, for example by leased lines or virtual circuits in a carrier's network, or else they can make use of encapsulation techniques to *tunnel* [Simpson95], over some other network (typically the Internet[1]). Often a combination of the two is used—in a corporate VPN the main sites may be connected by virtual circuits, with individual employees gaining access through their local Internet service provider (ISP).

One advantage of a tunnelled VPN is that its lightweight and dynamic topology makes it usable by mobile or occasional users, without the overhead of a permanent resource commitment. Another advantage is the ability to use a private addressing scheme over a public network. However a significant drawback is that there is no mechanism for assuring VPN resources. Its traffic will suffer from delay and loss due to congestion just as if it were not being carried within a VPN.

Leased lines or virtual circuits can be used to provide resource-guaranteed VPNs, but typically are inflexible, requiring a great deal of administrative overhead and manual establishment and maintenance. Although existing asynchronous transfer mode (ATM) networks are able to support dynamic provisioning of resource-guaranteed VPNs [Fotedar95], the control software of such networks does not support the customised control for each VPN which is a feature of VServ. Section 1.1.2 explores this issue further.

Service level agreements (SLAs) are often used to offer resource-guaranteed VPNs on networks that don't support resource reservation. An SLA specifies the level of service agreed between the supplier and customer in terms of measurable parameters, for example latency or availability. In general, the only mechanism available to the VPN provider to meet these SLAs is over-provisioning [Woods00]. This is effective, but is not economic and

[1] An internet is a collection of interconnected networks. The Internet refers specifically to the single, worldwide internet.

does not enforce compliance by the customer, with potentially adverse consequences for charging and congestion control mechanisms.

The existing lack of support for VPNs that are both resource-assured *and* dynamic motivates the development of infrastructure for their provision. This dissertation presents a service that offers such VPNs within a single network domain.

1.1.2 Multiservice Networks

In the past, networks have been designed to cater for one particular type of usage. An example is the public switched telephone network (PSTN), which was designed specifically to transmit the human voice. It is now widely used for computer communications as well, even though its characteristics as a highly effective voice network make it quite unsuitable for transmitting data [Tanenbaum96]. The *multiservice* network has long been considered a desirable alternative [McAuley90], but has not been realised because the control and management requirements for different network services can vary considerably. This often results in unnecessary and cumbersome complexity, or else overly simple solutions that scale poorly.

A promising approach for the provision of multiservice networks is to allow each service its own customised control plane. The idea of the control plane being logically separate from the data plane allows resource management (control) functions to be treated differently to those concerned with resource usage (the forwarding of data). Examples of control plane operations are connection establishment, routing and connection admission control (CAC). Throughout this dissertation the entity responsible for this form of management is termed a *control architecture*. As will be explained further in Chapter 3, control architectures in VServ are devolved from the network hardware and can therefore be optimised to support a specific service, or can provide a generic interface to the network for applications. A well known general purpose control architecture is the ATM User-Network Interface (UNI) protocol [ATMF96a]. An example of a service-specific control architecture is Signalling System No. 7 (SS7) [ITU93], used in the PSTN.

VServ is built over a network control infrastructure that partitions resources at the lowest possible level. Combined with devolved control, this allows the network to support different services and applications, without dictating any of the out-of-band control mechanisms that must be adopted by these services. The basic framework was first proposed in [van der Merwe97a], and is introduced in Section 1.2.3 and described further in Chapter 3. Although this earlier work presents the fundamental mechanisms for achieving a multiservice network, it does not adequately address how a dynamic VPN service is deployed. For example, the specification and dynamic construction of VPNs are not considered, and neither are the requirements and constraints of such a system.

1.1.3 QoS in the Internet

The Internet is commonly regarded as a ubiquitous (and free) multiservice network. Yet user expectations of application performance are rarely met, largely due to the lack of mechanisms for guaranteeing resource, congestion control and fairness. A great deal of research is ongoing into extensions and modifications of Internet protocols to support, among other things, streaming media, multicast, traffic engineering, congestion avoidance, pricing and charging. Some of the work on guaranteeing resources, in particular when traffic is aggregated within the network, finds parallels in the management of VPN resource allocations. As [Pan00] states, there are three main challenges for the provision of QoS in the Internet:

Reservation of resources: Resources can be reserved at different granularities, from per-flow to per-VPN. Mechanisms are required to enable reservation to take place, and the reservation state must be retained at participating routers. Finer granularities of reservations permit greater flexibility, but give rise to scalability concerns.

Enforcement of existing reservations: Traffic must be policed and possibly shaped to ensure adherence to reservations. This is a data path operation, and may have an impact on forwarding or switching performance.

Inter-domain administration: The Internet is made up of a collection of *autonomous systems* (AS) which agree to forward traffic on behalf of each other. For end-to-end QoS across the Internet to be a realistic prospect, any intra-AS reservation mechanisms must interoperate, or else cooperate by means of an inter-AS reservation protocol. This is similar to the case of routing protocols, where the *interior* protocol operates independently of the *exterior* routing protocol.

In common with a number of proposals for incorporating QoS in the Internet [Nichols99, Baker00, Bernet00], VServ addresses the first point—scalability of resource reservation—by making use of resource *aggregates*. However these other proposals do not address inter-aggregate resource management issues. In a multiservice network aggregates are not necessarily homogeneous, but rather consist of different topologies and resource allocations. They may also be modified dynamically in response to changing patterns of network usage. In addition, there is the challenge of giving strong resource guarantees whilst maintaining efficient network resource usage. This dissertation directly addresses these issues as part of automated VPN provision in VServ.

1.2 Context

In this section the architectural goals for VServ are defined, and then its deployment environment is described. Finally, the underlying network control framework, the Tempest, is introduced.

1.2.1 A Dynamic VPN Service

The service offering in VServ is a VPN comprising a guaranteed allocation of resources in the network, together with the capability to manipulate those resources as desired. This functionality is supported by the Tempest, described in Section 1.2.3. The out-of-band manipulation of VPN resources is performed by a control architecture. When a customer acquires a VPN, they are able to run the control architecture of their choice over that VPN in order to use the allocated resources.

From the perspective of the customer, there is some essential functionality that should be offered by a dynamic VPN service. This includes a mechanism for the specification and acquisition of a VPN, a means of using the VPN's resources, including both data transfer and control and management, and a way of modifying a VPN. The *VPN service provider* (VSP) also has requirements for the service: it should make efficient use of available network resources and should allow the VSP to impose its own policies.

This dissertation is primarily concerned with mechanisms for providing this functionality. The design of the system is influenced by specific goals for VServ. In particular that it should be:

- usable—the interface semantics and the behaviour of the system should be easy to understand, and it should be possible to apply site-specific and global policies;
- responsive—requests pertaining to short-lived VPNs must be serviced promptly;
- flexible—dynamic resource reallocation must be supported, and it should be able to occur on a range of timescales;
- robust—the service should be fault tolerant and available.

1.2.2 Deployment Environment

The deployment environment for VServ is a single network domain or AS. Within a single domain the VSP has complete authority over resource allocation and usage within the network, although policing and shaping may be required on traffic entering from, or exiting to, external sources.

Restricting the scope of the service to a single domain does not preclude interoperability between peer VSPs. Indeed, an important feature of a commercial deployment of VServ would be the ability to establish VPNs spanning multiple network domains. This dissertation does not explore this topic in depth, but it is a worthy subject for future research.

The implementation of VServ over the Tempest network control framework is discussed throughout this dissertation. Although some implementation aspects of the underlying resource partitioning architecture are touched upon in Chapter 3, in general the low level details are not considered as they are described in detail elsewhere [van der Merwe97b, Mortier00].

The Tempest, and consequently VServ, are deployable on any network technology that can offer a connection-oriented interface for control plane operations. Currently, implementations exist using both ATM and Multiprotocol Label Switching (MPLS) over Ethernet.

1.2.3 The Tempest

Research on the devolved control of ATM switches was initiated at the University of Cambridge in a project known as Devolved Control of ATM Networks(DCAN) [Herbert94]. One of the goals of DCAN was to devolve the control and management functions of switches and other internal network devices to general purpose processors in order to open up the network and allow new services to be accommodated quickly and easily. This was in accord with other, concurrent, *open signalling* activities, described further in Section 2.3. As a result, the *Ariel* open switch control interface [van der Merwe97b] was developed.

On its own, an open interface does not solve the problem of inherently heavyweight and inflexible control software. Irrespective of where it is located, any one control architecture for a multiservice network must be able to support all conceivable network services, including those that are as yet unrealised. The general purpose nature of such a control architecture exacerbates its complexity, and may preclude the introduction of new network services that it cannot support.

The realisation that devolved control in isolation was insufficient led to the idea of the *switchlet* [van der Merwe97a]. A switchlet is a logical resource partition, and is accessed through the same open control interface as the switch itself. Using switchlets, multiple control architectures can operate simultaneously on a single physical network. A process called the *Divider* is responsible for the policing of switchlet invocations to ensure that coexisting control architectures do not interfere with each other. It is also responsible for switchlet management tasks such as creation and deletion. In-band policing and traffic shaping are used to enforce partitioning in the data path.

Switchlets on one or more adjacent switches can form a VPN. The resource allocation and topology of that VPN can be altered dynamically, by acquiring or releasing switchlets, or changing the size of an existing switchlet. A *network builder* (NetBuilder) allows such VPNs to be constructed. The entire system comprising Ariel, Dividers and the NetBuilder is known as the Open Service Support Architecture (OSSA) [van der Merwe97b], or the Tempest [van der Merwe98b, Rooney98b]. Chapter 3 describes these components and their interactions in more detail.

Other work on the Tempest is found in [Rooney98a] where issues concerning open control are explored in the context of a control architecture called the Hollowman. [Bos99] presents a framework for customisation of network control, which allows programmability of the Tempest components themselves. This dissertation is concerned with extending the Tempest to support VServ, with particular focus on the VPN creation and domain-wide resource management functions of the NetBuilder.

1.3 Contribution

This dissertation addresses the dynamic provisioning of resource-assured and programmable VPNs with a VPN service known as VServ. It shows that VServ is *feasible*, having considered its requirements and investigated potential deployment difficulties, and also *desirable*. The contribution made in the design of VServ is summarised as follows:

- A means of provisioning "better" VPNs is realised. VPNs can be acquired and modified on demand, and an exclusive resource allocation is made to each VPN.
- The facility for customised and fine-grained control for each VPN enables diverse and sophisticated network services to be supported. This network programmability, combined with guaranteed resource, gives the network genuine multiservice capability.

- The management issues that arise with the creation of resource aggregates, in particular with automated topology determination, are explicitly addressed. Problems are identified and practical solutions proposed.

- Distributed VPN creation is considered. Aspects of concurrency and distribution that are critical for dynamic VPN provisioning are explored.

- Dynamic reconfiguration of VPN resource allocations is supported in a controlled yet flexible manner with a resource revocation protocol.

VServ is developed over the Tempest network control framework which provides the support for resource partitioning and network programmability. Extensions made to the Tempest enable it to offer resource-assured and dynamic VPNs, and to fully support a multiservice network. The original Tempest implementation [van der Merwe97b] was extended to associate QoS resource types with resource partitions. Dynamic configuration of the amount of each resource allocated to a VPN was introduced, and an implementation for Internet Protocol (IP) networks was carried out to demonstrate the practicality of the system over network technologies other than ATM.

A crucial part of a dynamic and responsive VPN service is fully automated VPN provisioning. Automation of the VPN design process requires some means for customers to specify the desired VPN, the ability for the service provider to express local policies, and a procedure for determining a satisfactory VPN topology. A formalised, yet simple and flexible notation for VPN specification called VANDAL is introduced. The VANDAL specification, policies of the VSP, for example, load balancing preferences, and the current state of the system are all incorporated into the automated design process. The search for a suitable VPN topology can be expressed as a optimisation problem that is not computationally tractable except for very small networks. This dissertation describes how the search is made practical by tailoring it according to the characteristics of the desired VPN.

The feasibility of VServ is further supported by the capability for distributed VPN creation, thus improving the availability and robustness of the service. VPN design is a potentially complex operation which requires global knowledge of resource commitments throughout the network and is therefore not naturally amenable to distribution and concurrent execution. Transactional semantics for VPN creation, including support for preemptive termination of the topology search, mitigate this to some extent. The trade-offs for the VSP with respect to the advantages of distribution and its associated overheads are recognised and explored.

Finally, the dynamic resource management capabilities of VServ are exploited to give the flexibility required for automated and dynamic VPN provisioning. A resource revocation protocol is presented which allows *adaptive* control architectures to react intelligently to an impending revocation event. As a result, the VSP has a great deal of freedom in how (and when) resources are allocated to a VPN, supporting, for example, periodic resource allocations. Examples of adaptive control architectures are given to illustrate the usefulness of this approach, which would not be viable without the distinction between intra- and inter-VPN resource management which is inherent in VServ.

This dissertation investigates a feasible design for VServ, exploring in some depth issues that arise in the dynamic provisioning of resource-assured VPNs. Its contribution is twofold, including the identification of problems and concerns that arise in such a system, as well as practical proposals for its implementation and deployment.

1.4 Outline

In Chapter 2 background on related networking issues is reviewed. An overview of existing types of VPN is presented, and then important approaches to the provision of QoS in networks are covered. The concepts of open signalling and devolved control are also described.

Chapter 3 describes the Tempest and the extensions made to the original version in order to deploy a dynamic VPN service. The implementation over MPLS is also discussed.

The process of automated design and deployment of VPNs is covered in Chapters 4 and 5. The process includes the parsing of a VPN specification language and a procedure for finding a topology and resource allocation that satisfies the requirements of both service provider and customer. Customisation of the design process in order to make it computationally tractable is described.

Chapter 6 focuses on the replication of the NetBuilder to improve the availability of VServ. A protocol for distributed VPN creation is proposed, and the associated trade-offs for complexity and performance are explored.

Resource revocation is discussed in Chapter 7 with the description of a simple revocation protocol. The ways in which different types of control architecture can use the protocol for adaptation in advance of a revocation event are described.

A summary, further work and concluding remarks are in Chapter 8.

Related work is reviewed throughout as appropriate.

Chapter 2

Background

This chapter describes other work on VPNs and network QoS and highlights the differences to VServ. It also includes background material relevant for work presented in later chapters.

There are a number of different definitions of the term *Virtual Private Network* in common use. Section 2.1 reviews the various VPN service offerings currently available, and clarifies the meaning used in this dissertation.

Many VPN services rely on support for QoS in the underlying network. Likewise, VServ assumes a connection-oriented network with per-connection QoS capabilities. Two such network technologies, ATM and MPLS, are described in Sections 2.2.1 and 2.2.2 respectively.

Attempts to introduce QoS support in IP networks are presented in Section 2.2.3 which describes the Integrated Services architecture, and in Section 2.2.4, which discusses the Differentiated Services architecture. Section 2.2.5 covers endpoint admission control—a hybrid of integrated and differentiated services. These proposals are of particular interest because they attempt to construct resource-assured VPNs, and therefore encounter similar global resource allocation issues as those tackled in this dissertation.

A unique point of the Tempest, and hence of VServ, is that the signalling and other out-of-band control functionality can be carried out by bespoke software. The owner of the VPN—the customer who has acquired the VPN from the VPN service provider—is free to run the control architecture of their choice over the allocated set of resources. Section 2.3 reviews the concepts of open signalling and devolved control that are used to achieved this.

2.1 Virtual Private Networks

Virtual Private Networks (VPNs) provide dedicated connectivity to a closed group of users on a shared network. The customers of a VPN service do not need to operate their own network, and are therefore relieved of the need for maintenance and upgrades. VPNs are more flexible than fixed networks: topologies can be customised, network robustness and QoS can be contractually ensured, and dynamic usage patterns can be supported with a usage-based charging structure. [Hall94] identifies other features of VPNs such as security, private addressing and flexible control mechanisms (for example, call screening or hotline access in a telecommunications context). In addition, because multiple VPNs may share the physical network, VPNs can allow the network operator to achieve better utilisation of resources.

VPNs were first introduced to a wide customer base in the early 1980s [Hall94], typically offering wide-area connectivity to corporations. Prior to this, dedicated leased line services were common, but required either over-provisioning of resources or else suffered from insufficient capacity to meet peak demands, as well as inflexibility and high configuration overheads. Early VPN services[1] were deployed on broadband networks using permanent virtual circuits (PVCs) over ATM or Frame Relay.

These types of VPN are still in common use, and while they don't suffer from the over-provisioning that occurs with leased lines, they do have similar problems of inflexibility and high configuration overheads. Access to network control and management functions is restricted, so configuration changes requested by the customer must be performed by the service provider. This level of indirection causes what is a very simple operation at the level of the switch or router to take a disproportionate amount of time and human involvement. The VPN customer does not have fine-grained control over their resource allocation—it can only be used in a predetermined manner. A simple example is a VPN comprising two PVCs, each with a guaranteed constant rate of throughput. The *usage* of these resources is intimately tied to the nature of the *allocation*—the customer cannot transfer some bandwidth from one PVC to another (without undue overheads). In contrast, VServ explicitly divorces resource usage from allocation, allowing the customer to manipulate their set of resources directly and in a manner of their choosing.

More recently, the provision of VPNs on IP networks [Gleeson00], in particular the Internet, has become common. Tunnelling protocols, such as Point to Point Tunnelling Protocol (PPTP) [Hamzeh99] and Layer Two Tunnelling Protocol (L2TP) [Townsley99]

[1] Although widespread, VPNs on voice-only networks are not considered here, as this work is concerned with data, or multiservice, networks.

encapsulate the packet at the ingress to the network and then route it across the intervening network to the tunnel egress point, where the encapsulation header is stripped. PPTP enables the Point to Point Protocol (PPP), designed to send data across dial-up connections or serial lines, to be tunnelled through an IP network. L2TP is based on and extends PPTP, notably in being able to operate over non-IP point-to-point networks such as ATM and Frame Relay.

The use of public IP networks may warrant additional security measures for the traffic of a tunnelled VPN. IPSec [Kent98], which encrypts the payload and adds an authentication header verifying the source of the packet, is often used in conjunction with tunnelling protocols to ensure data security and integrity.

Tunnelled VPNs have the advantage that addresses in the original packets are hidden from public routers, so a private addressing scheme can be used in the VPN. They also benefit from not requiring centralised management and supporting flexible and dynamic topologies. Thus a mobile host or occasional VPN member, for example a company employee working from home, needs only standard Internet access, together with the appropriate protocol stack software and access permissions, to become part of an existing VPN. However, tunnelled VPNs have limited support for QoS, due to the underlying best effort nature of IP networks.

Irrespective of the ability of the underlying network technology to explicitly support QoS, service providers can offer higher-level QoS guarantees using network management techniques which are often described to customers in the form of service level agreements (SLAs). For example, within its local autonomous system an ISP can use traffic engineering techniques or over-provisioning to enable SLAs to be met. However SLA establishment across other domains requires negotiation and trust between all the parties concerned. There is currently no standardized interface for inter-VPN QoS negotiation, although [Fox99] defines a globally unique VPN identifier that might play a part in such an agreement.

Further disadvantages to the use of SLAs are the difficulties in their construction and assessment. The content of an SLA usually covers network QoS, e.g. data rate and latency, as well as contractual matters such as charging and compensation in the event of non-compliance with the agreement. There are significant challenges involved: it is very difficult to map a customer's perception of what they want from their VPN to a set of specific QoS parameters, and monitoring for compliance can be complex and burdensome. Furthermore, SLAs often require negotiation by networking and legal experts over a considerable length of time, and are inappropriate for a highly dynamic VPN service.

VServ is intended to operate within a single domain, with all internal network elements under the control of the VSP. *Virtual LANs* (VLANs), first proposed in [Martillo90], adopt a similar approach for the switched local area network (LAN). VLANs logically divide the physical LAN into different subnets, limiting the propagation of multicast and broadcast traffic. The idea is that users with a common interest can reside on the same VLAN, isolated from the users of other VLANs. In the IEEE 802.1Q standard [IEEE98a], membership of a VLAN is defined by port or by Media Access Control (MAC) address. More sophisticated switches can filter packets into VLANs based on header information such as protocol type or IP subnet address. IEEE 802.1Q also defines a VLAN identifier for tagging packets. Standard VLAN identifiers enable interoperability of VLANs across different media and different proprietary VLAN systems, so, for example, on a campus network geographically dispersed users on separate physical LANs can be connected with the same VLAN. Traffic between VLANs must be routed, but a VLAN allows the fast switching capabilities of the hardware to be fully exploited for its internal traffic.

Along with the VLAN identifier, IEEE 802.1Q defines a 3 bit user priority field, which provides rudimentary support for QoS in the form of values encoded in packet headers. Flows can therefore be differentiated in the network by associating different priority queues at the switches with different traffic classes. This mechanism is used to define support for Integrated Services (described below in Section 2.2.3) over switched LANs [Seaman00]. A set of default mappings from delay targets to user priority values is proposed. For example, user priority 0 maps to best effort, and user priority 6 maps to "delay sensitive, 10ms bound."

VLANs require support in every switch, and usually need manual set up and maintenance. Proposals for automating this process include dynamically adding a port to a VLAN when an IP datagram is received from it containing the subnet address allocated to that VLAN, using traffic observations obtained by on-line measurement to trigger VLAN creation or removal [Rooney99], and GARP VLAN Registration Protocol (GVRP) [IEEE98a]. GVRP is an application of the Generic Attributes Registration Protocol (GARP) [IEEE98b], which supports dynamic registration by stations for membership of a VLAN. These techniques are generally not amenable to the out-of-band VPN management used in VServ, and in addition do not support resource partitioning or reservation which is essential for provisioning resource-assured VPNs.

The most common VPN applications have been identified as [Lucent99]:

- Remote access for mobile or work-at-home employees.
- Intranets: IP networks within companies, often using VPNs to connect remote sites.
- Extranets: networks allowing limited access to an intranet for outside parties, for

example customers or suppliers.

- Multi-user applications such as on-line meetings, or multicast seminars.

It is notable that only the last of these applications has any stringent real-time QoS demands. A VPN on a multiservice network should be usable by a very diverse set of applications. However, to fully exploit the capabilities of such a network, individual VPNs need to be able to control their resource allocation in a manner suited to the service they are offering. VServ has this facility for customisation, which together with guaranteed resource allocations, enables it to support legacy and standardized signalling systems and network protocols alongside specialised *service-specific* control software. This feature enables VServ to uniquely offer a truly multiservice VPN service.

2.2 Quality of Service in Networks

The ability to partition resources inside a network generally relies upon there being some way of attributing resource usage to the source of the traffic responsible. A connection-oriented network does this by associating resources with a connection, and policing the amount of traffic that is carried by that connection. Implementations of the Tempest make use of this facility, together with in-band QoS capabilities such as shaping, dropping and scheduling. In the following sections, two such network technologies for which implementations of the Tempest exist are described. Section 2.2.1 describes ATM—essentially a link layer technology—and Section 2.2.2 discusses MPLS, which can operate over various link layers, including ATM.

The ubiquity of the Internet has prompted much investigation into support for QoS, and hence provision of a multiservice network, with IP. Three approaches are described in Sections 2.2.3, 2.2.4 and 2.2.5. Not only are these proposals prompting interest in explicit provision of resource-assured VPNs over the Internet [Gleeson00], but they themselves are faced with some of the global resource allocation issues that are the topic of this dissertation.

2.2.1 Asynchronous Transfer Mode

Asynchronous Transfer Mode (ATM) is a connection-oriented network technology that supports both constant and variable rate traffic. It uses small, fixed-size cells which can be switched at very high speeds. The switching is based on an identifier pair in the cell header consisting of *Virtual Path Identifier* (VPI) and *Virtual Channel Identifier* (VCI). These labels have only local significance and are remapped at each switch as necessary. The VPI and/or VCI refers to the *virtual circuit* (VC) or connection over which the cell is being transmitted, with a virtual path being made up of a bundle of virtual channels.

ATM connections are either semi-permanent, known as *Permanent Virtual Circuits* (PVCs), or else are created on demand and exist only for the duration of the call—*Switched Virtual Circuits* (SVCs). Each ATM connection is associated with an allocation of resources, consisting of VCI and/or VPI identifiers, bandwidth and other physical resources in the switch such as buffer space. The resource allocation is usually made when the connection is established (bandwidth may be statistically multiplexed in the case of variable rate connections).

The set up and release of connections is performed out-of-band, i.e. on a separate, dedicated VC, using a signalling protocol. An in-band *Usage Parameter Control* (UPC) function enforces compliance of a connection's traffic with its negotiated traffic contract. The UPC function of a switch monitors and shapes incoming cells to ensure that a misbehaving source does not adversely affect the QoS guarantees of other connections.

The resources associated with a connection provide it with a certain QoS, which can be defined in terms of attributes such as cell loss, delay and delay variation. The ATM Forum has standardized five service categories, which attempt to encompass the most common kinds of traffic envisioned on ATM networks. The categories are defined as follows [de Prycker95].

Constant Bit Rate (CBR): Provide tight bounds on delay and delay variation, together with a constant throughput. Typically used by voice and circuit emulation traffic.

Real-Time Variable Bit Rate (RT-VBR): Suitable for applications with stringent real-time requirements, i.e. low delay, and delay variation, with variable rate throughput. Compressed video and audio would usually make use of this service.

Non-Real-Time Variable Bit Rate (NRT-VBR): Intended for applications which generate bursty traffic and require bounded delay, but can tolerate some delay variation. An example is multimedia email.

Available Bit Rate (ABR): Suitable for sources that can modify their transfer rate according to current bandwidth availability. ABR provides rate feedback from the network to the sender.

Unspecified Bit Rate (UBR): Best effort transmission, cells will be discarded if there is insufficient capacity.

The Tempest Ariel interface makes use of the ATM service categories as a convenient abstraction for defining the QoS capabilities of a switch in a generic manner [van der Merwe97b].

ATM Adaptation Layers (AALs) provide a mapping between the ATM layer and the next higher layer. An AAL segments protocol data units (PDUs) into cells and then reassembles the PDUs after transmission. Different AALs have been defined to provide various classes of service using functions such as manipulating bit rate, limiting cell delay variation, handling lost or misdelivered cells and so on.

ATM was designed as a multiservice network—a single universal network that can support all types of service and thus be efficient, cost-effective and flexible. It was standardized by the International Telecommunications Union (ITU) in the early 1990s, and the ATM Forum has standardized the *User-Network Interface* (UNI) [ATMF96a] and the *Network-Network Interface* (NNI) [ATMF96b] as signalling protocols to ensure interoperability between the equipment of different vendors. UNI signalling is used between an end-system and a switch, and NNI between elements within the network. The aim was to achieve a world-wide Broadband Integrated Services Digital Network (BISDN) [de Prycker95].

In spite of, or perhaps due to, these standardization efforts, and also its high cost and relative complexity, ATM is commonly deployed only in the network backbone. It is not well suited as a public access network, since, for example, UNI is not appropriate for setting up SVCs across public networks [Alles95]. Nevertheless, there are advantages in the connection-oriented model for the provision of network QoS, in particular that fine-grained resource guarantees can give flexibility while maintaining good utilisation. MPLS, described in the following section, adopts this model at a higher network level, hence providing a way of associating resources with connections even over connectionless network technologies such as Ethernet.

2.2.2 Multiprotocol Label Switching

Multiprotocol Label Switching (MPLS) [Callon99, Rosen99] is a means of packet forwarding that selects the next hop according to a short, fixed length label in the packet's header. It operates over various link layer technologies, and divorces route determination functionality from the forwarding mechanism, thus enabling high-speed switching in the network core and providing a way of introducing traffic engineering functionality, such as load balancing and QoS routing.

When a packet first enters the network, it is classified by the ingress router into a Forwarding Equivalence Class (FEC). The appropriate label is then inserted into a spare field in the network or data link layer header, or else the packet is encapsulated with a special-purpose (shim) header. Within the network core the packet is forwarded at the link level according to its label and no further analysis of the packet's header is necessary, allowing fast and simple forwarding. This also means that new routing services can be introduced without any change to how forwarding takes place. At label switching routers (LSRs), the label serves as as an index into a table containing the next hop and a new label which replaces the old in the packet's header. The route taken by a packet through the network is known as a label switched path (LSP).

A label distribution protocol (LDP) is used to advertise assigned labels, and hence establish LSPs, throughout the network. There are several variations of LDP which may coexist [Callon99]. In *topology driven* LDPs [Andersson00], label distribution occurs as part of the processing of conventional routing protocol traffic. *Request driven* LDPs allocate labels according to control protocols such as Resource ReSerVation Protocol (RSVP) [Zhang93]. *Traffic driven* LDPs are triggered by the arrival of data, for example, IP switching [Newman98b] in which the observance of an IP flow prompts the establishment of a connection to carry the constituent traffic. The different LDPs support different granularities for allocating packets to FEC, and there are trade-offs between consumption of labels, label setup latency, and support for per-flow QoS and fine-grained traffic engineering capabilities.

The shim header contains a 20-bit label carried after the data link layer header. Multiple label encapsulations (stacks) can exist between data link layer header and network layer header, and this is indicated by a stack bit in the shim header. Label stacking allows several individual streams to be forwarded under a single label within one context, but still be switched separately later within a different context.

The Internet Engineering Task Force (IETF) MPLS Working Group is currently in the process of standardizing MPLS in order to support its implementation over various link-layer technologies. The objectives of the standardization process include defining MPLS packet header formats and LDPs, and discussing support for QoS.

MPLS provides a platform for building resource-assured VPNs. However, on its own it is not sufficient for a multiservice network. In addition to resources for packet forwarding, such as buffer space and bandwidth, the provision of resource-assured VPNs requires partitioning of the label space. The lack of this facility in MPLS limits the way resource may be guaranteed to each VPN, and the freedom afforded a control architecture to manipulate its resource allocation. A further problem is that inter-VPN resource management is not supported: since a VPN's resource allocation as a whole is only expressible as the sum of the resource allocations of its LSPs, resources cannot be guaranteed for the future, other than by pinning down a path—in effect associating each label with a predetermined QoS. The capability for both inter- and intra-VPN resource management, as provided by the Tempest, is an essential part of VServ that is not inherently supported by MPLS.

2.2.3 Integrated Services

The Integrated Services architecture (IntServ) [Braden94] is an attempt to introduce QoS support into the Internet to enable it to act as a multiservice network with resource guarantees. As has already been discussed, significant benefits are to be gained from integrating telephony, multimedia and other services into a single network. These advantages include economies of scale, ubiquity of access and improved statistical multiplexing [Clark92]. An additional objective of IntServ is to enable controlled link sharing—the division of link bandwidth between traffic classes by the network operator.

A number of service classes have been proposed for IntServ that provide data flows with certain QoS. Of these, two have been formally specified by the IETF [White97]: *guaranteed service* [Shenker97] and *controlled-load service* [Wroclawski97]. Guaranteed service gives bounded end-to-end delay, assured bandwidth and no queueing loss for conforming packets, and is suitable for applications with stringent real-time requirements. In contrast, controlled-load is appropriate for applications that are highly sensitive to overloaded conditions, but can tolerate a certain amount of delay and loss. Controlled-load service uses admission control to ensure that a data flow always receives approximately the same QoS. As the network load increases, service to the flow, i.e. its delay and loss, does not deteriorate, but always approximates the service that would be received as a best effort flow on a lightly loaded network.

In addition to service classes, IntServ includes a mechanism for communicating the requirements of the application to the network and for establishing the corresponding flow-specific state within the network. Although this task can also be carried out in other ways, for example using network management procedures, the IETF has defined a resource reservation protocol called RSVP [Zhang93, Braden97]. This is a signalling protocol in which the receiver makes a resource reservation request, which is forwarded along the reverse path to the data source (or to where the receiver's path joins a multicast distribution tree). Resources are reserved for that flow at each network element along the way, provided admission control and policy constraints are satisfied. The per-flow state thus established is maintained by periodic refresh messages to support dynamic multicast group membership and automatic routing changes.

It has been widely observed that there are significant scalability problems for RSVP

[Berson98, Blake98, Stoica99, Baker00]. Each network element must keep per-flow state and process per-flow reservation requests. Even a small reservation entails the transmission of a number of reservation protocol messages, as well as using computation and memory resources within network nodes. The aggregation of reservations has been considered [Baker00], where a number of flows that share ingress and egress routers are aggregated into a single reservation between those routers. Within the aggregation region some mechanism for traffic classification is presumed, such that all the traffic included in a single resource reservation aggregation is assigned to the same traffic class and forwarded accordingly. An example of such a mechanism is DiffServ, described in the next section.

2.2.4 Differentiated Services

The Differentiated Services (DiffServ) architecture [Blake98, Nichols99] is another approach to providing QoS in the Internet that was designed in response to the perceived lack of scalability of IntServ. Rather than giving per-flow QoS, DiffServ assigns each packet to a *behaviour aggregate*, according to the service it requires. Throughout a DiffServ domain, routers apply different *per-hop forwarding behaviours* (PHBs) to the different behaviour aggregates, i.e. classes of traffic.

PHBs are implemented with buffer management and packet scheduling mechanisms. The allocation of buffer and bandwidth resources to a behaviour aggregate defines the forwarding treatment that it receives. Standardized PHBs include Assured Forwarding [Heinanen99] and Expedited Forwarding [Jacobson99], and these are defined in terms of behaviour characteristics rather than in terms of a particular implementation. Assured Forwarding gives four different levels of forwarding assurance, and within each level the packets can be marked by one of three possible drop precedence values. The probability that a packet is forwarded depends upon the amount of resource allocated to its forwarding assurance class, together with the current load in that class and, in the case of congestion, the drop precedence of the packet. Expedited Forwarding is a form of "premium service", which gives low loss, latency and jitter, together with guaranteed bandwidth. This is achieved by eliminating queueing for packets in that aggregate, in other words ensuring that the departure rate of an aggregate's packets from a node equals or exceeds a certain configurable arrival rate.

At the ingress node of the DiffServ domain traffic is classified, possibly conditioned (i.e. metered, marked, dropped or shaped) and then assigned to a behaviour aggregate. Traffic is forwarded through the domain according to its *codepoint* [Nichols98], which identifies the behaviour aggregate to which it belongs. The complex classification and traffic conditioning functions are only performed at boundary nodes, so in contrast to IntServ there is no need for per-flow state to be maintained in the network. Applying PHBs to aggregates gives a coarse-grained allocation of buffer and bandwidth resources among competing traffic streams.

The partitioning of resources within a DiffServ domain between different behaviour aggregates has parallels with the allocation of resources to resource-guaranteed VPNs. The service provider performs the partitioning according to some policy, for example based on SLAs specifying the forwarding service the customer should receive. The administration of these aggregate resource allocations is ideally done dynamically and automatically, in response to changes in network load and usage patterns. Agents known as *Bandwidth Brokers* (BBs) have been proposed to fulfill this function [Nichols99]. BBs keep track of the current allocations within their DiffServ domain, handle requests for new allocations, and interact with peers to configure service classes that cross domain boundaries.

The key difference between resource management for VPNs in VServ and for behaviour aggregates in DiffServ and is that VServ guarantees resources not only to the aggregate, but also to individual flows within the aggregate. In DiffServ individual flows can affect the QoS received by others within the same behaviour aggregate if the load in the service class is allowed to go too high. This means that either the BB has to be aware of the load on a service class, perhaps by some feedback mechanism, and respond accordingly to maintain the service, or else the users of a behaviour aggregate must perform traffic policing outside the DiffServ domain in order to obtain fine-grained control of resources. Moreover, because traffic policing at the edge routers is done without regard to the route taken by that traffic through the network, transient overload situations may occur at points within the network. This may violate the service agreement, and, if the BB becomes aware of the overload, may also result in oscillation of the resource allocation to that behaviour aggregate.

2.2.5 Endpoint Admission Control

Endpoint admission control [Elek00, Cetinkaya00] is an attempt to combine the advantages of the per-flow QoS that can be obtained with IntServ with the superior scalability of DiffServ. It works by probing the network at the requested rate, and only admitting a flow if the probe experiences the desired quality. Thus routers do not maintain per-flow state, and there are no reservation overheads. DiffServ extensions within the core can provide different priority levels for admitted traffic, usually just two—a best effort service and a "better than best effort" service. Due to the inherent imprecision of network measurements, endpoint admission control only offers a soft real-time service, similar to the controlled-load service defined for IntServ.

The types of service supported by endpoint admission control each receive a resource partition. With this scheme, the potential for contention between flows for a share of resources within a partition is diminished by applying admission control based on actual network use. However the need for higher-level resource management in order to fairly assign a share of the domain's resources to each service class remains.

2.3 Open Signalling and Devolved Control

Both IntServ and DiffServ employ some form of resource partitioning or reservation for the provision of QoS in the network. Likewise, the Tempest architecture partitions resources, but also adopts an *open signalling* approach, which is the key to enabling customised control architectures for VPNs.

The idea of open signalling was formed in a telecommunications context, where control and management operations are separate from the data path. It was felt that the traditional "closed" model for service provision, in which the equipment vendor has complete control over all hardware and software in the network, has a number of serious flaws [Lazar97]:

- it hinders the rapid deployment of new and novel services;
- it assumes that endpoints (Customer Premises Equipment) are dumb—this is no longer generally the case for data networks, yet the processing capabilities of endpoints are not capitalised upon;
- it impedes portability and upgrades, as service software tends to be highly dependent on the hardware;
- it creates complexity and can result in unanticipated interactions between services.

Open signalling *devolves* control away from the network elements by defining open programmable interfaces which allow third parties to introduce services into the network. An example of an open signalling solution is IP Switching [Newman98b] for IP on ATM, in which the standard ATM signalling software is discarded in favour of the General Switch Management Protocol (GSMP) [Newman97], which acts as an open, programmable interface for the ATM switch. IP packets are forwarded on a hop-by-hop basis over a default VC, unless a flow is identified by the IP switch controller, in which case a new VC is established using GSMP and constituent packets are switched directly in hardware. Soft state is maintained for the switched connections, which are periodically refreshed provided traffic belonging to the IP flow is being received.

MPLS may also be considered the basis for an open signalling approach. Within the network, data is forwarded simply by label switching, and control and management operations, such as routing and establishment of LSPs, are performed out-of-band. However MPLS in the form proposed by the IETF [Rosen99] does not support an open, programmable interface for setting up LSPs, instead requiring the use of either a pre-defined LDP or else manual intervention.

By opening up the network, open signalling supports service-specific and nonstandard control software. The Tempest combines this facility for customised control with guaranteed resource allocations, thus giving a flexible and powerful framework for support of a VPN service.

2.4 Conclusion

This chapter described the various types of VPN that exist, and compared each with the types supported by the Tempest. It explained that no other VPN service offers both a guaranteed resource allocation *and* the facility to manipulate those resources as desired. VServ opens up the multiservice capabilities of a network to exploitation—in a controlled manner—by third parties.

Approaches to QoS provision in networks were described. ATM and MPLS, two connection-oriented technologies which underly the Tempest implementations reported in the next chapter, were presented. Efforts to introduce QoS support in IP networks were described, and parallels drawn between the allocation of resources to traffic classes and the allocation of resources to VPNs.

Finally, open signalling and devolved control were discussed. These ideas enable control architectures of the VPNs offered by VServ to exert customised and fine-grained control over their allocated resources.

Although this dissertation is concerned with the particular problems that arise in the provision of a VPN service over the Tempest, this chapter highlighted the more general applicability of many of the issues. The following chapter continues with more detailed background on the Tempest network control framework, before describing the extensions made to enhance its support for QoS.

Chapter 3

The Tempest

The Tempest is a network control framework that was developed to address perceived problems of ATM networks. This chapter gives a brief overview of the motivation behind the original proposal and the resulting architectural design. Features added to both the implementation and the design to support VServ are then described and some related work is reviewed. Detailed descriptions of earlier work on the Tempest can be found elsewhere [van der Merwe97b, Rooney98a, Bos99].

3.1 Motivation

Until recently, the management and control functionality of networks was tightly coupled to the network hardware, leading to a 'closed' and restrictive environment. The provision of new network services, signalling protocols and other software was exclusively the domain of network equipment manufacturers, often under the auspices of international standards bodies. The drawbacks of this approach are widely recognised:

- Although the use of standards ensures interoperability, the process of defining them is a slow and often highly political exercise. It is not unusual that by the time a standard is produced, technological developments have made it largely obsolete.
- Standards are often monolithic, resulting in heavyweight and unwieldy control mechanisms, further stifling the impetus to develop and quickly deploy new network services.

- Ways of introducing innovative network services are invariably very limited. If the demand has not already been anticipated, it is unlikely the network will be able to cater for the service in an optimal manner, if at all.
- Independent of whether standards are deployed in the network, close integration of control software and the internal network elements makes upgrades and bug fixes difficult and costly.

Many instances of these shortcomings can be observed in commonly deployed networks. Examples include the awkward integration of intelligent network services into the telephone network, and the requirement for some ten thousand lines of code on the switch for UNI/NNI signalling in ATM networks.

Open signalling is a partial solution to these problems. The functionality of the switch or router is encapsulated in an open control interface which can be accessed by third parties, who may not necessarily be either the user of the application or network service, or the manufacturer of the hardware. Typical operations available through such an interface include connection management, routing, alarm notification and the gathering of statistics. GSMP [Newman98a] and Virtual Switch Interface (VSI) [Buckley98] are examples of open switch control interfaces. The Tempest employs an interface developed in the Computer Laboratory called Ariel [Rooney98b, van der Merwe98b].

Open signalling is a useful mechanism for the deployment of lightweight, non-standardized network control protocols, for easy maintenance and rapid upgrade of the control software, and for divorcing that software from the internal network hardware. However, it does not by itself provide a satisfactory solution to the problem of inflexible and monolithic control software. A single control architecture on a multiservice network is either too complex or too restrictive when used to support an arbitrary set of services. Operations that suit one type of control may not be suitable for another, and as services evolve new kinds of operation may be required.

The approach adopted in the Tempest is to partition the physical network by subdividing the resources of individual switches into one or more logically separate *switchlets*. Each of these is presented to its owning control architecture through an open switch control interface, which gives the illusion that the control architecture is managing an entire switch. This is the crucial feature that allows multiple control architectures to coexist on a single physical network, whilst also giving them fine-grained control of the resources they have been allocated.

A set of switchlets, or resource partitions of physical nodes, can be combined to form a VPN which then has effectively exclusive access to its share of the network resources.

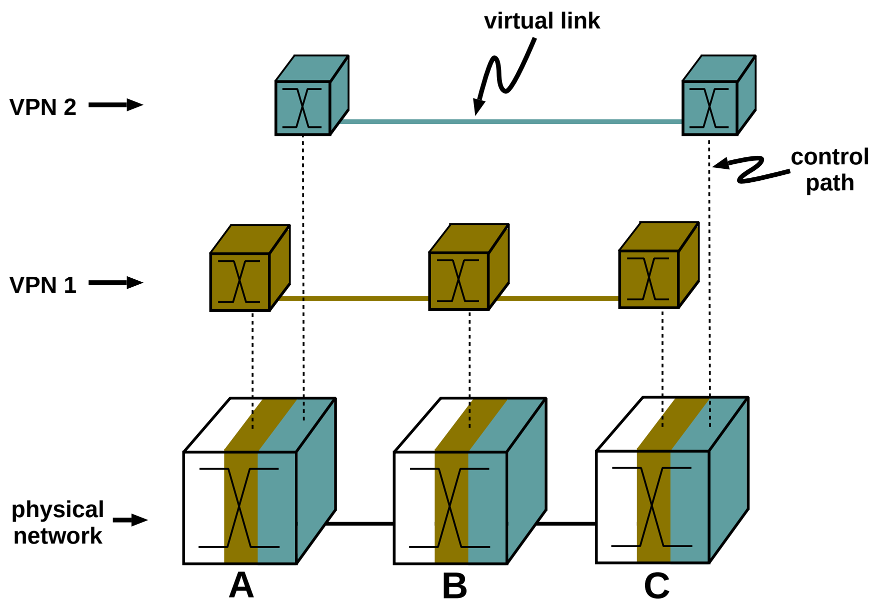


Figure 3.1: Two VPNs with different virtual topologies.

This is illustrated in Figure 3.1, which shows two VPNs coexisting on a physical network comprising three switches. Each VPN has a different topology, and control software for each VPN sees only the resource allocation and control interface exported by its switchlets. Note that VPN 2 has a different *virtual* topology to the underlying physical topology—a resource allocation for that VPN is still made at the intervening switch `B`, but the VPN control software is unaware of this as no control interface is exported at that node.

The provision for multiple control architectures to operate simultaneously means that no single system need be prescribed for all users (which is not to say that multiple instances of the same control architecture cannot coexist). Although for many users standard, general purpose control architectures will suffice, others are able to run service-specific control architectures tailored to their individual needs, if they wish to do so.

3.2 Architecture

The creation, deletion and modification of switchlets is managed in the Tempest by the *Divider*. One Divider operates for each switch in the network. The Divider is external to the network hardware, possibly running on a general purpose workstation, ideally in a resource-controlled environment such as that provided by the Nemesis operating system [Leslie96].

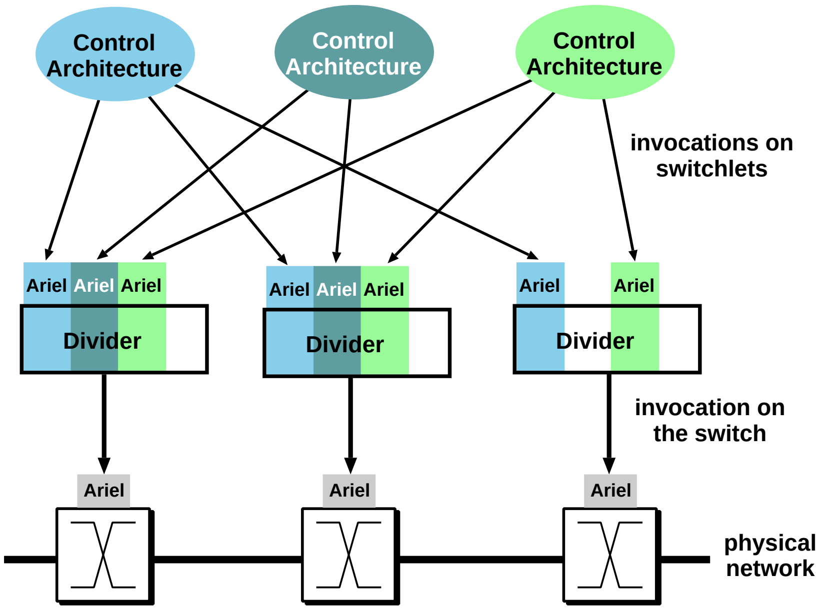


Figure 3.2: Multiple control architectures operating simultaneously.

It may also operate on the physical node itself to improve performance, but the hardware and the partitioning functionality of the Divider remain logically distinct.

Invocations on switchlet interfaces are policed on the control path by the Divider to ensure there is no interference between control architectures. Figure 3.2 shows multiple control architectures sharing the resources of the physical network. Each control architecture makes invocations on the Ariel interface exported by the Divider for the corresponding resource allocation on that switch.

A full-blown VPN is constructed by acquiring switchlets in one or more network nodes. The *NetBuilder* is responsible for the creation and deletion of VPNs, and for allocating the resources required by those VPNs. The topology of a VPN need not map directly onto the underlying physical network. A single link VPN can span multiple physical links simply by reserving resources on all intervening nodes without exporting Ariel interfaces—such resource reservations are termed *tunnel switchlets*.

Figure 3.3 illustrates the steps involved in the acquisition of a VPN: the customer sends a description of the required VPN to its control architecture (1), which passes on a formal specification of the topology and QoS requirements to the NetBuilder (2). The NetBuilder determines the switchlet specifications that are appropriate for that network—a process termed *VPN design*, discussed further in Chapters 4 and 5. The NetBuilder then creates a

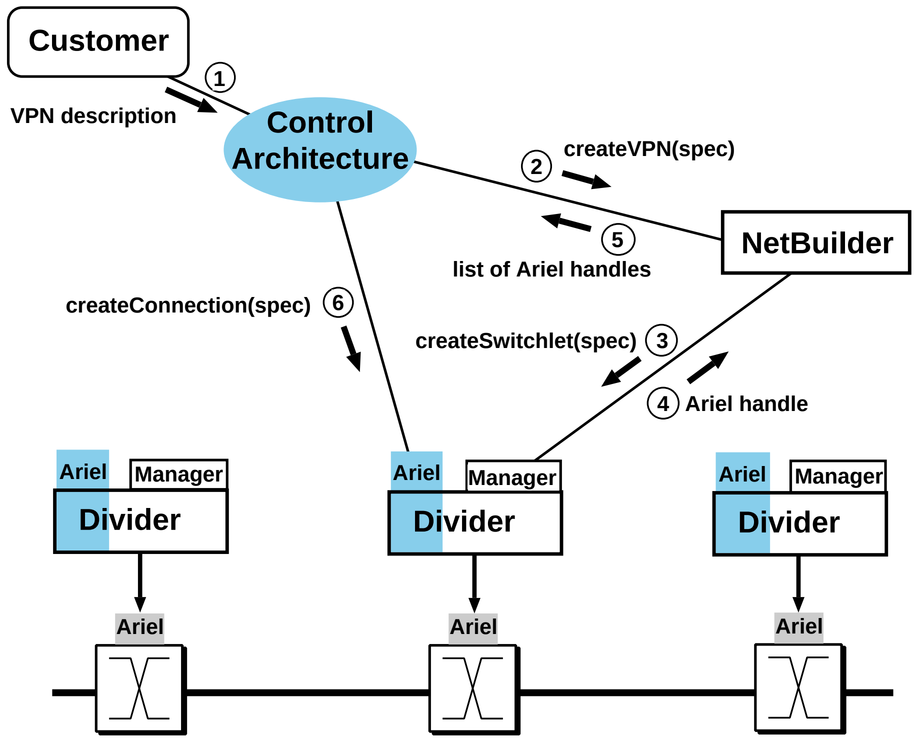


Figure 3.3: Acquisition of a VPN.

VPN comprising those resources by locating the Divider at each of the constituent nodes, and requesting a new switchlet at each (3). After creating a switchlet, the Divider returns a handle for the new Ariel interface for that switchlet to the NetBuilder (4). The handle provides a means of accessing the operations of the interface—the next section describes the handles used in the Tempest, where they are also known as *interface references*. The switchlet handles are then passed back to the control architecture (5), which can then make invocations on switchlets directly (6) and hence control its partition of the physical network resources.

Although the Tempest framework provides the means to acquire a VPN, the basic functionality does not include VPN design. This dissertation considers the process of VPN design in the context of a dynamic VPN service running over the Tempest.

3.2.1 Component Communication

The Tempest makes use of the facilities of a *Distributed Processing Environment* (DPE) for communication between Tempest components [van der Merwe97b, Rooney98a]. Although implementations of the Tempest have been made with several DPEs, a Common Object Request Broker Architecture (CORBA) [OMG95, OMG00] DPE was eventually solely adopted due to its ubiquity together with the availability of free implementations with support for a range of programming languages. Hence client-server interaction semantics are defined using CORBA Interface Definition Language (IDL), and the transport facilities are provided by omniORB [Lo00].

In the CORBA specification, *objects* provide services to clients through interfaces that define the set of operations that a client may request of that object. A particular object is specified within the Object Request Broker (ORB) by its *object reference*, also known as its *interface reference*. Once a client has been handed an interface reference, it is able to make requests to the corresponding object. Entities such as *traders* may pass interface references to clients. The Tempest currently implements trading by storing the interface references of its components as files in well known locations. When a new object is created, for example when a switchlet is constructed in response to a request from a control architecture, an interface reference for that object must be handed back to the requester. In the case of a new switchlet, an interface reference for an Ariel object is handed back to the control architecture.

When deployed, the Tempest components interact within a dedicated *control VPN*. The resource allocation guaranteed to that VPN ensures that traffic belonging to customer VPNs does not compromise the network service given to control and management operations.

A bootstrap VPN is employed at start of day. This is described further in Section 3.4.

3.2.2 Granularities of Resource Management

Within the Tempest two different granularities of resource management are distinguished:

1. **Intra-VPN** resource management is the responsibility of the control architecture. Within a VPN's resource allocation, the control architecture performs connection establishment and teardown, and the association of resources with connections. A connection is always associated with at least one *label*, and may also be given QoS in the form of resources such as bandwidth or buffering. The resource partitioning of the Tempest ensures that a control architecture cannot exceed its resource allocation when it sets up or modifies connections. The in-band policing that is employed ensures that the data sent along those connections is within the established traffic contracts.

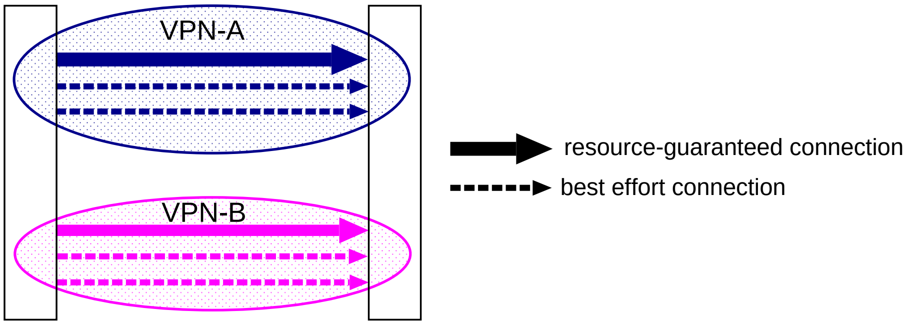


Figure 3.4: Experimental setup for the demonstration of inter- and intra-VPN resource management.

2. **Inter-VPN** resource management is the responsibility of the NetBuilder. A VPN comprises a set of switchlets, in other words, a resource *aggregate* distributed across the network. The NetBuilder allocates resources to (and removes resources from) VPNs by modifying switchlet specifications at Dividers.

Figure 3.4 illustrates the experimental setup for a demonstration of the inter- and intra-VPN resource management capabilities of the system. The figure shows a single link, on which two VPNs each have a resource allocation. Both VPNs have established three connections within their resource allocation, consisting of one bandwidth-guaranteed connection and two best effort connections. The total bandwidth usage of the three connections in each VPN comprises the aggregate bandwidth allocation to that VPN.

The graph of Figure 3.5, taken from [Mortier00], contains the results. It shows the User Datagram Protocol (UDP) throughput of the 6 flows sampled over 500ms. Two resource reallocation events take place in the course of the experiment. The first, at 20 seconds, involves an internal VPN resource reallocation: within VPN-A, the bandwidth to the guaranteed connection is increased by 10Mbps. As expected, the best effort flows of VPN-A correspondingly reduce by 5Mbps each, whilst the flows of VPN-B are unaffected.

The second reconfiguration, which occurs at 40s, demonstrates an increase in the overall resource allocation of VPN-B from 10Mbps to 30Mbps. Now each best effort connection of VPN-B receives around 10Mbps more. The allocation to VPN-A is not changed, nor is the service received by VPN-B's guaranteed flow. This is an instance of an inter-VPN resource management event, of the type that would be carried out by the NetBuilder, perhaps in response to a request by VPN-B's control architecture.

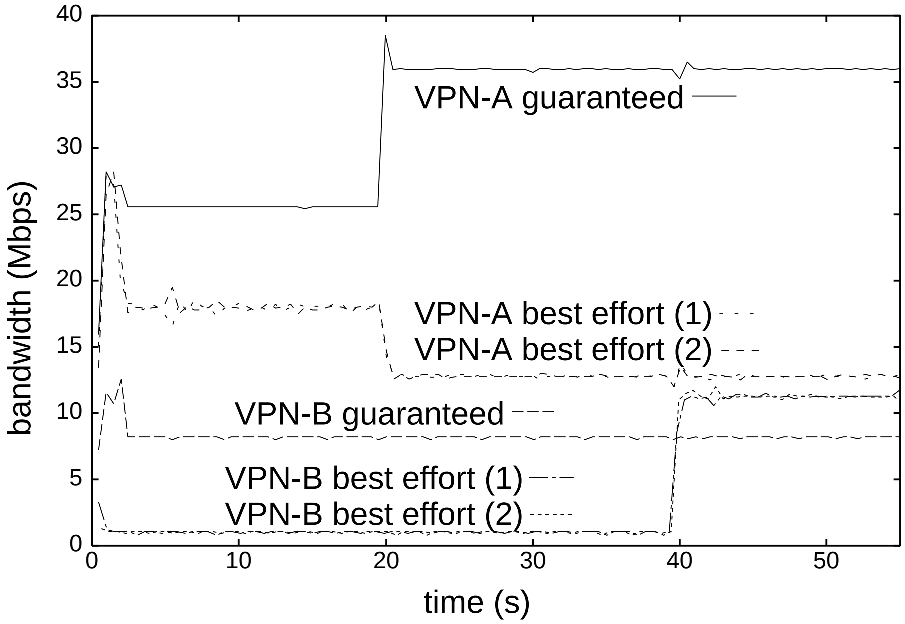


Figure 3.5: Experiment demonstrating inter- and intra-VPN resource management.

3.2.3 The Tempest and VServ

Figure 3.6 depicts how components of VServ and the Tempest interact. Each VPN customer runs their own control architecture, which deals with resource management within that VPN. Applications that use the resources of the VPN interact with the control architecture as appropriate, depending on the nature of the control architecture (for example, through a sockets-style interface), but transmit data directly through the physical network. The VSP owns VServ and Tempest software components, and control architectures may only be allocated network resources, and hence VPNs, through VServ. The diagram shows two logical interfaces exported by the physical network, reflecting the two types of resource management available through the Tempest. VPN resource management is realised through the creation and deletion of switchlets at Dividers throughout the network, and connection resource management takes place within the resource allocation of a particular VPN.

This dissertation is concerned with the resource management issues of VServ, in particular, the per-VPN resource allocation carried out on behalf of VPN customers. It builds on earlier work which proposed the original Tempest design [van der Merwe97b], and complements work examining flexible connection management in the context of the Hollowman, an ATM control architecture for the Tempest[Rooney98a].

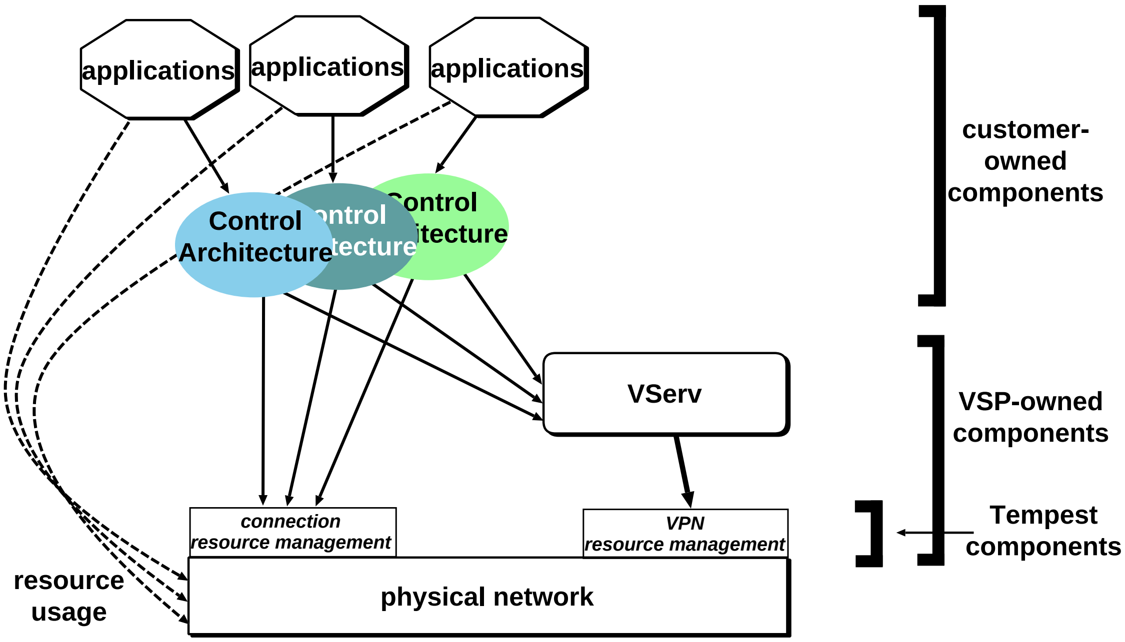


Figure 3.6: Overview of VServ and the Tempest.

The following sections describe the enhancements that were made to the Tempest in order to support VServ. The most important of these are the QoS extensions covered in the next section.

3.3 Quality of Service Extensions

In early versions of the Tempest [van der Merwe97b] resource partitioning was carried out in a very minimal fashion simply by dividing up the label space. These switchlets used a pre-defined VCI range that was also *static*, i.e. could not be altered during the lifetime of the switchlet (although a VPN could be dynamically modified by the addition or removal of switchlets). There were several reasons for these constraints: the switches belonged to the laboratory infrastructure and hence also operated in the manner of a standard ATM network (meaning that label space had to be pinned down in advance to prevent the switch's own signalling software from using it); and development of different facets of the software was proceeding simultaneously, hence several Dividers running at the same time for a single switch had to be accommodated. More extensive resource partitioning, together with capabilities for dynamic resource reallocation, were required for VServ. The next section describes the QoS modifications made to switchlets, and Section 3.3.2 how

QoS parameters were introduced for connections.

3.3.1 Switchlets With QoS

In order to explore dynamic resource reallocation between switchlets, in particular the resource revocation discussed in Chapter 7, the implementation of a switchlet was extended to incorporate QoS characteristics as well as label space. These include bandwidth for each in/out interface and a list of available ATM service categories. These parameters can be specified for each port individually in the switchlet.

Figure 3.7 contains types for the new switchlet specification structure in CORBA IDL. The `ResourceSpec` structure uses service categories defined by the ATM Forum to describe the QoS characteristics of the switchlet. ATM Forum service categories were originally chosen for use in the Tempest because they define a convenient abstraction for hiding the specific buffer allocation and cell scheduling policies of switches [van der Merwe97b]. Because these internal attributes distinguish one switch from another, it may be unrealistic to expect vendors to expose this information in an open control interface.

Nonetheless, it is not the intention that the Ariel interface prevent the *exploitation* of a particular feature of a switch or router. Nor should the generic definition of a switchlet specification be limited to a particular network technology—indeed, future implementations may support VPNs encompassing different types of network for which ATM service categories may not be appropriate. A possible alternative approach is that taken by the IETF draft GSMP standard [Worster00], in which a switch advertises its support for one or more *Services*. The attributes of each Service that are supported by the individual switch are also advertised in the form of one or more *Capability Sets*, and unique identifiers are assigned to each Service and to each Capability Set. Service definitions listed in the current Internet Draft include the ATM Forum Service Categories (similar to the Tempest implementation), Integrated Services, MPLS and Frame Relay. The definition and implementation of an extensible version of Ariel is an area for further work.

Unlike the original switchlet specification, the capacity is allocated per-port rather than per-service category. When creating a connection within an existing switchlet, the Divider determines the proportion of the per-port bandwidth used by that connection according to its service category. In the current implementation, a naïve algorithm is used for this purpose: a connection specifying CBR will use bandwidth equivalent to its peak cell rate (PCR), a VBR connection will use the sustainable cell rate (SCR) and a UBR, i.e. best effort, connection is considered to be using 0[1].

[1]The bandwidth allocated to a UBR connection is accounted within the switchlet as 0, but the connec-

```
enum ServiceCategory { UBR, CBR, VBR };

typedef short PortId;
typedef short SwitchletId;
typedef sequence<ServiceCategory> ServiceCategory_seq;

struct ResourceSpec {
    short min_vpi;
    short max_vpi;
    short min_vci;
    short max_vci;
    long in_bandwidth;
    long out_bandwidth;
    ServiceCategory_seq service_categories;
};

struct PortSpec {
    PortId port_id;
    ResourceSpec port_resources;
};

typedef sequence<PortSpec> SwitchletSpec;
```

Figure 3.7: Switchlet specification types.

This simple approach is adequate for the proof-of-concept implementation described here. A more sophisticated algorithm may give better resource utilisation, but leads to a potential mismatch between the resource allocation policies used on the switch and those used within the switchlet. So a connection request may be approved by the Divider, but rejected by the switch, or vice-versa. These subtleties of internal switchlet resource management are not considered further in this dissertation.

A switchlet creation request passes a switchlet specification to the Divider which checks for resource availability, allocates the resources and creates a new Ariel interface associated with the allocation. It also assigns a unique identifier to the new switchlet and returns a string containing the interface reference for the Ariel interface. Figure 3.8 contains the CORBA IDL for the switchlet creation and deletion methods.

tion itself is actually created with bandwidth equivalent to the remaining uncommitted bandwidth within the switchlet. Multiple best effort connections each receive a fair share of the uncommitted bandwidth.

```
string createSwitchlet(in SwitchletSpec spec,
                       out SwitchletId id)
     raises (DividerResourcesUnavailable,
             DividerInvalidSwitchletSpec,
             DividerOperationFailed,
             DividerNotImplemented);

void deleteSwitchlet(in SwitchletId id)
     raises (DividerSwitchletDoesNotExist,
             DividerOperationFailed,
             DividerNotImplemented);
```

Figure 3.8: Divider switchlet creation and deletion methods.

3.3.1.1 Dynamic Resource Reallocation for Switchlets

The `SwitchletSpec` argument of the `createSwitchlet` invocation includes specific VCI and VPI ranges. The Divider assigns that label space to the new switchlet provided it is not already in use. As discussed in Chapter 6, the NetBuilder determines what the label space ranges should be at the time of switchlet creation.

The previous Tempest implementation pre-allocated VCI ranges to a small and fixed set of VPNs, with each VPN able to acquire a switchlet with the same VCI range on every switch. Although this static behaviour is in general inflexible, it is desirable in some situations to have 'default' VPNs with known label space allocations, for example for the bootstrap VPN described in Section 3.4. This is catered for by an additional method defined for the Divider interface: `createDefaultSwitchlet`.

Dynamic resource reallocation permits the capacity or label space of an existing switchlet to be modified. This entails the addition of two methods to the Divider's management interface, shown in Figure 3.9.

The ability to dynamically modify the size of a switchlet enables VServ to reconfigure VPN resource allocations on-the-fly. The use of the `switchletSpec` structure both to create and modify a switchlet allows fine-grained specification of the resource partition.

```
void addResources(in SwitchletId id,
                  in SwitchletSpec resources)
     raises (DividerSwitchletDoesNotExist,
             DividerResourcesUnavailable,
             DividerInvalidSwitchletSpec,
             DividerNotImplemented);

void removeResources(in SwitchletId id,
                     in SwitchletSpec resources)
     raises (DividerSwitchletDoesNotExist,
             DividerInvalidSwitchletSpec,
             DividerNotImplemented);
```

Figure 3.9: Divider resource reallocation methods.

3.3.2 Connections With QoS

In addition to introducing QoS parameters to the switchlets themselves, the association of connections with QoS *contexts* was also implemented. A context comprises the QoS attributes of a connection, and thus defines the *traffic contract* that is used for in-band policing of data on that connection. This provides the fine-grained intra-VPN resource management capabilities that are available to control architectures of VServ.

The idea of separating resource *allocation* (via the creation of contexts) from resource *usage* (by connection establishment) was motivated by two factors [van der Merwe97b]:

1. to expedite connection setup;
2. to permit control systems to exploit additional knowledge about application-level resource usage to over-commit resources on the switch if desired.

The IDL in Figure 3.10 contains the structures for contexts and connections. Each new context is allocated a unique context identifier, but no policing of potential resource usage within that context takes place until it is requested that a connection be associated with it. A consequence of this is that a control architecture may be able to create multiple connections with a cumulative context allocation that exceeds the switchlet's total resource allocation, and rely on in-band policing to monitor actual usage[2].

[2]Some switches, including the FORE Systems switches used in the Tempest research network, do not

```
typedef short ContextId;

struct ContextSpec {
    ContextId context_id;
    ServiceCategory service_category;
    long peak_cell_rate;
    long sustained_cell_rate;
    long max_burst_size;
};

struct ConnSpec {
    PortId inport_id;
    short invpi;
    short invci;
    PortId outport_id;
    short outvpi;
    short outvci;
    ContextId context_id;
};
```

Figure 3.10: Ariel context and connection specification types.

Every switchlet is assigned a default context with which connections are associated when no context is specified. This means that control architectures need not necessarily be aware of QoS issues, and need not create a context in advance in order to establish a connection and transmit data. It also has the advantage that even if the network node itself does not support connection QoS, as is the case with the two ASX-100 ATM switches in the Cambridge Computer Laboratory's research network, the connection manipulation operations are the same.

In the current implementation, all switchlets are given the same default context—a best effort resource allocation. However it may be appropriate to assign different default contexts according to the characteristics of the control architecture that owns the switchlet.

The invocations on the Ariel interface to create and destroy contexts and to set up and tear down connections are shown in Figure 3.11.

allow connection resource reservations (i.e. usage contracts) to exceed availability. Thus in this case the corresponding Tempest implementation does not support such behaviour.

```
ContextId createContext(in ContextSpec context_spec)
     raises (ArielInvalidContextSpec,
             ArielOperationFailed,
             ArielNotImplemented);

void destroyContext(in ContextId context_id)
     raises (ArielInvalidContextId,
             ArielOperationFailed,
             ArielNotImplemented);

void createConnection(in ConnSpec connection_spec)
     raises (ArielInvalidConnectionSpec,
             ArielFailedToContactSwitch,
             ArielOperationFailed,
             ArielNotImplemented);

void deleteConnection(in ConnSpec connection_spec)
     raises (ArielInvalidConnectionSpec,
             ArielFailedToContactSwitch,
             ArielOperationFailed,
             ArielNotImplemented);

ContextId getConnectionContextId(in ConnSpec connection_spec)
     raises (ArielInvalidConnectionSpec,
             ArielNotImplemented);
```

Figure 3.11: Ariel context and connection methods.

3.3.2.1 Dynamic Resource Reallocation for Connections

Resources can also be dynamically reallocated between *connections*. Figure 3.12 contains the IDL for two methods added to the Ariel interface.

Theoretically, renegotiation of the QoS contract associated with individual flows is of some benefit. It has been observed that variable bit rate compressed video traffic is bursty over multiple time scales [Garrett94]—short bursts resulting from action within a scene and long bursts caused by variation of information content in different scenes. As a result, there is a trade-off between large buffer sizes, which are expensive with long delays, and high loss rates. In addition, reduced statistical multiplexing gain must be balanced against the con-

```
void addConnectionResources(in ConnSpec connection_spec,
                            in ContextSpec resources)
     raises (ArielInvalidConnectionSpec,
             ArielResourcesUnavailable,
             ArielInvalidContextSpec,
             ArielNotImplemented);

void removeConnectionResources(in ConnSpec connection_spec,
                               in ContextSpec resources)
     raises (ArielInvalidConnectionSpec,
             DividerInvalidContextSpec,
             ArielNotImplemented);
```

Figure 3.12: Ariel connection resource reallocation methods.

sequent loss of protection (i.e. data lost through coinciding bursts from multiple sources because of inadequate buffering inside the network or at the receiver) [Grossglauser97]. Such traffic flows may receive better service with dynamic renegotiation of their QoS parameters.

However, in practice, the dynamic renegotiation of the QoS associated with an existing connection is not a panacea. The amount of per-flow state that is maintained at each network node is often reduced by allowing multiple connections to share the same QoS context. In the case of FORE ATM switches, a connection's traffic class and its parameters define the leaky bucket, (i.e. traffic shaper), instance for that connection. Therefore multiple connections with the same QoS description share the same leaky bucket. The technique that was adopted in this implementation for renegotiation on these switches involved creating the new context in advance, deleting the original connection, and then creating a new connection with the new context. Clearly this will result in a glitch in the data stream, and more graceful renegotiation solutions are desirable. In general, solving this problem requires careful engineering to ensure that re-ordering of cells does not occur, and to minimise disruption in the traffic.

3.4 Bootstrap VPN

The bootstrap VPN provides basic start-of-day connectivity between Tempest components, enabling the dedicated control VPN to be initialised, after which VServ can be started.

It therefore only requires a minimal amount of resource, and its control architecture need only support the basic functionality for this task.

As explained in Section 3.2.1, Tempest components communicate using a CORBA-compliant DPE. The CORBA specification [OMG00] defines the Internet Inter-ORB Protocol (IIOP), which specifies how messages of the General Inter-ORB Protocol (GIOP) are exchanged over TCP/IP connections. Thus the control architecture of the bootstrap VPN offers a TCP interface to its applications in order to allow the bootstrapping process to use the same DPE.

A control architecture for the bootstrap VPN, called *Bootstrap IP Control Architecture* (BICA) was designed for ATM using the x-kernel [Hutchinson91]. The x-kernel is an architecture for constructing and composing network protocols. It provides a set of abstractions for describing protocols, together with a library of routines for common protocol tasks.

BICA over ATM can be pictured as the protocol graph depicted in Figure 3.13. The IP, TCP and UDP protocols are all provided in the standard Unix distribution of the x-kernel. xksocket is an x-kernel protocol, that provides BSD sockets functionality. It is currently only available in the version for Mach, and was ported to run in the Unix environment used in the Tempest. MIP is a minimal version of the standard x-kernel IP protocol, that makes use of the underlying FMIP or XMIP protocol to deliver packets. FMIP and XMIP are two of a family of protocols that are identical except for being programmed for different network interfaces. FMIP uses the FORE ATM API, and XMIP the X/Open Transport Interface (XTI) [OpenGroup94].

There is no routing protocol in BICA. Instead it sends a packet on an ATM PVC as determined by a lookup in a configuration table that maps IP addresses to pre-configured PVCs. The PVC lookup operation takes the place of an address resolution protocol, and relies on a pre-existing full mesh of PVCs in the network.

When the Tempest is brought up for the first time, an x-kernel server is run on each workstation hosting a Tempest component. Using an instance of the DPE that has been linked against the BICA sockets library, a bootstrap application can then make the appropriate invocations on the Tempest components to enable the switchlets that comprise the control VPN to be created, and the full-blown control architecture for that VPN to be deployed.

Since its PVCs must be pre-configured, the switchlets of the bootstrap network are examples of (hard-wired) *default* switchlets. The bootstrap VPN relies on owning a particular set of labels, and only one instance of its control architecture can be operational on a single physical network.

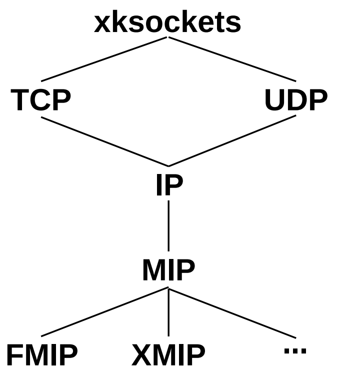


Figure 3.13: Protocol graph for BICA.

3.5 Implementation on MPLS

The Tempest ATM implementation is a well established codebase that has evolved over four years, and to which several people have contributed. It is currently operational on a network consisting of 5 switches, all fully connected, 4 Solaris workstations, 2 Linux PCs, 5 AVAs (ATM video encoders) and 2 ATVs (ATM video decoders). The network is also connected by wide-area link to a smaller ATM research network at British Telecom's Adastral Park. In this section a recent implementation of the Tempest which uses MPLS and IP is described.

Connectionless networks such as IP networks do not naturally provide a separation between the control and data planes. However increasing pressure for such networks to support traffic engineering (e.g. load balancing) and QoS has led to proposals for more connection-oriented approaches. In particular, MPLS divorces routing functionality from packet forwarding, allowing very fast packet switching as well as flexible routing strategies. Such a mechanism provides a means of implementing the Tempest on a packet switched network, because similarly to ATM, connection and resource reservation functions can take place out-of-band. MPLS was described in more detail in Section 2.2.2.

The MPLS version of the Tempest is based on a freely available MPLS implementation for Linux [Fraser00], developed within the Cambridge Computer Laboratory [Mortier00]. MPLS establishes 'virtual' connections (LSPs) along which packets with the appropriate label are forwarded. As with the ATM implementation, a resource-guaranteed VPN on MPLS comprises a set of switchlets across which LSPs can be set up by the control architecture using the standard Ariel interface.

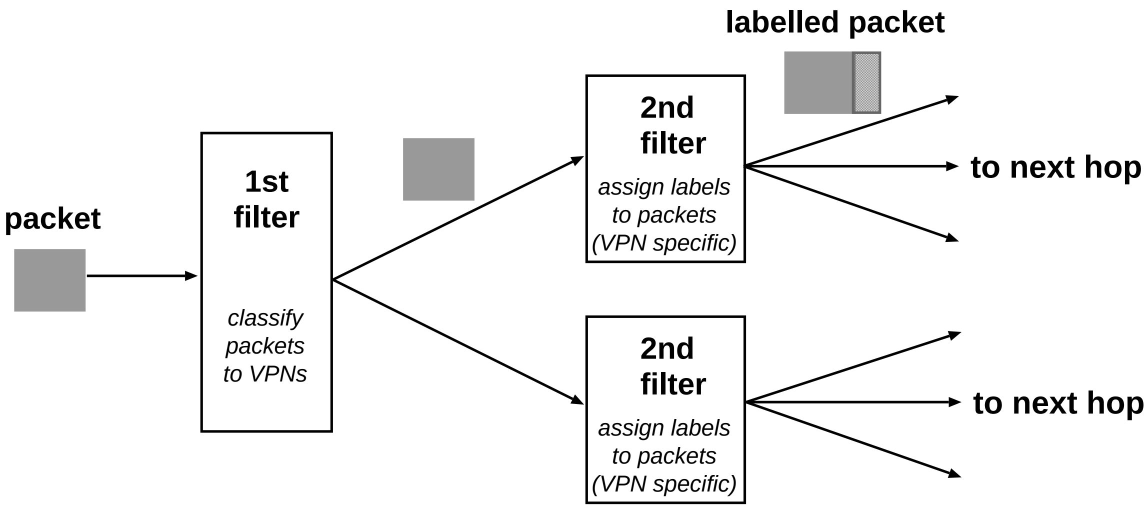


Figure 3.14: Classification of packets at the ingress router in a Tempest MPLS implementation.

A key difference between MPLS and ATM is that additional functionality is required in the data path at the ingress router in order to classify the VPN to which a packet belongs, and to then assign it a label. Standard MPLS classifies unlabelled traffic to an FEC according to some policy, for example network layer route information such as IP destination address. The semantics of this classification are defined by the LDP; since a label only has significance between two hops, the downstream node determines the binding of FEC to label.

In a Tempest VPN service each control architecture has available a range of labels that it may assign to incoming, unlabelled, traffic as it sees fit. A straightforward lookup of destination address will be insufficient, as the control architecture may wish to apply arbitrary classification policies. Dynamically loadable modules allow the control architecture to exert some influence over the decision process. Classification policies can be expressed in a packet filter language, such as Mach Packet Filter (MPF) [Yuhara94], and installed for each VPN at the appropriate (ingress) routers. To avoid the possibility of one control architecture 'stealing' packets belonging to another, every VPN that is created is associated with a default first pass filter which performs the initial mapping of a packet to that VPN. This identifies the appropriate second filter, consisting of the classification code downloaded by the control system, which then allocates a label to the packet. Figure 3.14 illustrates this process.

This scheme allows VPN interoperability, both with other VPNs within the same domain and with traffic from outside the VSP's network. Routing updates from external nodes must be handled at the edge router, with reachability of a destination address depending on the connections established within the relevant VPN.

3.6 Related Work

Many of the concerns that motivated the development of the Tempest have also motivated other, related work. These efforts fall into two broad categories: *active networks* and *service architectures.*

3.6.1 Active Networks

In an active network users can program the internal nodes to perform customised computation on packets traversing the network. The active networks concept is motivated by "technology push"—support for code mobility and restricted execution—and "technology pull"—the plethora of specialised nodes that provide for user-driven computation inside the network such as firewalls, proxies, multicast routers and other services [Tennenhouse96]. Similar to the devolved control and open signalling techniques adopted in the Tempest, active networks provide a mechanism for supporting network programmability. In an active network, different packets can receive specialised forwarding treatment, which effectively allows multiple control systems within the network. Unlike the open signalling approach, control operations take place in-band, and a hop-by-hop forwarding model is assumed. Network QoS is not explicitly addressed, but rather mechanisms that are available in the network itself are relied upon.

Two different mechanisms for programming network nodes have been identified: the *discrete* approach where code is downloaded into the switch or router and a field in the packet header identifies which program is to be run on the packet, and the *integrated* approach where *capsules* transport the code themselves that is then executed at each node they pass through. Within this framework many other issues must be considered, in particular security (integrity of code and authentication), safety (controlled resource usage), performance and interoperability.

Since the original proposal, additional definitions have been coined. An active network node comprises two types of entity—the *execution environment* (EE) within which *active applications* are executed, and the *node operating system* (NodeOS) which is responsible for resource partitioning and other shared functionality between multiple EEs [Smith99].

Code executed within the EEs at a router is dynamically injected, either as a capsule or downloaded onto the node in order to perform some computation over incoming packets.

Customisation of the network control facilities offered in the Tempest has been proposed by means of *sandboxes* that allow the execution of dynamically loadable code within Tempest components [Bos99]. Although not advocating the integration of control and data that is found in active networks, it is recognised that in-band computation can occasionally be useful, for example in filtering, compressing or encoding of data. Consequently an implementation of an active network node is described, using the extended version of the Tempest. The data path is accessed by "capturing" an endpoint of a connection using a dynamically loaded module at the control architecture's Ariel interface. It is pointed out that open signalling and devolved control can be designed to support the active networks approach, but the reverse is not true due to the tight integration of control and data in active networks.

3.6.2 Service Architectures

Service architectures open up the network to new and novel services using devolved control techniques. The xbind architecture [Lazar95, Lazar96] is an example which specifically aims to facilitate the creation and deployment of multimedia services on ATM networks. It is an implementation of a conceptual framework known as the Extended Reference Model (XRM) within which higher-level services are created and managed through the composition of lower-level abstractions. Access to the local physical resources is provided through QoS abstractions, which are assembled by a higher-level entity using algorithms to provide end-to-end communications with QoS guarantees. These algorithms are in turn abstracted and presented to the highest level of the model as a set of APIs which can be combined to create and manage multimedia services.

Although xbind addresses many of the problems inherent in 'closed' signalling by opening up access to the physical resources of the network and by providing scope for manipulating those resources in whatever manner is fitting, there are still areas of concern within the architecture itself. For example, the definition of the QoS abstractions relies on switch manufacturers providing this information, and realising the abstractions as a set of APIs means having to upgrade whenever a new service class is defined. The definition of a fixed set of abstractions implies an attempt to cater for all possible future configurations and services, which cannot be regarded as a realistic goal.

Another example of a service architecture is Telecommunications Information Networking Architecture (TINA) [Barr93], which is defined by a consortium of telecommunications and computer companies. TINA aims to provide a framework in which diverse and emerging communications and network technologies can be incorporated into the telecommunications infrastructure while maintaining existing systems and standards. Its remit encompasses traditional telecommunications services as well as multimedia and internet technologies in a model that covers a range of requirements from security and billing down to the data transport. Its somewhat heavyweight specification tends to hide the nitty-gritty details and also makes widespread implementation unlikely.

Service architectures are similar to the Tempest in that they logically separate the network services and applications from the physical infrastructure. However none provide a general solution to the problem of monolithic and complex control systems, and thus do not fully support the multiservice network. Both of the architectures described here could be deployed as control architectures (simultaneously, if required) within the Tempest environment.

3.7 Summary

This chapter described the Tempest network control framework and reviewed the motivation for its original conception. In order to properly support a resource-assured VPN service, the initial Tempest design and implementation was extended to incorporate a richer QoS environment. Dynamic resource reallocation facilities were introduced which are of particular relevance for the resource revocation mechanism discussed in Chapter 7. A bootstrap VPN for the ATM implementation was described.

The implementation of the Tempest on MPLS enables its benefits to be realised over a much wider range of network technologies. This has important implications for interoperability with other networks, especially the Internet, but also affects the design of the Tempest architecture itself. In order to sustain strict resource partitioning in the network, traffic entering the Tempest domain from external sources must be accounted to a VPN. In addition, once a packet has been assigned to a VPN, the framework must ensure adherence to the relevant control architecture's routing policies. A solution to this problem was proposed whereby packet filters are deployed at the ingress router, with one filter mapping packets to VPNs, and the next being dynamically installed in the router by the control architecture itself.

The Tempest forms the underlying system support for VServ. The remainder of this dissertation is concerned with issues specific to VServ itself, starting with the process of VPN design, which is discussed in the following chapter.

Chapter 4

VPN Specification and Design

The *VPN Service Provider* (VSP) is responsible for the domain-wide management of VPNs. It must allocate resource aggregates throughout the network, sharing resources *between* VPNs. It is also concerned with the usability of VServ, VPN design and operational issues such as interoperability, efficiency and robustness. In addition, the VSP has the dual goals of satisfying the requests of individual customers while also maximising overall network utility. It needs to be able to implement *policies* as part of the design procedure, and to modify those policies as necessary. The following three chapters examine VSP management in more detail, focusing on VPN creation in the context of providing a VPN service that meets the goals specified in Section 1.2.1.

The VSP is not necessarily the owner of the network: inter-domain agreements can be negotiated that enable a VPN to span multiple domains, or the VSP may themselves be offering a service over a VPN obtained from one or more other VSPs. Regardless of whether it offers its VPN service on a physical or virtual network, the VSP must manage its network as a whole.

A highly dynamic VPN service can only be realised if much of the administrative and processing burden of establishing, modifying and maintaining VPNs is automated. This chapter examines the issues involved in the dynamic provisioning of VPNs and describes a framework for the automation of the VPN design procedure in light of both the VSP's responsibilities and its goals. The feasibility of VServ depends on VPN design that is both effective and responsive. The automated design process presented in this chapter makes use of the characteristics of the desired VPN, together with the current state of resource reservations in the network, to mitigate much of the inherent complexity of the procedure.

The following sections cover the automated design and development of VPNs. First an overview of the entire process is given followed by the description of a case study in VPN design to be used throughout the chapter. The individual stages within the process are then considered in more detail. In Section 4.3 the translation of user-level VPN requirements to network quality of service parameters is examined, and VANDAL, a simple notation for describing VPNs, is introduced. The ultimate goal of the design process is to find a topology and resource allocation that meets customer requirements whilst taking into account VSP policies and current resource availability. This *topology search* problem can be expressed as a least cost search over a weighted, directed graph with certain constraints. Issues concerning the determination of the edge weights for this graph according to current resource availability and administrative preferences are discussed in Section 4.4. Related work on dynamically provisioned VPNs is reviewed in Section 4.5. Chapter 5 considers the algorithm used for the search itself.

4.1 The VPN Design Process

The process of VPN design takes a description of the desired VPN and produces a switchlet specification, with the primary goal of finding the cheapest VPN that satisfies the requirements of both customer and VSP. The 'cost' of a given VPN is determined by a combination of the VPN's attributes, VSP policies and the current resource availability in the physical network. Other goals for the process include:

- Enhance usability by presenting a simple way of describing the desired VPN.
- Support VSP policies and resource usage goals.
- Increase responsiveness by reducing the complexity of the topology search as much as possible.

An overview of the automated VPN design process is shown in Figure 4.1 and consists of three phases. The pre-search steps fall naturally into two distinct phases, with the third being the search itself. The first phase is **VPN description**, which comprises the translation of high-level customer requirements to a formally expressed network level description. This stage fulfils the goal of enhanced usability. The second, known as **search space construction**, involves tailoring the search space according to attributes of the desired VPN, current resource availability and VSP policies. This addresses the last two goals for the design process identified above. The final phase—**topology search**—is the search for a close-to-optimal VPN topology, returning the switchlet specification of the new VPN.

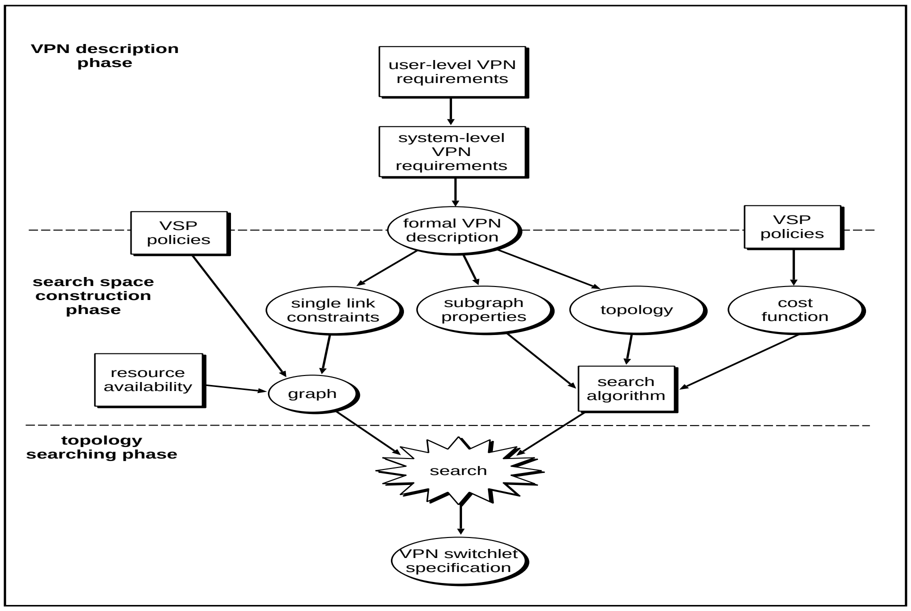


Figure 4.1: The VPN design process.

Chapter 5 describes the topology search phase, which deals with an optimisation problem that is theoretically intractable. Although a search over a graph of the entire physical network is necessary in the worst case, the design process reduces the likelihood of such a search being required. Information about the desired VPN, provided by the earlier phases of the process, is used to prune the network graph and, where appropriate, to select a suitable polynomial time search algorithm. As will be demonstrated, such customisation mitigates the complexity of the search, and makes the procedure feasible in practice.

In the following section, a case study scenario is described. The case study is used throughout the chapter to illustrate the concepts under discussion.

4.2 Case Study Scenario

The case study involves the acquisition of a VPN used to run a video conferencing session. This VPN's control architecture is a form of Videoman [van der Merwe98a], known as *Videoman Light* (VL). Videoman is a service-specific video conferencing control architecture which makes efficient use of network resources by optimising connection structures for the particular requirements of a continuous media conferencing environment.

Similar to Videoman, VL employs gather and distribution trees to optimise resource usage of the multicast groups. Within the core switch, floor control is exerted by mapping a single multicast connection from the source port to multiple sink ports. Whenever the designated source changes, the connection is remapped.

VL differs from Videoman as follows. When Videoman starts up it acquires a VPN comprising the entire available physical topology, and then calculates a suitable core switch before establishing the conference connections. In contrast, VL pre-allocates a suitable core switch and ensures that this switch is present in the initial VPN topology. VL potentially wastes fewer network resources, although it does rely on customer knowledge of the optimal placement of the core switch (which is not unrealistic in certain situations).

One final difference between VL and Videoman is that Videoman was developed as a proof-of-concept control architecture for the original version of the Tempest. VL makes use of the QoS options available in the newer dynamic and QoS-aware version.

The network topology for this case study is shown in Figure 4.2. It comprises 7 switches, and shows the bandwidth availability for each link, together with available label space in brackets. For clarity, links are shown as bidirectional, with a single figure for bandwidth and label space availability. It is assumed in this case that resource allocations are always bidirectional although in practice this may not be the case. The diagram also shows the endpoints—the camera sources, television sinks and audio-only sinks—that will participate in the VL VPN. The node designated as the core switch of the video conference is highlighted.

Figure 4.3 shows a schematic of the desired VPN topology for running VL using the sources and sinks shown in Figure 4.2. The video sources attached to switches A and E each require 10Mbps bandwidth to the core switch for sending video, and 64Kbps for sending audio, together with label space for two connections. There is an audio-only sink on switch D, needing 64Kbps and a single connection from the core switch. The remaining three sinks on switches A, B and E take both video and audio, so require 10Mbps plus 64Kbps and two connections. As a further constraint (somewhat contrived for the purposes of this example), a limit of 5 hops from the core switch to any endpoint is imposed.

VL and the network topology used for this case study were devised specifically to illustrate the concepts explained in this chapter. It will be shown how a formal VPN specification for VL is produced, and also how VSP policies, together with the requirements of VL, influence the search for an appropriate VPN topology.

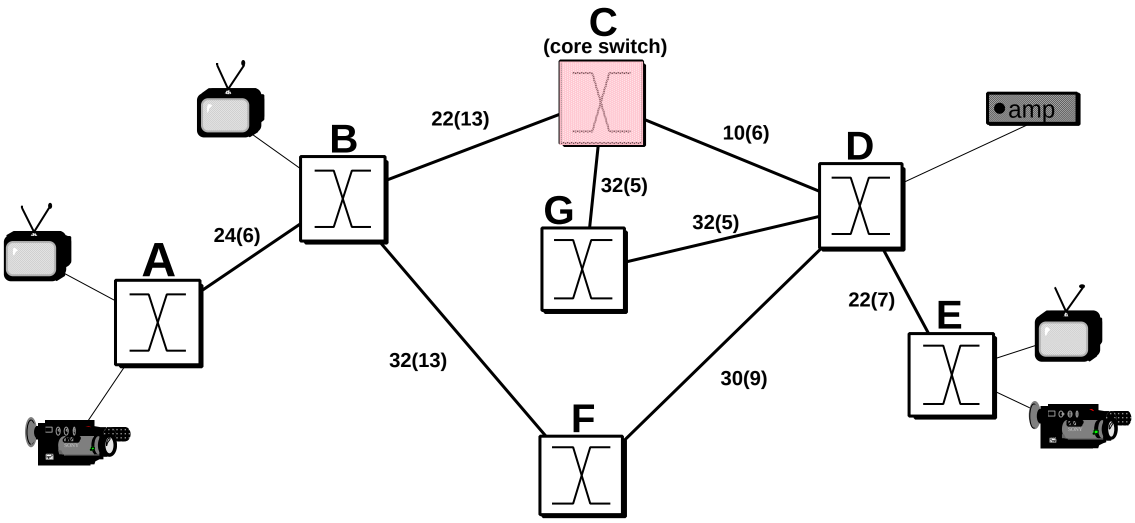


Figure 4.2: Network topology for case study. Each link is labelled with available resources: bandwidth (size of label space).

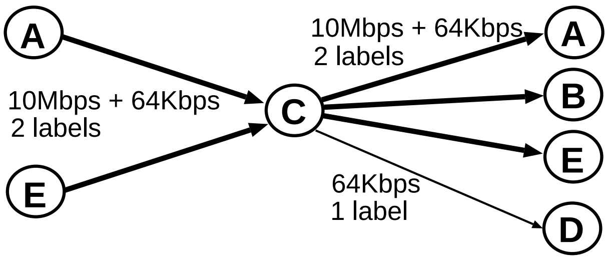


Figure 4.3: Schematic of the VPN topology and resource requirements for VL in the case study network.

4.3 VPN Description

For some types of customer, particularly the casual or naïve VPN user, the mapping from a user-level articulation of their requirements to the corresponding network-level resource specification is not well understood. In the VPN description phase a high-level request such as "sufficient bandwidth to make a phone call" might be translated to "64Kbps constant data rate". The output from this first phase is a VPN specification that is suitable for automated parsing in order to extract three pieces of information:

1. **Single link constraints** prune the graph representing the underlying network topology. Such constraints include availability over the desired VPN duration (which may involve some scheduling if advance reservations are permitted), and QoS constraints such as bandwidth, delay, loss and jitter. Any link that cannot meet these requirements is discarded.

2. **Subgraph properties** (together with the desired VPN topology) determine whether a particular subgraph is a candidate topology. Some subgraph properties are cumulative with the size of the subgraph, for example end-to-end hop count or delay. If any such properties have been specified, then a subgraph obtained as a partial solution can be examined and if the specified limits are already exceeded then that line of search can immediately be abandoned, reducing the size of the search space.

3. **The VPN topology** not only defines the constituent nodes, but also the *shape* of the VPN (i.e. whether it forms a tree). This information is used to select an appropriate search algorithm—in certain situations a more tractable searching procedure may apply, as will be discussed further in Section 5.1.

The VPN description phase consists of two stages. Firstly, user-level requirements are translated to a specific network-level description, as discussed in Section 4.3.1. Secondly, these requirements are expressed in a domain specific notation known as VANDAL, presented in Section 4.3.2. Finally, in the search space construction phase considered in Section 4.4, the VANDAL description is parsed to derive the desired topology, single link constraints and subgraph properties.

4.3.1 From User Requirements to Network QoS Parameters

The attributes that distinguish one VPN from another are topology, resource allocation, duration and the deployed control software. A VPN that is short-lived and that runs application or service-specific control software is referred to as *lightweight*. Distributed games and broadcast lectures are typical services for which lightweight VPNs may be used. In contrast, *heavyweight* VPNs exist for long periods of time and run general purpose control software. Examples of heavyweight VPNs are those owned by ISPs, telephone companies or corporate users to support their intranet. These terms are defined for descriptive purposes only; many VPNs will fall somewhere between these two extremes.

The degree of detail specified by customers who wish to acquire a VPN also differs widely. A VPN can be loosely defined with no particular resource requirements specified—for example "a cheap network between A and B"—or it may have all of its attributes specified. The degree of detail will depend upon the sophistication of the customer and the intended use of the VPN.

The control software that is deployed is a key factor in determining the resource requirements for a VPN. Therefore, third party control architectures are distributed with configuration 'wizards' containing the information needed to obtain a VPN with suitable QoS characteristics. This is similar in spirit to the GNU `configure` utility, included with many source code distributions in order to automate the installation setup and to assist program portability across different Unix variants. A control architecture `configure` wizard holds the default QoS parameters required for its operation, and is parameterised by the user with situation or usage-specific information.

As an example consider an instance of INCA [Isaacs99], a telephony control architecture that supports dynamically loadable intelligent network (IN) services such as call forwarding. The network resource requirements for INCA are dictated by the number of telephone handsets, hence the deployer need only specify the handset locations. INCA's `configure` wizard will map this to the initial label space, bandwidth, delay, jitter and redundancy requirements using internal knowledge about the network resources needed for basic voice calls and its default IN services, as well as for its routing and call admission control algorithms.

A customer is not compelled to use a control architecture's `configure` wizard. The output from `configure` is a resource specification in a notation called VANDAL, described in the next section. Customers with the expertise, or those using a control architecture that does not provide a `configure` wizard can choose to write the specification directly.

A significant advantage of automating the VPN design process is that customers do not have to be networking experts, or even interact with those who are experts, to acquire a VPN that is suitable for their needs. For this reason a mechanism for the specification of network QoS parameters that is usable by even the most naïve customer is provided.

4.3.2 VANDAL: A VPN Description Language

Currently the provision of VPNs is often governed by *service level agreements* (SLAs). As described in Section 2.1, SLAs specify the level of service agreed between the supplier and customer in terms of measurable network parameters, and often require negotiation by networking and legal experts. The contents of the SLA may usefully act as a formalised description of the characteristics of the desired VPN. However, these types of SLAs are inappropriate for a highly dynamic VPN service: the overheads and costs of producing the SLA are disproportionate, and to a certain extent the resource guarantees inherent in network partitioning obviate much of the need for monitoring. Although some customers require stringent service monitoring (in which case a traditional SLA is suitable), for most lightweight VPNs an alternative means of specifying the VPN and service expectations is required. The simple VANDAL notation presented here is used to create an explicit statement of requirements in which the extent of a VSP's service commitment is contained by restricting the expressiveness of the notation. The successful design of a VPN using a customer-provided description acts as a contractual guarantee for a network that meets the specified requirements.

A description of a VPN in VANDAL comprises two top level elements: a topology description and a resource description. Figure 4.4 contains the BNF-style grammar for VANDAL, with terminal symbols in uppercase. The topology description is written as a set of nodes and a set of links. The node list essentially defines the switchlets that the customer is expecting to obtain, and the links indicate the incoming and outgoing ports that should be present in those switchlets.

```
<vpnspec>::         <topology> <rspecs>
<topology>::        [ <nodes> ] : [ <links> ]
<nodes>::           NODE | <nodes>, NODE
<links>::           <link> | <links>, <link>
<rspecs>::          <rspec> | <rspecs>, <rspec>
<rspec>::           <rdescription> : <rtopo>
<rtopo>::           aggregated([ <links> ]) | [ <links> ] | [ <nodes> ]
<link>::            NODE : NODE
<rdescription>::    RESOURCE OPERATOR VALUE
```

Figure 4.4: BNF-style grammar for VANDAL.

The resource description is made up of a set of resource requirements, each applied to a subset of the topology. Each resource type in VANDAL has a name, together with associated operators and value types. Every individual VPN service will have a mapping of the resource terminals to individual resource types, operators, value ranges and node identifier types. Kinds of resource that can be incorporated include label space, hop count, QoS parameters such as delay, loss tolerance and bandwidth, as well as start time, duration, redundancy and cost. The language definition does not specify the resource types because the applicable resources depend on the capabilities of the underlying network, and also on the degree of specification that the VSP wishes to allow its customers. Greater freedom inevitably leads to more complexity in the system.

A resource requirement is applied to a topology specification that is either a list of one or more nodes, one or more links, or an aggregation of those links. This allows requirements to be expressed that apply to the network as a whole, as well as those that apply to particular elements inside the VPN. For example, to specify a VPN described as "a 10Mbps network between A, B and C", it is neither necessary for the customer to over-provision their VPN, nor to decide exactly how much any given link must carry.

To illustrate the use of VANDAL, an example set of VPN requirements, together with the corresponding VANDAL specification, is given below.

1. 3 nodes `A`, `B` and `C`, fully connected.
   This is expressed by the `<topology>` part of the VANDAL specification. The `<links>` component captures the requirement for a fully connected VPN:

   ```
   [ A, B, C ] : [ A:B, B:A, A:C, C:A, B:C, C:B ]
   ```

2. No two nodes separated by more than four hops.
   This is a resource specification that applies to every VPN link:

   ```
   hopcount ≤ 4 :  [ A:B, B:A, A:C, C:A, B:C, C:B ]
   ```

3. An end-to-end delay of less than 2 milliseconds on the link between `A` and `B`, and this link to have built-in redundancy.
   The delay requirement is captured by a resource specification applied to the specified link only:

   ```
   delay ≤ 2 :  [ A:B, B:A ]
   ```

   and the original topology specification is augmented to include the redundancy requirement in the form of repeated links:

   ```
   [ A, B, C ] : [ A:B, B:A, A:C, C:A, B:C, C:B, A:B, B:A ]
   ```

4. A delay of less than 3 milliseconds on the remaining links.
   The resource specification here will be:

   ```
   delay ≤ 3 :  [ A:C, C:A, B:C, C:B ]
   ```

5. A total bandwidth across all the links of not more than 10Mbps.
   The `aggregated` keyword captures this requirement:

   ```
   bandwidth ≤ 10 :  aggregated([ A:B, B:A, A:C, C:A, B:C, C:B ])
   ```

6. The VPN duration to be one week.

```
duration = 604800 :  [ A:B, B:A, A:C, C:A, B:C, C:B ]
```

Another more realistic example is a notional VANDAL specification for an IP telephony service. Assume a requirement for constant bandwidth 30Kbps per call, with sufficient resource for one call at a time at the network edges, and in the core of at most 10 and at least 5 calls (where concurrent calls are multiplexed). An additional requirement is 100% resilience to single link failures in the core of the network, i.e. on the virtual link between nodes D and E. Figure 4.5 shows the VANDAL specification, with the VPN topology comprising the nodes $A \dots E$ as illustrated.

```
[A, B, C, D, E] : [A:D,D:A,D:E,...]
 bandwidth=30 : [A:D,D:A,B:E,E:B,C:E,E:C]
 labelspace=1 : [A:D,D:A,B:E,E:B,C:E,E:C]
 service="CBR" : [A:D,D:A,B:E,E:B,C:E,E:C]
 labelspace>=5 : [D:E,E:D]
 bandwidth>=150 : [D:E,E:D]
 labelspace<=10 : [D:E,E:D]
 bandwidth<=300 : [D:E,E:D]
 singlelinkresilience=1 : [D:E,E:D]
```

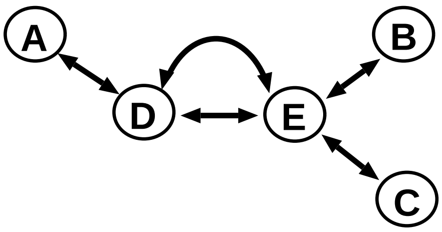


Figure 4.5: VANDAL specification of a VPN for an IP telephony service, together with a schematic of the desired virtual topology.

As discussed in Section 4.4.1, a VANDAL specification is automatically parsed to produce single link constraints, subgraph properties and the desired VPN topology.

4.3.3 Case Study: Using `configure` to Generate a VANDAL Description

The `configure` wizard for the video conferencing control architecture, VL, requires the following information from the user:

- a list of sources,
- a list of sinks, and
- a designated core switch.

From the details given in Section 4.2 and illustrated in Figure 4.3, a network-level characterisation of the required VPN is derived as follows:

- **Topology:** A, B, D, E and the core switch C. Links between these switches as required to form the gather and distribution trees, and also the port numbers for connection to the specified endpoints.
- **Bandwidth:** assuming only a single source is visible at any one time, bandwidth sufficient for 2 separate core-based trees (video and audio) is required:
  - 10Mbps between the core and all of the video sinks;
  - 10Mbps between each of the sources and the core for sending video;
  - 64Kbps between the core and all of the audio sinks;
  - 64Kbps between each of the sources and the core for sending audio;
- **Label space:** two connections are required from each source to the core, two from the core to each audio/video sink, and one from the core to each audio-only sink.
- **Hop count:** each endpoint can be at most 5 hops from the core.

VL's `configure` utility generates the following VANDAL specification for this VPN:

```
[ A, B, D, E, C ] : [ A:C, E:C, C:A, C:B, C:D, C:E ]
bandwidth = 10.0625 :  [ A:C, E:C, C:A, C:B, C:E ]
bandwidth = .0625 :  [ C:D ]
labelspace = 2 :  [ A:C, E:C, C:A, C:B, C:E ]
labelspace = 1 :  [ C:D ]
hopcount ≤ 5 :  [ A:C, E:C, C:A, C:B, C:D, C:E ]
```

4.4 Construction of the Search Space

The VPN search space construction phase performs two functions:

1. It incorporates the current state of the network together with the longer term requirements of the VSP into a search for a VPN that meets the customer's description.

2. It refines the search space and procedure as far as possible to be favourable for the specific search requested. Because the search problem is theoretically intractable, this is important to help make the finding of a solution practical in non-trivial networks.

This phase of the VPN design process produces a graph representing the network over which the topology search takes place, specific to the search requirements. It also generates code that is dynamically loaded into the searching procedure which prunes the search space when paths are encountered that don't meet cumulative subgraph requirements, and which provides a tailored cost function. Section 4.4.1 describes the parsing procedure further.

A snapshot of the current resource availability gives the physical topology and available resources at each link. Chapter 6 discusses how such a snapshot might be obtained when the NetBuilder is replicated—in other words, when several NetBuilders are acting independently and concurrently. As described in Section 4.4.2, VSP policies such as balanced link utilisation dictate how these attributes are combined to produce weights for the edges in the graph. Edges with attributes that don't meet the single link constraints are eliminated, and the resulting graph is then used as the search space in the third phase.

A cost function for comparing one candidate subgraph with another is generated by incorporating VSP policies. An appropriate search algorithm is selected, and then the code is augmented with checks for the derived subgraph properties and with the cost function.

The output from the second phase is the tailored searching procedure together with the network graph over which it is run.

4.4.1 Parsing the Formal Description

In the first stage of search space construction, the formal network specification obtained in the previous phase is parsed. This gives the desired VPN topology, constraints on single links in the VPN and properties of its topology as a whole.

The VANDAL grammar is context-free and so the language is amenable to straightforward and well known parsing techniques. A partial implementation of the parsing procedure was carried out which generates source code in a form suitable for dynamically loading into the search algorithm and graph determination modules. The output is Python[Python] modules that are imported at run time by the relevant programs.

The output includes code for the following functions:

- `prunesinglelinks(G)`: removes edges from the physical topology graph `G` that have insufficient availability for the required single link attributes, along with any nodes that are made redundant as a consequence.

- `iscandidate(G)`: given a subgraph `G`, checks whether `G` contains the desired VPN topology, and that end-to-end subgraph properties are satisfied.
- `checksubgraphproperties(G)`: may be called from `iscandidate()` or independently, checks whether end-to-end subgraph properties are satisfied in `G`.
- `getlink<resource>usage(L)`: for all `<resource>` types supported in this VPN service, return the desired usage for the virtual link `L`.

Optimisations are applied as appropriate. For example, multiple constraints referring to the same resource type on the same virtual link are merged into a single rule.

The VANDAL description specifies link requirements in terms of *virtual* links. In the physical topology there may be multiple paths between two nodes forming a single virtual link. Furthermore, different virtual links may have different requirements for the same resource type. There are two ways in which this can be handled by the `prunesinglelinks()` function.

The most conservative and simplest approach is to consider every possible path, and to eliminate all constituent links that can't meet all of the requirements. Depending on the load in the network this may be overly restrictive. A more relaxed solution is to incorporate end-to-end resource availability as a subgraph property, although it may increase the likelihood of search failure later in the process.

Consider the example of Figure 4.6. The resource requirement for virtual link [A:B] is 1 unit, and for virtual link [A:C] is 3. A conservative elimination of single links would remove all relevant edges in the graph that are unable to support all possible demands at the same time. This would leave only one edge in the example: [B:C]. A more pragmatic approach in this case is to eliminate only those links that are unable to support *any* virtual link resource requirement, and only check that *all* requirements (for a single type of resource) are met for the candidate solution as a whole.

The `iscandidate()` function is invoked by the searching procedure to determine whether a potential VPN topology has the desired topology and also satisfies end-to-end subgraph requirements. The separate function `checksubgraphproperties()` can be used as an optimisation in the search procedure to eliminate partial solutions that have already exceeded the limit set for some cumulative resource attribute such as hop count.

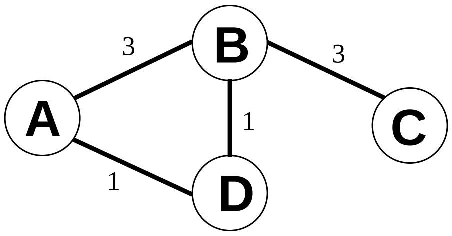


Figure 4.6: If virtual link [A:B] requires 1 unit of resource, and virtual link [A:C] requires 3 units of resource, then conservative elimination of physical links with available resource of less than 4 causes the search to fail unnecessarily.

As is explained in the following section, VSP policies combined with potential resource use by the VPN are used to derive edge weights for the physical topology graph. The function `getlink<resource>usage()`, where `<resource>` is a supported resource type such as `bandwidth` or `labelspace` is employed for this purpose.

4.4.2 VSP Policies and Edge Weights

The process of VPN design produces a switchlet specification from a VPN description, with the goal of finding the cheapest possible VPN that satisfies the requirements of both customer and VSP. The 'cost' of a given VPN is determined by a combination of the VPN's attributes, VSP policies and the current resource availability in the physical network. These factors are combined to derive the edge weights for the network graph used in the topology search.

The weights of the edges in the network topology graph are determined by a combination of factors:

- **Current resource availability.**
  Insufficient resource on any individual link will result in the elimination of that link from the graph.

- **The resources requested for this VPN.**
  Although the customer is charged for the resources actually reserved for a VPN, it may not be apparent how much resource is going to be used by the VPN on particular physical links until its topology is determined. Therefore, if the cost of a VPN includes a charge per unit of resource, a conservative estimate may be needed in order to assign an edge weight.

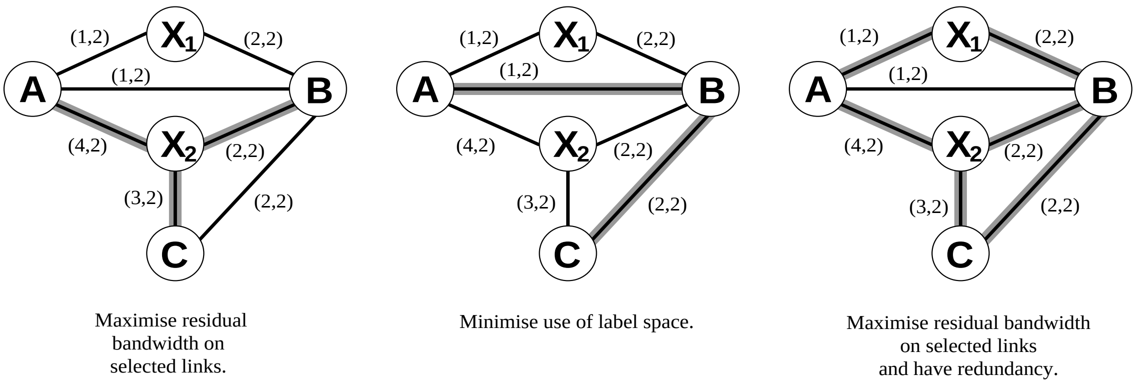


Figure 4.7: Three different VSP policies producing different optimal VPN topologies.

- **VSP policies with respect to both resource availability and resources requested.**
  VSP policy determines the manner in which the costs of resources are combined to give a single number for an edge weight. As discussed below, resources may be given different weightings to help the VSP attain its overall goals.

VSP policies may also be incorporated in a cost function `calculatecost()` that is used in the search algorithm (along with the other dynamically generated functions, such as `iscandidate()`, that were described earlier). Policies that contribute to the cost function will tend to cover charging schemes or time-of-day restrictions, for example discounts for a partner organisation or for weekend usage of the service. In contrast, policies that determine the way in which edge weights are calculated are more concerned with the relative costs of various network resources.

The policies that a VSP defines for the purposes of edge weight determination can be used to express, for example, that bandwidth 'costs' more than label space, perhaps because the VSP has only a few high-bandwidth customers. These policies are not fixed and will vary from one VSP to another, and may even alter over time at the one VSP. By recomputing the edge weights from current availability and current policy, these changes can be catered for.

As an example consider the network shown in Figure 4.7. The requested VPN consists of the nodes A, B and C, and the availability of bandwidth, bw, and label space, ls, is shown for each edge as a pair (bw, ls). Assume the desired VPN requires one unit each of bandwidth and label space. If the VSP favours the use of bandwidth where most is available, the topology produced will be the subgraph highlighted in the left-hand network, whereas the subgraph of the middle network is found if minimising the number of links is

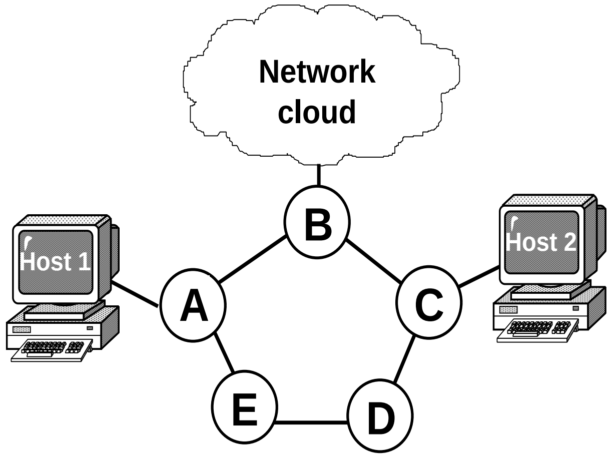


Figure 4.8: Example network where preferred VPN topology from Host 1 to Host 2 is different for VSP and customer.

given greater priority. If full redundancy together with maximum residual bandwidth is required, the subgraph highlighted in the right-hand network will be the result.

The VPN design process is also concerned with maximising the VSP's ability to satisfy future requests, whilst creating a VPN that conforms to the specification provided by the customer. Assuming the VSP's customers have roughly similar resource requirements, then the VSP may endeavour to spread the load of bandwidth, label space and switchlet allocation. Alternatively, given a different type of customer, the VSP may choose to maximise resource availability on certain links for *large* resource requests. As another example, Figure 4.8 shows a network topology where although a customer might prefer that their VPN from Host 1 to Host 2 passes through the shorter path A, B, C, the VSP might route the VPN through A, E, D and C in order to maximise the local resource availability for VPNs originating within the network cloud. The VSP policy expressing this preference will result in increased edge weights on the links into and out from B.

Note that even given a specific goal there may be trade-offs in making policy. In the context of QoS routing, to achieve high network throughput there is commonly a trade-off between conserving resources, i.e. selecting paths with the fewest hops, and balancing the network load by choosing links with the most residual bandwidth [Ma99].

Policies reflect the relative priorities in the local domain, and by incorporating policies in the edge weight computation, requirements for a particular VPN are balanced with the overall needs of the VSP.

4.4.2.1 Discussion

In the context of network routing algorithms there is some debate about whether link costs should be fixed or dynamic [Perlman00]. Fixed edge weights can be automatically calculated at start of day by measuring static characteristics of the network, such as delay, or can be assigned by the network manager according to known network characteristics together with the expected traffic matrix in the network. The latter option gives some scope for the network manager to input policy into the routing algorithm, but is potentially quite complicated. Edge weights can also be modified dynamically, by routers observing current usage and increasing the cost of a link as its utilisation increases. Dynamically changing edge weights can lead to problems of network convergence and disruption in routing and traffic flow, and also add to the amount of control traffic using network resources. However this is more responsive to changes in traffic patterns, and reduces the burden on the network administrator. It is argued that the gains in network capacity obtained with dynamic link costs do not justify the additional complexity and control overheads incurred [Perlman00].

This section has proposed the use of policy and current resource availability to generate edge weights dynamically for each and every search. Dynamic edge costs for VPN topology determination do not suffer from the above problems:

- The 'routing' employed by the VSP to find a VPN is completely distinct from the routing protocols used by a control architecture for connection establishment within a VPN. The edge weights only apply to the VPN search and have no other impact.
- As explained in Section 3.4, network control traffic is confined to a control VPN, with its own resource partition. The increase in control traffic that results from incorporating current resource availability has no impact on other traffic in the network. (Note that if network state is maintained in a centralised fashion, then increased control traffic is not an issue. Chapter 6 investigates the replication of the NetBuilder, and hence of the resource allocation state database.)
- The timescales for VPN topology search are much slower than those of packet forwarding, so oscillation and non-convergence happen slowly and can easily be identified and recovered from before causing problems.
- The resource 'usage' on a link in this case doesn't correspond to observed traffic, but to well specified resource allocations (i.e. switchlets). Therefore, although a search may still fail due to out-of-date information, there is no usage monitoring overhead.

For these reasons, it is feasible for the VSP to try to optimise the distribution of resource allocations based on dynamically calculated edge weights, whilst maintaining its ability to meet customer requirements.

4.4.3 Case Study: Search Space Construction

Once a VANDAL specification has been produced for VL as described in Section 4.3.3, it is parsed to generate the functions for constructing the weighted topology graph that is used in the search.

As explained earlier, when there are multiple requirements for a single virtual link that refer to the same resource type, there is a trade-off between a more restrictive VPN specification and a more complex search. Either the requirements can all be merged to give a larger, aggregate resource usage for that link, or else if multiple physical paths for that virtual link exist, then provided each individual requirement is satisfied by at least one physical path, then even links that do not have sufficient resource availability for all of the requirements are not pruned. However, the `checksubgraphproperties()` function must then ensure that the single link constraints as well as end-to-end properties are satisfied.

This situation arises in the VL case study. The link from the core switch to its neighbour D has sufficient capacity for the audio multicast tree, but not enough to transmit video. Potentially, the VSP could allow this link to be left in the graph, as the alternative physical path between C and D has sufficient unused bandwidth for the video tree. If it chooses to be more conservative then that link will be pruned—this behaviour is assumed, and in the resulting topology graphs of Figures 4.9 and 4.10 the link is dashed to indicate that it has been removed from further consideration.

To illustrate how VSP policies can be incorporated to influence edge weights, the topologies in Figures 4.9 and 4.10 show two versions of the VL network graph with different edge costs resulting from the application of dissimilar policies. In the graph of Figure 4.9, residual bandwidth is maximised, so links with less bandwidth available are more expensive, whilst the graph of Figure 4.10 favours links with larger free label space. The function `getlink<resource>usage()`, generated previously from the VANDAL specification, plays a part in deriving these edge weights.

Once underway, the `checksubgraphproperties()` and `iscandidate()` functions will be invoked by the search procedure. The only subgraph property defined for VL is that hop count from the core to each endpoint is limited to 5. This means that if the edge weights of Figure 4.9 are used, then the cheapest VPN topology (i.e. incorporating the nodes A, B, C, D and E), is one that has cost 16 (all links except B:C) but would violate

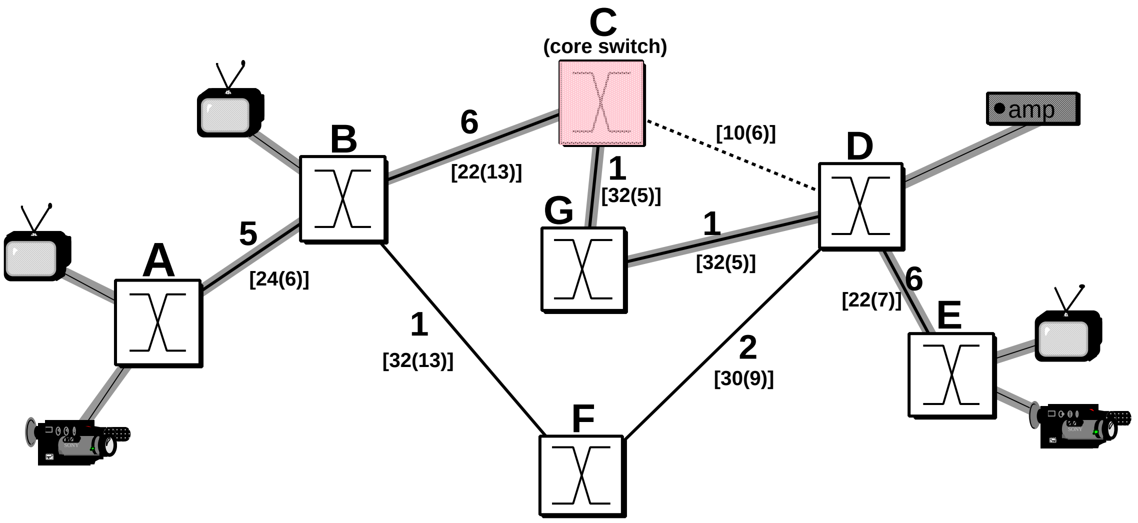


Figure 4.9: Case study network showing edge weights when residual bandwidth is maximised. Edge weights are calculated: (34 - available bandwidth)/2. The originally available resources on each link are labelled [bandwidth (label space)]. The cheapest VPN topology meeting the requirements for VL is shown in grey.

---

the hop count constraint. Thus the more expensive topology which excludes node F and costs 19 (shown in grey in the figure) will be returned by the search procedure.

When the different edge weight policy of maximising available label space is used, as shown in Figure 4.10, then the resulting VPN topology is as shown in grey in that figure, with cost 22.

4.5 Related Work

The VPN design process described in this chapter is related to other work on dynamically provisioned virtual networks, but is unique in that it explicitly tackles the resource management problem faced by the VSP. This section describes a selection of other approaches.

4.5.1 Darwin

Darwin [Chandra98] is a network resource management system in which virtual networks implement customised resource management policies. The system comprises four compo-

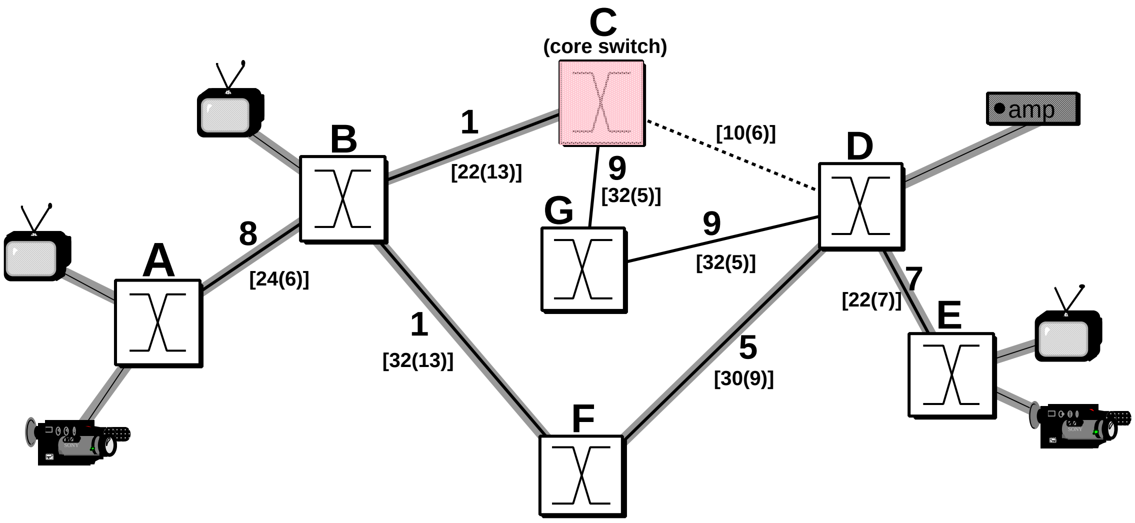


Figure 4.10: Case study network showing edge weights when available label space is maximised. Edge weights are calculated: (14 - available labels). The originally available resources on each link are labelled [bandwidth (label space)]. The cheapest VPN topology meeting the requirements for VL is shown in grey.

---

nents: Xena—a resource broker responsible for the global allocation of resources; Beagle—a signalling protocol that allocates the physical network resources; Java code segments running on routers for runtime resource management, known as control delegates; and hierarchical scheduling mechanisms for flexible sharing and isolation of resources among virtual networks.

Both Xena and the NetBuilder perform a similar role in the mapping of application or service-specific requirements to a virtual network topology and resource allocation. In contrast to the language-based specification of VANDAL, the resource request provided to Xena is in the form of a graph where the nodes represent services and edges the communications flows between services. Application-specific annotations constrain these requirements, for example by specifying the network address of where a given service is to be located, or by detailing the QoS parameters of individual flows. The problem is expressed as an integer linear program and the virtual layout is then determined using a solver package, with an explicit trade-off between quality of solution and computation cost.

The graph input to Xena allows a much larger degree of freedom than supported in VServ. In particular, Xena can map a description of the flow's semantic content, for example, "motion JPEG", to the required QoS parameters. Furthermore, Xena may also make use of application-specific knowledge to insert semantics preserving transformations into the network, for example transcoders to convert the JPEG flow to MPEG and back again. This can have benefits such as optimising resource usage in the network, but requires Xena to understand service semantics, thus adding complexity to the system. The explicit incorporation of application semantics into the network service infrastructure itself leads to the very problems that originally motivated devolved control: either the service provisioning software must constantly evolve to keep up to date with new services, or else the facilities to exploit the service-specific knowledge become redundant.

Beagle is a protocol that signals virtual networks, and as such is analogous to the NetBuilder. The virtual network specification received from Xena comprises a list of flows and control delegates. Beagle issues flow setup requests to nodes which specify link resources, such as bandwidth and buffers, together with how those resources are applied. It also instantiates delegates on nodes, after initialising the appropriate runtime environment and reserving the required amount of processor resources such as processor cycles and memory. Thereafter, delegates take care of local resource management of the flows under their control. A control API enables a delegate to monitor network conditions, shift resources between flows, send and receive messages and re-route flows.

Similar to VServ, the approach taken in Darwin is to support a service-oriented network by enabling customisation of both control and data planes. However, in contrast, a virtual network does not comprise a simple allocation of resources, but rather a set of flows and the means to apply resource management policies to them. Dynamic resource reallocation is not supported, and it is not clear how delegates can re-route a flow within a virtual network when the resource allocation of the network is defined by the existing set of flows.

4.5.2 The Genesis Kernel

Automated creation and deployment of programmable virtual networks is addressed in an architecture based on the Genesis Kernel [Campbell99], which *spawns* virtual networks. Prior to creation, a virtual network is described in a profiling script, which includes details of topology and resource requirements, among other attributes. A resource requirement for a link is expressed as an amount of bandwidth and one of three capacity classes: constant, controlled or best effort. As part of the process of verifying a profiling script, a database is queried in order to determine if sufficient resources are available, and policy servers are consulted for rules relating to QoS provisioning and admissibility of the specified virtual network.

The means of describing a VPN in Genesis is much less expressive than that allowed by VANDAL. It is not clear whether the format of the profiling script would allow additional resource types to be incorporated if appropriate. Nevertheless, it could be envisaged that the profiling script may well contain a VPN description written in a similar notation to VANDAL. Similar to the design process of VServ, Genesis includes a mechanism for taking into account service provider policies. However the details of how this policy server operates are not described, and the work does not acknowledge or address the inherent complexity of the search procedure.

4.5.3 Virtual Active Networks

In the Virtual Active Network (VAN) framework [Brunner99], VPNs are provisioned on demand within an active network infrastructure. In an active network, users can program the internal nodes to perform customised computation on packets traversing the network.

Customers of the VAN framework deploy and manage their own active services using active packets to install or upgrade service-specific functionality. Each VAN has a dedicated set of resources, including link bandwidth and node processing and memory resources, obtained through a provisioning interface.

This work is conceptually very similar to VServ, but makes use of an active network environment. It explicitly separates the VSP's management interface from the VAN management interface through which customers can exert fine-grained control and deploy customised network services. However, as with other active network architectures, it is not clear how resource partitioning is enforced, and there is a high performance penalty associated with using the data path (i.e. active packets) for control operations. Issues of resource management and distribution are not addressed.

4.5.4 The Hose Model

The *hose model* addresses customer characterisation of VPN requirements on IP networks [Duffield99]. It defines resource-assured VPNs in terms of the aggregate traffic to and from endpoints in the VPN and the associated performance guarantee. With a hose, the customer doesn't have to specify per-link traffic characteristics—a hose guarantees performance from a given endpoint to all other endpoints in the VPN. This results in simpler SLAs for the customer but more complex resource management for the provider.

The SLA for a VPN can be incrementally defined, starting with a rough estimate and re-negotiating after monitoring the VPN's performance (in terms of loss, delay, jitter etc.). SLAs can also make allowances for gradual changes in traffic load, or anticipated drastic changes. To ensure SLAs are met, the provider may police and/or shape traffic at hose ingress and egress points.

Similarly, the dynamic resource reallocation facilities of VServ give a mechanism for the customer to modify the topology or resource allocation of a VPN. The technique of allocating resources that approximate the desired VPN and gradually converging to a VPN with better characteristics is also used in VServ. As described in Section 5.1.1, this enhances responsiveness in the creation of lightweight or highly dynamic VPNs, where the time to carry out a search for a good topology may be disproportionate in relation to the VPN's complexity. It also explicitly allows user feedback to be part of the process of determining an appropriate resource reservation: provided the relationship between service and cost is predictable, the customer can use dynamic modification of their VPN to reach a satisfactory state.

The hose model addresses the resource management of VPNs, and explores the multiplexing gains that are possible with dynamic resizing. However the approach is not as flexible as that of VServ. In particular, fine-grained resource control, for example, service-specific routing such as the core-based trees used in VL, cannot be exerted by the customer, except in a limited fashion by changing the aggregate resource allocation of a hose. Furthermore, resource reservations to a VPN as a whole are not addressed, nor are issues of VPN topology search considered.

4.5.5 Bandwidth Brokers

The idea of the *bandwidth broker* was proposed as part of the DiffServ Architecture [Nichols99] to act as a centralised control plane entity responsible for domain-wide resource provisioning and admission control.

Zhang *et al* [Zhang00] propose the use of a bandwidth broker, together with a scheduling mechanism which uses the Virtual Time Reference System (VTRS) to support IntServ-style QoS in the Internet. In VTRS an edge traffic conditioning component does policing and shaping at the ingress router. At this point each packet is stamped with a packet virtual time stamp and with rate-delay parameters as determined by the bandwidth broker according to the flow's QoS requirements. Each router has an *error term*, i.e. upper bound on the delay to a particular packet given its QoS allocation. This means that the end-to-end delay bound of a path can be computed, and since the per-hop behaviour of a core router is characterised by its error term, any scheduling algorithm can be used provided it can be described with respect to the error term.

The bandwidth broker maintains the state of all flows, including QoS reservation and traffic description. It also keeps router information, such as bandwidth, buffers, scheduler, error term and current QoS reservation, together with path information such as number of hops, propagation delay, minimum residual bandwidth and the sum of the router error terms on the path. The scalability of the bandwidth broker state is addressed with coarse-grained class-based services, and the specific problems introduced by allowing *dynamic* flow aggregation are considered. Other issues such as bandwidth broker federation and distribution, flow routing, signalling, inter-domain QoS reservation and SLAs and support for statistical QoS guarantees are not addressed.

This system encounters many of the same problems as VServ, and the bandwidth broker performs some of the same functions as the NetBuilder. A significant difference is that there is an implicit assumption of a single, generic control plane, and the bandwidth broker must therefore keep track of every flow in the network. The VTRS could potentially act as a supporting technology for the Tempest, similar to both ATM and MPLS in current implementations. However because traffic policing is only carried out at edge routers, the control operations exported in the core of the network might have to be more restrictive—for example, it is not clear whether flows could be merged without exceeding the resource reservation in downstream routers.

4.5.6 NetScript

NetScript [Yemini96] is a scripting language for active networks which allows network node functions to be programmed on streams of packets. Delegated agents are dynamically dispatched to routers or switches, where they execute programs written in the NetScript language—described as the Postscript of network devices. Agents make use of standard libraries which provide access to local resource management, routing, scheduling and transmission functions, and can communicate with remote agents using a proprietary middleware layer.

Packet streams belong to virtual networks, which are defined as collections of node engines (i.e. execution environments) connected by virtual links. A NetScript header on each packet identifies the virtual network to which it belongs. Similar to VServ, NetScript supports a form of programmable virtual network, but there is no support for resource partitioning either in the data plane or in the control plane. The programmability of NetScript is oriented towards protocol implementation: in VServ this role is played by the control architecture of a VPN. NetScript does not address the global resource management issues that must be considered for a resource-assured, programmable VPN service.

4.6 Summary

This chapter was concerned with the management and administrative aspects of VServ. An automated VPN design process was introduced that attempts to make the system usable, to incorporate local policies and resource usage goals, and to improve performance of the search procedure that finds a suitable VPN topology and resource allocation.

The design process falls into three distinct phases. The first of these is VPN description, during which user requirements are expressed as network-level QoS parameters. The mapping of user-level requirements to network QoS parameters can be difficult, often requiring expert knowledge. A `configure` tool is proposed as a way of supporting sophisticated QoS specifications. It comprises control architecture-specific resource templates that are filled in by the user.

The output from `configure` is a VPN description in a simple notation for VPN topology and resource requirements called VANDAL. A VANDAL specification associates resource types, which are defined as appropriate for the local domain, with subsets of the topology.

The second phase of the design process is the construction of the search space. This involves parsing the VANDAL specification to derive single link constraints, which are then used to prune individual edges from the physical topology graph. The VPN specification also gives rise to subgraph properties—end-to-end constraints for the desired VPN—as well as identifying compulsory members of its topology. These are incorporated into the search procedure to assess when a candidate topology has been found.

VSP policies are used to assign edge weights for the topology graph used in the search. Such policies may include traffic engineering considerations, such as load balancing, or policies to make scarce resources more expensive. The calculation of the cost of a candidate VPN topology can also be tailored to incorporate administrative policies such as charging agreements.

The third and final phase is the search itself. This is the subject of the following chapter which will explore the selection of an appropriate search algorithm in order to further customise the search procedure and mitigate its complexity.

Chapter 5

VPN Topology Search

The previous chapter described the VPN design process for VServ in which a graph representing the underlying network topology is constructed with input from the VPN description provided by the customer, VSP policies and the current resource availability in the network. The VPN description is parsed, and code is generated that is then incorporated into the search algorithm at run time. The goal of the search procedure is to find a topology that meets the requirements for the VPN without being unduly costly. If successful, it produces a set of switchlet specifications for the new VPN. This chapter focuses on the the selection of an appropriate algorithm for the topology search in order to mitigate the inherent intractability of the searching problem.

Section 5.1 presents the issues involved in the selection of a search algorithm. In the general case, finding the best topology is computationally expensive. However certain types of VPN topology are amenable to the use of heuristic algorithms that have been proposed elsewhere. The categories of VPN that are suitable for certain search algorithms are identified, and a selection of such algorithms is described.

In Section 5.2 the general case is considered, for which there are no suitable polynomial time algorithms. It is argued that the application of even simple heuristics to the brute-force search can make the procedure feasible. Experimental experience is presented to support this view.

Examples of the VPN design process as a whole are described in Section 5.3. Finally in Section 5.4 inter-domain searching is discussed and related work presented.

5.1 Selecting the Search Algorithm

In a highly dynamic VPN service the performance of automated VPN creation is crucial. The time taken to determine the topology and resource allocation of a new VPN should not be disproportionately larger than its expected lifetime, nor should the algorithm employed compromise the VSP's ability to provide the desired service. A search algorithm that consistently gives poor results will only hinder the VSP in making efficient use of network resources, and a slow response time will deter customers who wish to deploy a lightweight VPN with a short duration.

The general problem has exponential complexity, as explained in Section 5.2, and Section 5.2.2 presents results from an implementation of the naïve solution that further demonstrate its impracticality. Thus it is desirable to identify cases where good quality heuristic algorithms may be applied.

The search for an optimal VPN topology can be expressed as the following graph problem, in which the subgraph S corresponds to the desired VPN topology:

> Given a directed, weighted graph $G = (V, E)$, with set of vertices V and set of edges E, and given a subgraph $S = (N, M)$, $N \subseteq V$, $M \subseteq E$, what is the minimum cost connected subgraph of G containing S, subject to certain path-specific and global constraints?

Four factors influence the selection of the searching algorithm:

1. The length of the VPN's lifetime.
   Below a certain duration an expensive search for a good topology cannot be justified, since it would be of no benefit to the VPN owner and it would not affect the resource allocations across the network for a significant length of time. Therefore when a short-lived VPN is designed, the search algorithm used should be fast, with the quality of the result of secondary importance.

2. Whether the VPN topology—i.e. the subgraph S—is a tree (a graph that does not contain any cycles).
   Unless explicitly stated otherwise by the customer, a VPN is assumed to be fully connected, which is the usual case for a communications network. However the capability to construct a service-specific VPN, together with the facility to tailor the topology of the VPN to suit the service, will lead to the adoption of tree topologies.

3. If the topology is a tree, then whether it is *source-based*.
   Source-based trees are point-to-multipoint directed trees, in other words a tree constructed of paths from a single source to multiple destinations. An example of a network service that would make use of a source-based tree topology is a distribution service. Source-based trees can be contrasted with *group-shared* trees. The latter would be constructed for a multipoint to multipoint service, for example video conferencing, where gather-distribution trees are constructed for reasons of scalability and efficient use of network resources. A Core Based Tree (CBT) multicast architecture [Ballardie93] is an example of a service that would make use of a group-shared tree.

4. Whether any subgraph properties have been specified.
   Recall from Section 4.1 that subgraph properties are either end-to-end constraints for the paths in the VPN topology, such as delay or jitter, or properties that apply to the VPN's topology graph as a whole, for example "no endpoint more than 5 hops away from a designated core router".

These characteristics allow four categories of VPN to be identified for selection of the search algorithm:

Category A: Lightweight or highly dynamic VPNs.

Category B: A tree with no subgraph properties defined.

Category C: A source-based tree with subgraph properties defined.

Category D: Remaining tree topologies and all cyclic topologies.

Categories A, B and C are suitable for the use of polynomial time search algorithms, as discussed in the following sections. A brute-force approach to the search is warranted for Category D VPNs. However as will be shown in Section 5.2, the application of some simple heuristics makes even this tractable for reasonably sparse networks.

The actual selection of the search algorithm is straightforward. The presence or absence of subgraph property specifications is already evident from the VANDAL parsing procedure. Whether the topology is cyclic can be determined using a topological sort, which has the useful side effect of identifying a source vertex, required for some tree searching algorithms.

5.1.1 Category A VPNs

The nature of the topology search problem means the searching procedure is likely to be computationally expensive. When a VPN is likely to frequently modify its topology or resource allocation then this could render the initial searching cost unnecessary. Thus a time-consuming search is not warranted for highly dynamic or short-lived VPNs.

A solution is to initialise Category A VPNs with a pre-configured topology—the simplest example being a resource allocation (i.e. a switchlet) on every node in the underlying network. Another reasonable approach is to use a fast and simple algorithm such as the minimum spanning tree combined with shortest path algorithms to add additional edges as required. In the case of VPNs that are anticipated to be highly dynamic, a topology that is considered more *resilient* should be chosen. A resilient topology in this case is one where frequent changes do not significantly degrade the cost of the VPN. A possible method of finding a resilient topology is to favour core nodes with larger degree.

Work on routing for multicast has considered the problem of dynamic group membership and the corresponding degradation in the quality of the routing tree. An example is the ARIES algorithm [Bauer97] in which a part of the multicast tree is rearranged when it has deteriorated beyond a certain threshold. The deterioration is measured by a *degradation factor* which is the ratio of the cost of the modified tree to the cost of the corresponding optimal Steiner tree. The rearrangement is limited to the immediate vicinity of the update.

A similar approach can be taken in VServ to trigger a reconfiguration of the VPN's topology, depending on the nature of the control architecture concerned. There are some particular considerations that arise in VServ which are not found in multicast routing:

- When the core of the VPN topology is exposed to the control architecture, that is, when the control architecture holds Ariel interface references for nodes inside its topology, then the topology cannot be rearranged in a way which excludes those nodes. Therefore, although reconfiguration may be desirable for a highly dynamic VPN, the VSP must take care to keep track of the switchlets for which the control architecture currently possesses a handle. Recall from Section 3.2 that a resource reservation at a node always results in a switchlet, but handles on the Ariel interfaces are only given to the control architecture when explicitly requested in the VPN specification.

- When connections have been established in the region of the topology undergoing reconfiguration, care must be taken to maintain service on those connections, which may be achieved by temporarily over-provisioning for the VPN. A trade-off can be made between the cost of the rearrangement and the potential for a better quality VPN.

- The resource revocation protocol described in Chapter 7 allows the control architecture to maintain service to applications whilst the underlying topology is reconfigured. Hence, uniquely, VServ provides a framework in which a gradual convergence to a better VPN can be performed without disrupting applications.

Although there is a strong argument for treating Category A VPNs differently in the topology search phase, the most appropriate way to do so is determined by a combination of factors, many of which are specific to the particular environment in which VServ is running. Such factors include the characteristics of the physical topology of the network, the nature of commonly provisioned VPNs, and the requirements of other VPN customers. In the light of its own environment and circumstance, the VSP must decide *which* VPNs fall into Category A, as well as how to determine a sensible initial topology and to what extent subsequently to reconfigure that topology. Fortunately, the dynamic resource reallocation facilities of VServ, in particular the revocation protocol, support this functionality whilst mitigating ill effects.

5.1.2 Category B VPNs

In a Category B VPN, i.e. when the topology is a tree and the customer has *not* specified any subgraph properties, the searching problem is the same as that of finding a Steiner minimal tree (SMT). The SMT problem is well known: the goal is to connect a set of vertices in the graph by finding a minimum weight tree that can also use any of the remaining vertices. When the tree is source-based, this is the problem of determining an optimal multicasting tree in networks. Several heuristic algorithms for finding a SMT have been proposed and analysed, of which a selection is described in this section.

The **KMB** heuristic [Kou81] obtains a solution based on the minimum spanning tree (MST) of the nodes in S. It starts by constructing a fully-connected graph containing S, where the weight of each edge corresponds to the sum of the weights on the shortest path between its endpoints in G. The MST of this graph, T_1, is determined. Next the shortest paths in G are swapped back for the edges in T_1 to get G_s, and finally the MST of G_s is found, and edges deleted as necessary so all the leaves in the tree are in S. The running time is polynomial—$O(|S| \cdot |V|^2)$—and it is shown in [Kou81] that the resulting tree has cost at worst twice the optimum.

Another well known SMT algorithm is the **TM** heuristic [Takahashi80], which progressively adds nodes to the solution tree one at a time. At each step it chooses the closest node to the tree that is not already included, and connects it by the shortest path between it and any existing node of the tree. Similar to the KMB heuristic, TM has complexity $O(|S| \cdot |V|^2)$ and results in a tree that is no more than twice as bad as the minimal solution.

The **RS** heuristic [Rayward-Smith83] differs from KMB and TM in that it attempts to identify *Steiner points*, i.e. the nodes outside the required set that are included in the optimal solution. It starts with a forest of trees, each tree comprising a single node from S. A heuristic function is used to choose a node n in close proximity to at least one of these trees. The two closest trees to n are then joined by a shortest path. These steps are repeated until only one tree remains. RS has complexity $O(|V|^3)$, and has been shown to give closer to optimal solutions than KMB or TM [Rayward-Smith86].

As pointed out in [Salama96], the RS algorithm was designed for undirected graphs, and an efficient implementation for an asymmetric network is not readily apparent. In general, VPN topology determination is required to operate over directed graphs, that is, networks with asymmetric links. Ramanathan [Ramanathan96] has proposed a polynomial time algorithm for finding a directed SMT called Selective Closest Terminal First (SCTF).

SCTF allows the user to trade-off running time against tree cost by means of a tunable parameter, k. Similar to TM, it starts with a single node and then at each step extends a branch of the tree to include the closest new node in S. The running time is $O(|S|^2 \cdot k + |E|)$, where k can range from 1 to $|V|$. When k is small, i.e. tuned to decrease running cost, the new path is allowed to pass through vertices already in the tree. It is shown that the algorithm subsumes both KMB and TM, depending on how k is set. When $k = |V|$, the algorithm is equivalent to TM, and when $k = |S| + 1$, the algorithm is equivalent to KMB. It is also shown that the degree of asymmetry of the graph dictates how badly the algorithm will perform. If the amount of asymmetry is bounded, then the solution is proved to be no worse than a constant times the optimum (however it is shown that for arbitrary graphs, it is as unlikely as $P = NP$ that a polynomial time algorithm exists that performs better than a constant times optimum).

Because it has been designed to operate over directed graphs, the SCTF algorithm is the most appropriate choice for searching for Category B VPN topologies. Figure 5.1 contains the pseudo-code of SCTF, with s being the designated source node of S, and k the tunable parameter.

```
SCTF(s, k, S)
  V' = {s}, E' = {}, Q = s  # Q is a priority queue
  while S not empty do
     n = min(k, |Q|)
     B = first n vertices in Q
     z = shortest path from any of the vertices in B
         to any of the vertices in S
     branch = the path from w to z where only w is already in V'
     add vertices in branch to Q, V' and E'
     remove vertices in branch from S
  return G' = (V', E')
```

Figure 5.1: SCTF heuristic algorithm for finding an unconstrained Steiner tree.

5.1.3 Category C VPNs

For a Category C VPN, where the topology is a source-based tree and the customer *has* specified one or more cumulative subgraph properties, determining the optimal topology parallels the computation of a QoS-constrained Steiner tree. In such a tree, the routing is constrained by factors such as maximum delay variance along the paths to the receivers. Although it is possible that the requirements for a VPN topology are less stringent, it may still be advantageous to make use of one of the proposed heuristic algorithms.

Many of these algorithms concentrate on two factors particularly relevant for multicast—edge cost and delay. The KPP algorithm [Kompella93] is one such algorithm. KPP attempts to find a minimal cost tree with end-to-end delay on any path from the source to a destination bounded by a constant value Δ. It first calculates the constrained cheapest paths between all pairs of nodes in S, and then uses a greedy approach to construct a constrained spanning tree of S. Finally the edges in the spanning tree are replaced with the equivalent paths in G and loops removed. In [Kompella93] two heuristic functions are considered for choosing an edge in the constrained spanning tree: the first minimises the cost without considering the delay (which is already less than Δ), and the other chooses cheap edges while maximising residual delay. The first is shown to perform slightly better in terms of resulting tree cost, although the two converge as Δ increases (i.e. the delay bound becomes larger than the actual path delay). It is difficult to characterise the absolute performance of the heuristics. Results for the second heuristic are presented only in comparison with the first for graphs up to 100 nodes, with the first compared only to the optimal solution for graphs up to 20 nodes in size. The running time is $O(|V|^3 \cdot \Delta)$ for either heuristic.

Another constrained SMT heuristic for source-based trees is presented in [Sun95], henceforth referred to as SL. The algorithm initially computes a low cost spanning tree T_1 using Dijkstra's shortest path algorithm and keeps only those paths that satisfy the delay constraint. Secondly it constructs a minimum delay tree that spans any remaining vertices in S not already in T_1, again using Dijkstra's algorithm. The two trees are then combined, and any loops eliminated by removing edges from T_1. The running time of SL is $O(|V|^2)$. Its performance is compared to the KPP heuristic, and it is found to produce solutions between 15 and 30% worse than KPP, although still better than a simple least delay solution.

Salama *et al* [Salama97] evaluate and compare these and a number of other source-based multicast routing algorithms. As expected, SL is found to give consistently worse results than KPP in terms of both cost and maximum end-to-end delay, but runs significantly faster.

The choice between these two algorithms may be thought of as analogous to the way the k parameter of the SCTF algorithm for unconstrained trees is used. The priorities of the VSP, together with the circumstances of the individual search, will influence the selection of appropriate algorithm for this category of VPN topologies. Other factors such as extent of edge cost asymmetry and degree of variation between edge costs have not been analysed for these algorithms, but may also play a part in their performance characteristics.

5.2 The General Case

Based on currently popular network services, in particular content distribution, many VPN topologies are expected to fall into one of Category A, B or C as defined above. Nevertheless a significant proportion will not, and these can only be dealt with using an algorithm for the general case. This section explores the inherent complexity of the general case problem, and observes how performance can be improved with a simple example heuristic.

An implementation that finds the best subgraph (i.e. VPN topology) from a sufficiently large graph is computationally expensive because of the large number of candidate subgraphs. If no subgraph properties are specified, and there are no cycles (i.e. a Category B VPN), then finding the optimal VPN is equivalent to finding the minimum Steiner tree, which is known to be NP-complete [Garey79].

Although it could be argued that a hand-crafted searching procedure, tailored to local topology and policy, would be sufficient for automated VPN design, a more general solution is desirable for a number of reasons:

- portability across different network topologies;
- the ability to accommodate changes in topology, including short-term changes, for example caused by a temporary failure in the network; and
- the quality of the results—the expertise required to hand-craft an optimal search procedure may not be available.

5.2.1 Experimental Overview

The following sections describe experimentation which was carried out with the aim of gaining some *insight* into the brute-force algorithm for Category D VPNs. Since no polynomial time heuristic algorithms are known for the general case of VPN topology, the feasibility of this type of search is an important consideration for the practicality of VServ as a whole. To this end, this chapter does not tackle the problem of finding a good heuristic algorithm for the general case, but rather, investigates the practical implications of the theoretical complexity of the problem.

Two questions are addressed by the experiments:

1. How bad in practice is the theoretical intractability of the brute-force search? In other words, how large or dense is a non-trivial network for this problem?
2. Is it feasible to apply a simple heuristic to the brute-force search?

In order to ensure the correct functioning of the brute-force algorithm, and to obtain an optimal solution, henceforth referred to as OPT, a control experiment was carried out. The results of the heuristic algorithm described in Section 5.2.3 are compared with those of OPT.

The control problem used for OPT was the computation of the minimum spanning tree (MST). MST is a restriction of the general case in four aspects: the resulting subgraph always spans the entire physical graph, i.e. the VPN that is returned covers all network nodes; no specific edges are required; no subgraph properties are specified; and the resulting topology is a tree. The quality of the results obtained with the brute-force and heuristic algorithms described in this section can be compared with MST only when these four conditions apply. This simplification is not intended to be representative of the general problem, which is arbitrarily complex. However for the purposes of the analysis presented in this chapter it is convenient to be able to assess the quality of the results obtained.

Prim's MST algorithm was used for the control experiment, and implemented using a binary heap as the priority queue.

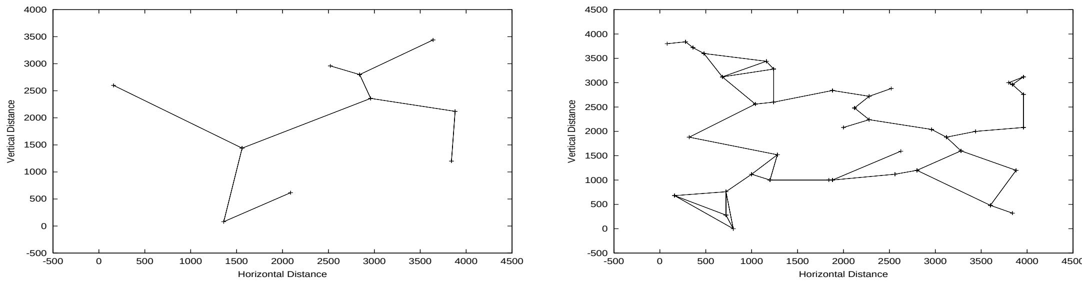
5.2.1.1 Setup

The simulations were carried out first using a naïve solution, and then repeated with the incorporation of a simple heuristic. The observed performance of the brute-force searches further motivates the need for practical searching heuristics. Both the performance figures and the results (which, when obtained, are optimal for the limited problem domain detailed above) were then used to gain some insight into the benefits and drawbacks of applying a heuristic search to the general case.

The programs used in this evaluation were written in Python [Python]. In using a scripting language for performance analysis there is a trade-off between ease of implementation and capability for rapid prototyping, and the inherent lack of efficiency. Therefore the performance figures presented in this chapter should only be regarded as indicators of comparative performance. The execution platform consisted of a 500 MHz Pentium PC, running Linux.

The search experiments were carried out over topologies taken from a set of 4800 produced using the TIERS [Doar96] random network topology simulator. In TIERS network generation, the redundancy at a node is assigned according to the uniform random distribution from 1 to the specified redundancy parameter. Thus the number of edges is at least one less than the number of nodes, with additional edges introduced randomly according to the specified redundancy. Examples of networks from the set are shown in Figure 5.2. The number of nodes in the topologies ranged from 5 to 100 inclusive and the degree of redundancy ranged from 1 to 5. 10 networks of each size and degree of redundancy were produced, hence every data point plotted represents an average over 10 randomly generated topologies of identical number of nodes and degree of redundancy (the actual number of edges may differ).

Even though the problem definition applies to directed graphs, in order to be able to use MST as a control experiment, the networks in the TIERS topology set are undirected. Both of the searching algorithms presented in the following sections operate by performing



10 nodes, redundancy 1

40 nodes, redundancy 3

Figure 5.2: TIERS randomly generated networks.

a breadth-first search on the graph, therefore undirected topologies do not impose an additional computational burden over directed ones. Since the purpose of the experimentation is to demonstrate the feasibility of the VPN search process, rather than to present a viable practical solution, it was felt that the benefits of being able to assess comparative performance *and* the quality of the results outweighed the slightly unrealistic data set provided by undirected topologies.

5.2.2 The Naïve Approach

A brute force method of determining the optimal VPN topology is to construct every possible connected subgraph that satisfies the given VPN description, calculate the cost of each and then choose the cheapest. The pseudo-code of Figure 5.3, described further below, shows an algorithm that takes this approach. Note that it incorporates the function `iscandidate` that was automatically generated from the parsing of the VANDAL specification, as described in Section 4.4.1. It also includes the function `calculatecost` which implements specific charging policies of the VSP as described in Section 4.4.2.

Let S denote the set of nodes in the VPN. For any node $v \in S$, `singlehops[`v`]` contains all paths from v to each of the other nodes in S, that do not pass through any other node in S and do not contain any cycles. The construction of `singlehops` uses a modified form of the Floyd-Warshall transitive closure algorithm to generate one possible set of paths between VPN nodes, and then takes the transitive closure of the resulting paths to obtain *all* possible paths. Note that the term `singlehops` refers to single *virtual* edges—a virtual edge may encompass more than one physical edge.

The algorithm to find the minimum cost subgraph containing S is shown in Figure 5.3. The procedure is initially called with an empty subgraph and `singlehops` is pre-computed.

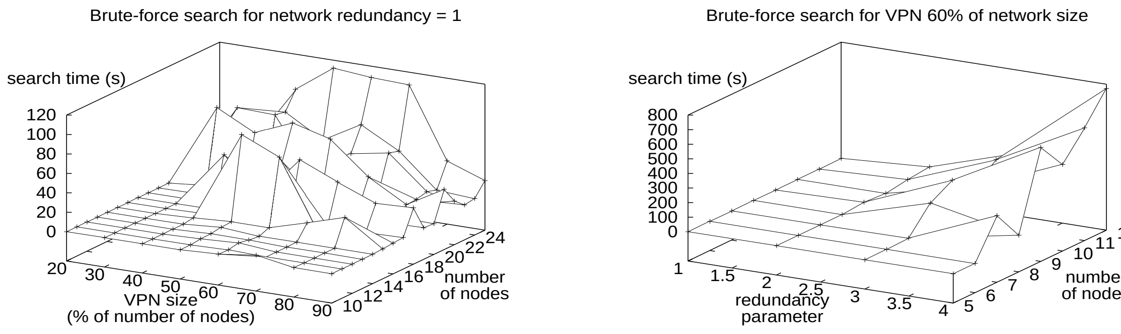
```
MINSUBGRAPH(subgraph)
1   if iscandidate(subgraph) then
2      c = calculatecost(subgraph)
3      if c < mincost then
4         mincost = c
5         minsubgraph = subgraph
6   else
7      paths = singlehops[subgraph.last]
8      for each p in paths do
9         MINSUBGRAPH(subgraph + p)
```

Figure 5.3: Brute-force algorithm to find a minimum cost subgraph.

At line 7 whichever VPN node was added most recently to the subgraph is taken, and its paths to other VPN nodes looked up in `singlehops`. The procedure is then called recursively for each of these paths. Note that a subgraph is costed only if it is a valid candidate, that is only if it comprises the desired VPN topology and meets all of the specified constraints. At line 8 processing is also carried out to ensure that continuously cycling paths are not considered, and to guarantee termination within a reasonable time period, the search is abandoned after a certain (large) number of recursions.

Let m be the number of edges in the graph G and let k denote the number of nodes in the VPN. The number of paths between any two nodes without cycles is at most 2^m, and for any node v there are paths to at most $k-1$ other nodes in the VPN. Therefore the maximum number of iterations of the loop at line 8 is at most $(k-1)$, with each iteration recursing at most 2^m times. As expected, the brute-force algorithm is computationally intractable.

A comparison of the costs of subgraphs obtained using brute-force searching and of spanning trees from the MST algorithm confirms the correct operation of the naïve algorithm. Execution of both algorithms over the entire topology database gives identical cost results (although not necessarily identical subgraph topologies) except when the brute-force search was abandoned (in order to ensure that the experiment terminated). The point at which a search is terminated was arbitrarily chosen to be 2^{20} iterations of the `MINSUBGRAPH` procedure. A total of 370 topologies were searched using the brute-force algorithm, with the number of nodes ranging from 5 to 12 and redundancy degrees from 1 to 5. 57 searches failed to finish within the maximum number of iterations allowed. The complexity of the



(a) Different VPN proportions on sparse networks.

(b) Single VPN proportion on increasingly dense networks.

Figure 5.4: Performance of brute-force search.

brute-force algorithm precluded checking correctness on larger networks.

The graph in Figure 5.4 (a) shows how badly the brute-force algorithm actually performs in practice. This experiment is run only on topologies with a redundancy parameter of 1 (i.e. sparse networks). It indicates the extreme deterioration in performance as the graph size, and the proportional VPN size, increase. At networks containing more than 15 nodes and the VPN covering more than about 60% of the network, the search time increases dramatically.

Problems are also caused for brute-force searching by increases in graph density, as shown by the graph in Fig. 5.4 (b). This graph shows the search time for a VPN covering a fixed proportion of the physical network—60%, with redundancy parameter increasing from 1 to 4. Search times increase substantially once the graph size exceeds 7 nodes. Clearly the brute-force approach is unlikely to be adequate for any network of reasonable size or density.

5.2.3 Heuristic Approach

The intention of the heuristic approach considered in this section is *not* to derive an algorithm of sufficient quality to be usable in practice, but rather to verify that the procedure itself is feasible, and to expose promising approaches for improving on the brute-force search. The investigation of a polynomial time heuristic algorithm for this task is a good candidate for further work, but more in-depth exploration is beyond the scope of this dissertation. Therefore analysis is restricted to one simple heuristic with the aim of demonstrating that substantial improvements can be made without undue penalty.

The heuristic algorithm stops searching on finding a candidate topology and it tries to ensure that this is a reasonably good result by ordering the search. Figure 5.5 contains the pseudo-code for the ordered search algorithm. The search proceeds as for the naïve algorithm, with two additional steps. At line 4 an exception is thrown when a result is found, which terminates the search procedure. At line 7, a sorting procedure referred to as `ORDERBYHEURISTIC` is called, and this determines the next partial subgraph that is examined.

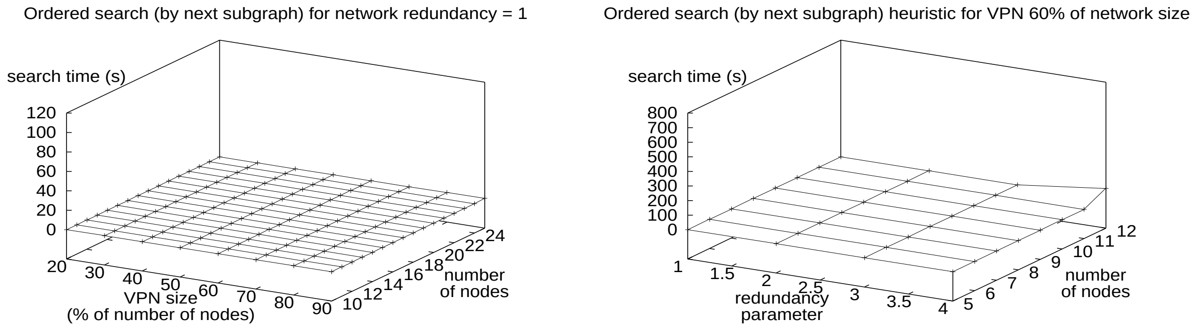
```
MINSUBGRAPH(subgraph)
1   if iscandidate(subgraph) then
2      mincost = calculatecost(subgraph)
3      minsubgraph = subgraph
4      raise ResultFoundException
5   else
6      paths = singlehops[subgraph.last]
7      orderedpaths = ORDERBYHEURISTIC(paths, subgraph)
8      for each p in orderedpaths do
9         MINSUBGRAPH(subgraph + p)
```

Figure 5.5: Ordered search heuristic algorithm to find a first candidate subgraph.

Similar to the manner in which code is generated for pruning single links, checking subgraph properties and getting the VPN topology as described in Section 4.4.1, code can be generated for the `ORDERBYHEURISTIC` function here from the VANDAL specification. For example, if a maximum hop count has been specified, then it would be appropriate for the paths to be ordered by increasing number of links. The ordering procedure used for this experiment orders the subgraphs as a function of their cost, calculated by adding a path to the current subgraph under consideration and sorting the paths according to the costs obtained with `calculatecost`, from cheapest to most expensive. Not all VPN descriptions will have subgraph properties specified. Nevertheless, a default ordering is found to give substantially improved performance. Assessment of the efficacy of the heuristics is carried out by comparison with OPT.

The graphs in Figures 5.6 and 5.7 show the performance of the heuristic. In Figure 5.6 the performance of the heuristic is plotted with the same axes as the graphs of Figure 5.4 showing the performance of the brute-force algorithm. The performance of the search is markedly improved.

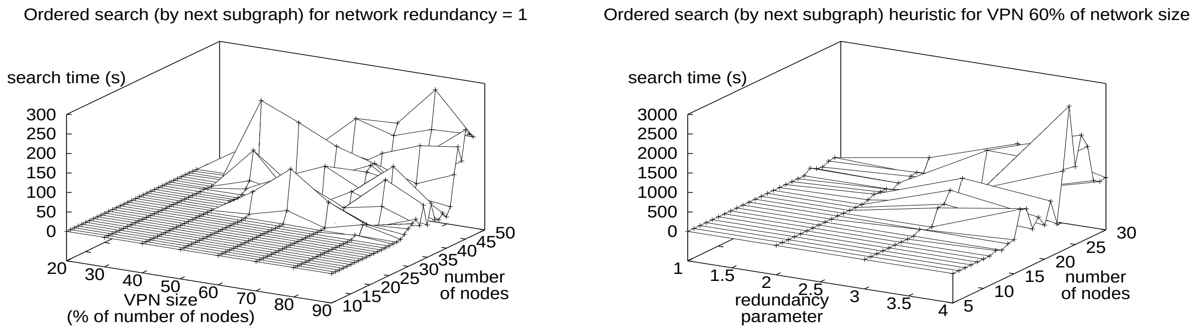
Figure 5.7 contains results with larger and denser networks, indicating the extent to which use of the heuristic might be considered reasonable. In Graph (a) of Figure 5.7, the number of nodes on the z axis ranges from 10 to 50 compared to a maximum of 25 nodes on the z axis of the graph showing the brute-force searching times over the same data set. Similarly, as shown in Graph (b), the heuristic allowed data to be collected for searches of dense networks of up 30 nodes. The same ranges of results are not available for the brute-force search because the search tended to exceed the maximum number of iterations before terminating.



(a) Different VPN proportions on sparse networks.

(b) Single VPN proportion on increasingly dense networks.

Figure 5.6: Performance of the ordered search heuristic, plotted on the same axes as the graphs of Figure 5.4.



(a) Different VPN proportions on sparse networks.

(b) Single VPN proportion on increasingly dense networks.

Figure 5.7: Performance of the ordered search heuristic.

As expected, the performance improvements are accompanied by a reduction in the quality of the results. The MST algorithm is used as a measuring stick for this decrease, in the particular case where it applies. The bar chart in Figure 5.8 indicates the proportion of results that are suboptimal, and also the extent to which those results are worse than OPT. The experiment was run over all topologies with number of nodes ranging from 5 to 20, and degree of redundancy from 1 to 4. The x axis is the number of nodes in each topology and the y axis is the percent of topologies. With 10 networks of each size and redundancy combination, the number of topologies represented by each vertical bar is 50. The 'badness' is indicated by the normalised percentage that the discovered topology cost varies from the OPT cost, with 100% badness being the sum of every edge in the graph.

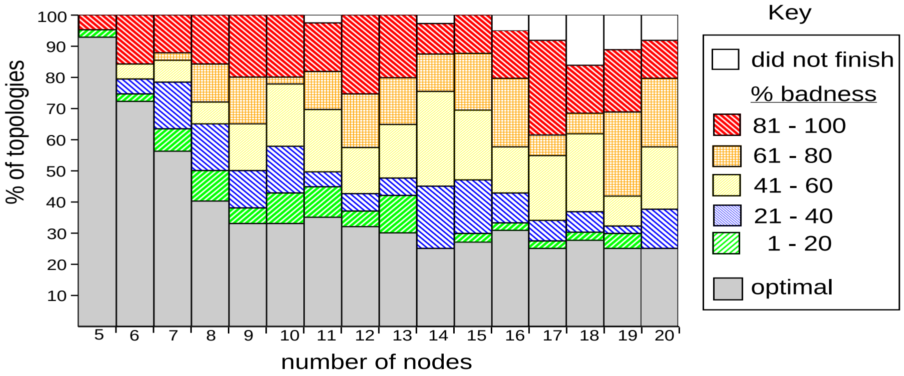


Figure 5.8: Optimality of the ordered search heuristic.

The histogram shows that this particular heuristic works optimally for only about 30% of topologies. When the result is suboptimal, the variation from the optimal result depends not only on which edges were chosen for the VPN topology, but also on the variation of edge weights within the sample topologies. In the set of 4800 which was used for all the experiments described in this section, the minimum edge weight was 5, the maximum 3925. 73% of all edge weights were less than 500, and 55% of edge weights were between 200 and 500.

The experimental evaluation reported here demonstrates that even with a slow, non-optimised implementation of the heuristic search algorithm, reasonable performance can be obtained for sparse networks up to about 30 nodes. Careful engineering and the use of compiled code would undoubtably improve this situation markedly.

This experimental analysis has assessed the performance of the brute-force search and a simple heuristic in comparison with the MST problem. Although considering only a simplification of the general problem, some insight is gained into the feasibility of the

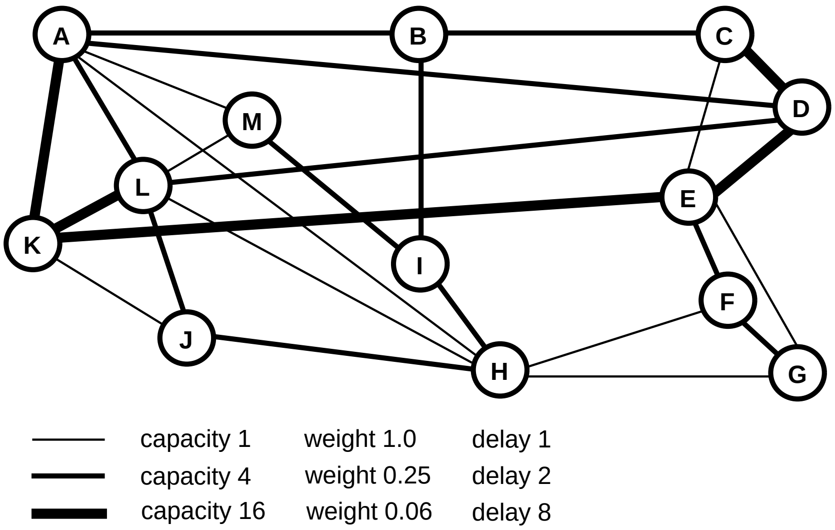


Figure 5.9: Example network topology.

---

general case VPN topology search. The penalty for introducing a heuristic algorithm is that the resulting suboptimal results mean that the customer can pay more for their VPN and that it is harder for the VSP to achieve its overall goals. However both gain substantially from search speed increases, and this benefit alone is likely to justify the use of a heuristic.

5.3 Examples of the VPN Search Process

The preceding chapter described the VPN design process in which the characteristics of the desired VPN are used to reduce the complexity of the search procedure by pruning the network graph and by customising the search algorithm. In this chapter, the selection of an appropriate search algorithm was explored in more detail. Certain categories of VPN are amenable to employing polynomial time heuristic algorithms that have been developed to address similar problems in slightly different situations. When a brute-force search must be performed, it was observed that even a simple heuristic can improve its running time, without inordinately compromising the quality of the results achieved.

In this section, two examples of the VPN design process as a whole are presented. They make use of the example physical topology shown in Figure 5.9, which resembles that of a real ISP. The edge weights are simply the inverse of capacity, each link size has a fixed delay, and for clarity the graph is undirected.

```
[A, C, E, G, J, K ] : [A:K, K:J, K:E, E:C, E:G]
bandwidth=4 : [K:J, E:C, E:G]
bandwidth=8 : [K:E]
bandwidth=12 : [A:K]
```

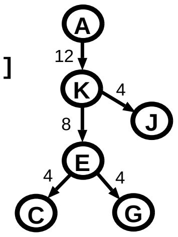


Figure 5.10: VANDAL specification of a VPN for a content distribution service, together with a schematic of the desired distribution tree topology and bandwidth.

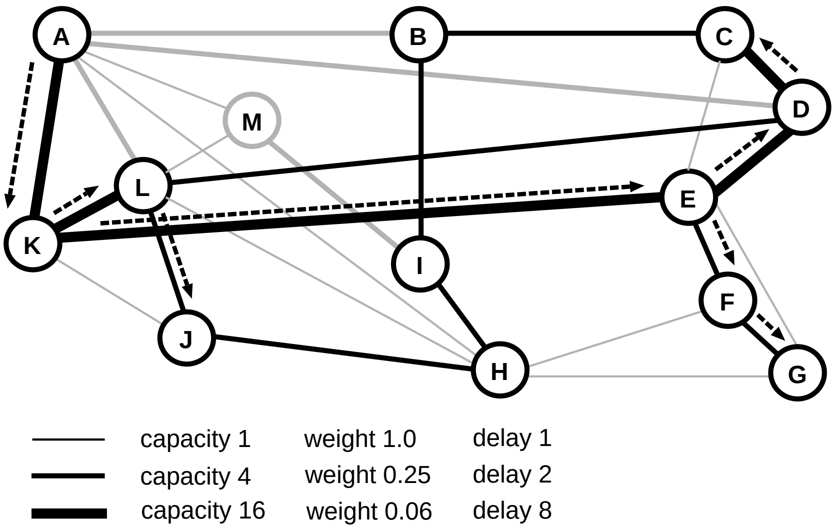


Figure 5.11: Example network topology with links and nodes pruned for the content distribution VPN greyed out. The dashed arrows indicate the discovered VPN topology.

The first example is a VPN required for a content distribution service. The VANDAL specification, shown in Figure 5.10, indicates that the service provider (i.e. the customer) has explicitly requested the distribution tree topology illustrated. Parsing of the VANDAL specification causes edges in the graph with insufficient capacity to be pruned—shown greyed out in Figure 5.11. The node M and its single remaining link to I are also pruned, as this portion of the topology is now redundant. Note that now the graph is significantly sparser, from 26 edges to just 12 edges. Since the VPN topology is a tree with no path constraints, an SMT algorithm is selected to carry out the search. The dashed arrows in Figure 5.11 indicate the result, which in this case is the least cost topology.

A second example is a VPN for an event monitoring notification service. Once again a tree topology is required, with the same root node (`A`), and leaf nodes (`C`, `G`, `J`), as

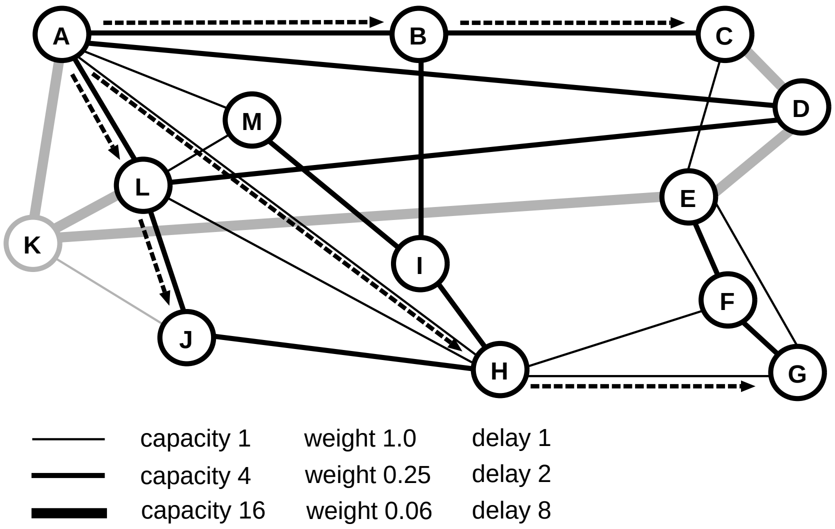


Figure 5.12: Example network topology with links and nodes pruned for the event monitoring notification VPN greyed out. The dashed arrows indicate the discovered VPN topology.

---

for the content distribution VPN. In this case bandwidth is not the critical requirement, but rather bounded delay. Assume for simplicity that the VANDAL specification bounds end-to-end delay on the virtual links between the root and each leaf node: `delay < 8 : [A:C, A:G, A:J]`. Also assume that delay is additive on multiple links. Now the graph is pruned to eliminate the high latency edges, along with the subsequently redundant node K. Furthermore, when the VANDAL specification is parsed, code is generated for the search algorithm that prunes the search tree whenever a *path* is constructed during the search that exceeds the delay bound. This VPN topology falls into Category C, and the graph is therefore searched using a QoS-constrained multicast tree algorithm. Figure 5.12 shows the pruned edges in grey, and the discovered topology with dashed arrows.

5.4 Inter-Domain Searching

An inter-domain reservation protocol that allows the provision of resource-assured VPNs spanning more than one VSP domain is an important complementary mechanism for VServ. However there are a large number of issues in the reservation of resources across multiple domains with the purpose of achieving a common SLA. Concerns include the need for a common VPN specification and inter-domain signalling protocol, charging mechanisms, monitoring, penalties for non-compliance and failure recovery. Although it is beyond the scope of this dissertation to address inter-domain searching in any depth, this section motivates future work in this direction for VServ by describing two existing inter-domain resource reservation protocols used in different contexts.

5.4.1 PNNI

Private NNI (PNNI) [ATMF96b] is a link state routing protocol for ATM designed to scale to large numbers of network nodes. It organises the network into a hierarchy of *peer groups*. Within a peer group, nodes flood the network with QoS and reachability information. Each node maintains resource availability and reachability information about all of the other nodes in the group. Aggregated reachability information is exchanged between peer groups by group leaders, elected by the nodes within a group.

The link metrics and attributes supported by PNNI include delay, jitter, cell loss ratio and available cell rate, together with some second order statistics. As in the search for a VPN topology in VServ, PNNI takes into account both single link QoS parameters, and additive link metrics that are summed along individual links to give a measure for an entire path.

When a connection request is signalled, the ingress node calculates a route according to its current information about the network state. It then signals along that path, and intermediate nodes perform local connection admission control (CAC) to establish the connection. The generic CAC (GCAC) algorithm is used to to predict the 'typical' CAC behaviour of other nodes, using a small number of metrics. Individual nodes can control the results of the GCAC algorithm by modifying the degree of laxity in their broadcast metrics, and this allows for some variation in the CAC implementations by switch vendors.

The ingress switch uses GCAC to determine a signalling route as follows [Alles95]:

1. Links that do not have sufficient resource availability are pruned from the set of all possible paths.
2. A shortest path computation is used to find a set of possible paths to the destination.
3. Paths are pruned according to end-to-end characteristics such as delay.
4. One of the remaining paths is chosen, and the signalling request is forwarded along it.

The ingress switch will inevitably have an imperfect view of current network state due to delays in propagating state information, as well as loss of information resulting from aggregation and hierarchical summarisation. Because of this, the local CACs along the path will not always succeed, therefore PNNI supports *crankback*. If the connection request is blocked, then it is rolled back to an earlier node in the path, which attempts to find an alternative route using the GCAC algorithm. The network state should now be newer, and hence potentially more accurate, thus improving the chances of success.

The complexity of the PNNI link parameters means that there is a chance that no possible paths will be found in step 2 of the GCAC algorithm. A technique called *fallback* allows particular attributes, for example delay, to be selectively relaxed and the path then re-computed.

There are similarities in the functioning of PNNI to the VPN design process described in this dissertation. In PNNI, the ingress node uses its knowledge of current state within its peer group to calculate a source-routed path along which the signalling request is forwarded. Likewise, in VServ a VPN creation request causes a topology to be determined, and, as will be described in the next chapter, the NetBuilder then attempts to reserve the appropriate resources in the network.

PNNI does not attempt to find the optimal path, but will choose any acceptable path. Although the problem of optimal path determination is easier than that of finding the best VPN topology, it would not be sensible for PNNI to attempt this. One reason for this is simply that PNNI is designed to scale to large networks with hundreds or thousands of nodes. However the most important consideration for a signalling protocol is responsiveness. Unlike the relatively coarse-grained timescales seen by VServ, PNNI is concerned with individual connections. The signalling to establish the connection is pure overhead, as no data transmission can take place until it is complete, and as such signalling performance is crucial.

Because the allocation of resource aggregates that is carried out by VServ is expected to occur less frequently, and with less stringent expectations for responsiveness, it is possible to accommodate more demands. Thus the design process described in this and the previous chapters incorporates both the QoS requirements for the VPN as well as taking into account VSP policies.

A signalling protocol similar to PNNI could be used to allow VServ to offer VPNs that span multiple domains. In particular, some means of representing the aggregate resource availability of an entire domain would greatly simplify the topology search problem. A higher-level VPN search algorithm could then be applied to the domain aggregates, with each VSP dealing with the topology search within their local domain as desired. It may be the case that many multi-domain VPNs will only require a route *across* intervening networks. In this case, a signalling protocol to determine an acceptable path, similar to

PNNI, would be sufficient.

5.4.2 Border Gateway Reservation Protocol

Border Gateway Reservation Protocol (BGRP) [Pan00] is a protocol for inter-domain resource reservations for the Internet. It works by aggregating the reservation state of each autonomous system and constructing a sink tree for each domain that contains either an originating or terminating router. The tree is formed using the shortest paths from each sink to the destination as determined by the standard inter-domain routing protocol BGP (Border Gateway Protocol) [Rekhter95]. BGRP messages are exchanged between adjacent border routers in order to determine network resource availability and to signal the setting up of reservation paths. Each backbone router only maintains reservation state information for each autonomous system, giving substantial scalability improvements over alternative proposals for the Internet such as RSVP. As in RSVP, the reservation state at each router is soft, with periodic refresh messages transmitted between routers.

Similar to the current Tempest implementation, BGRP currently only reserves bandwidth for each traffic class, ignoring other resources such as buffers. It relies on DiffServ traffic classification at edge routers for the enforcement of QoS in the data path. Other optimisations, such as over-reservation and accounting for hysteresis, can also be employed.

As is the case with PNNI, BGRP could be used to establish inter-domain reservations for VServ. However, the aggregation policy used by BGRP may not be appropriate. This is because reservations for aggregates that include connections from different VPNs could potentially result in interference between individual flows within the aggregate. Therefore, use of BGRP for establishing inter-domain VPNs in VServ may require replacement of the aggregation policy, and the maintenance of a larger amount of reservation state.

An advantage of an inter-domain reservation protocol such as BGRP is that since it makes use of BGP—the standard Internet inter-domain routing protocol—BGRP provides a realistic mechanism for deploying instances of VServ on the Internet. In domains that are not Tempest-aware, but that do support BGRP, sufficient resources could be reserved to carry the data of a single VPN from one VServ domain to another without violating the desired QoS for that VPN.

5.5 Summary

The search for a VPN topology is known to be intractable in the general case. This chapter has considered how this can be made workable by classifying VPNs according to their characteristics and applying different algorithms according to the resulting classification. Whenever applicable, heuristic polynomial algorithms developed in other contexts are used. It was also demonstrated that even simple heuristics can improve the computational performance of even the general case search. The feasibility of the VPN design process as a whole was illustrated with some realistic examples, and the issues inherent in inter-domain searching were discussed.

In the next chapter the actual reservation of resources in the network is considered. It is argued that there is a need to support concurrent VPN creation, and therefore issues that arise when the NetBuilder is replicated within a single VServ domain must be addressed.

Chapter 6

A Replicated Network Builder

The two previous chapters described the design process used in VServ for VPN creation. This chapter investigates the replication of the NetBuilder—the component that performs both VPN design and the corresponding resource reservation in the network in response to a creation request. Implementing multiple instances of the NetBuilder greatly enhances the availability of VServ, but adds complexity and other overheads. This chapter explores the difficulties of deploying a replicated NetBuilder and the inherent trade-offs.

In the next section the motivation for replicating the NetBuilder is presented. Section 6.2 describes the model and assumptions for the design of the replicated NetBuilder. The distributed creation of a VPN is described in Section 6.3 and a discussion concerning the cost of replication is in Section 6.4. Related work is briefly reviewed in Section 6.5.

6.1 Motivation

Within VServ, the NetBuilder is responsible for VPN creation, deletion and modification. The absence of a NetBuilder in the Tempest framework does not make the physical network unavailable: once a VPN is in existence, its control architecture is able to manage and use its resources as desired, and it is possible to manually establish VPNs by setting up switchlets at the appropriate Dividers. However, as was argued in Chapter 4, dynamic provisioning of VPNs is an essential part of the service offered by VServ, and so the NetBuilder is considered an important component.

The two steps of VPN creation are VPN design, as described earlier, and the subsequent reservation of resources at the appropriate Dividers[1]. Ideally, VPN creation is carried out by a single, centralised entity because VPN design requires up-to-date knowledge of the state of resource reservations throughout the network. In addition, resource allocations must be exclusive to a VPN and the VPN as a whole should be created with 'all or nothing' semantics. However, a centralised NetBuilder suffers from some serious drawbacks:

Bottleneck: Chapter 5 highlighted the potentially high cost of the search for a VPN topology. The possibility of blocking all other VPN creation requests behind one which takes a long time to service is not desirable. Furthermore, some VPNs may affect mutually exclusive regions of the network and these requests could be serviced concurrently without penalty.

Lack of scalability: A VServ domain is not intended to scale to a large number of network nodes due to the complexity of the VPN design process. However, there are no inherent limitations on the number or arrival rate of VPN creation requests. Although the bottleneck of the previous point may be mitigated by careful allocation of incoming requests to instances of the replicated NetBuilder, a single NetBuilder restricts the volume of requests that can be responded to in reasonable time. This in turn limits the number of customers and the rate of service requests they can be allowed to make.

Vulnerability: The centralised NetBuilder is a single point of failure. A problem with its host machine could halt any further dynamic provisioning in VServ. A replicated NetBuilder gives a more available service, and even if the physical network becomes partitioned, may allow VPN control operations to continue in the remainder of the network.

Inflexibility: Multiple NetBuilder nodes allow a more flexible configuration of the Tempest components than possible with a single NetBuilder. For example, a 'plug and play' version of VServ could be possible by deploying a NetBuilder (along with a Divider) on the physical network devices themselves. Although the resulting coupling of hardware and network control software is not generally considered desirable, it might be advantageous in particular cases to have a stand-alone VServ which does not require any processing capability external to the physical network devices.

For these reasons, in spite of the resulting increase in complexity and overheads, replication of the NetBuilder is essential in a practical deployment of VServ. In the following section the model and assumptions for the replicated NetBuilder are explained.

[1] Although this dissertation mostly considers VPN creation, many of the concerns also apply to deletion and modification.

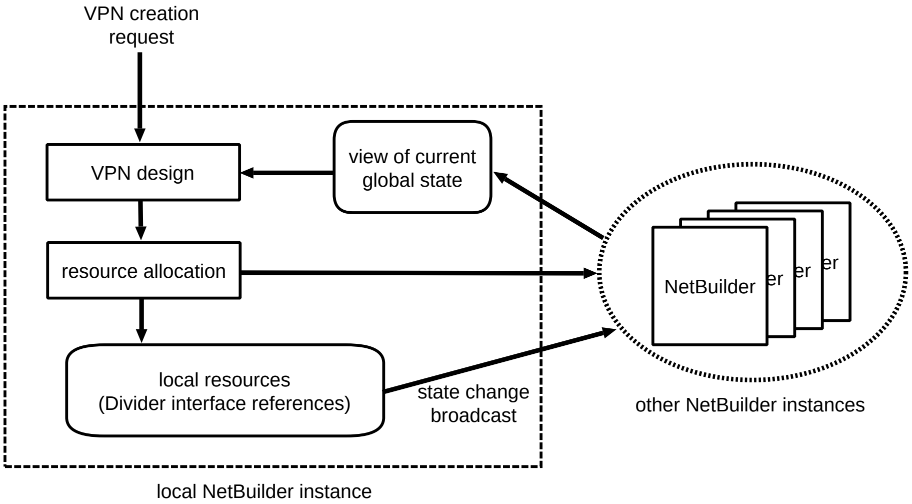


Figure 6.1: Internal structure of an instance of the replicated NetBuilder.

6.2 Model and Assumptions

Each instance of the NetBuilder is responsible for the switchlet management operations on one or more 'local' Dividers. It also has a view of current global state, i.e. the switchlets currently in existence throughout the network. Local state change operations are broadcast to all other NetBuilder nodes so they can update their views of global state. Figure 6.1 depicts the internal structure of a replicated NetBuilder node. The diagram shows a VPN creation request arriving at the NetBuilder. There are two stages of processing for this request: firstly VPN design is performed as described in Chapters 4 and 5 using the NetBuilder's current view of topology and resource allocations throughout the network. Secondly resources are allocated at the appropriate Dividers according to the resulting switchlet specification. This involves creating switchlets at the NetBuilder's local Dividers, and also passing the request on to other NetBuilder nodes as required. For clarity, the diagram does not show the switchlet handles being passed back to the initiator of the creation request, nor the information flowing in the reverse direction—for example, an update at the local Dividers also causes the local view of global state to be updated, and failure of the resource allocation stage may cause further iterations of the design stage.

When the NetBuilder is replicated, it might appear sensible to integrate its tasks into the Divider, thus eliminating a level of indirection. However, as will be shown in this chapter, distributed VPN creation is an expensive and complicated operation. The Divider's presence on the control path introduces an additional cost for each control invocation; therefore extra functionality beyond its primary task of policing should not be introduced unless absolutely necessary. Nonetheless, in extreme situations it may be considered desirable to integrate the two components, for example, as mentioned in Section 6.1 where both NetBuilder and Divider are deployed *on* the switch. Furthermore, there is a trade-off between performance and degree of concurrency which is discussed further in Section 6.4. Therefore the model proposed here does not impose any restrictions on the extent to which the NetBuilder is replicated, allowing each instance to be responsible for resource reservation on an arbitrary number of Dividers.

It is assumed that communication between NetBuilders is reliable, ordered and that messages are delivered at most once. It is also assumed that there is a known and finite delay on the links, and that if a reply is not received within an appropriate timeout period that the remote node has failed. Although nodes themselves may crash and recover or even fail completely, such aspects of distribution have been extensively studied elsewhere and it is assumed that well known failure and recovery techniques [Weihl93] are employed by the NetBuilder for these situations.

Ariel invocations for connection establishment and removal use port, label space and label to identify and define the connection. However, in order to allow dynamic modification of existing VPNs, possibly requiring that label allocations are changed dynamically, the actual value of the labels assigned to a switchlet must be hidden from control architecture. Therefore it is assumed that a mapping from the control architecture's value for a label to the real value used in the physical network takes place in the Divider at the same time as control invocation policing. This label space indirection also requires a remapping to the real labels in the data path when hosts have the ability to label packets (as in the case of ATM networks), albeit only at the edges of the network.

Finally, an ordering is defined for the NetBuilder instances in the network. Each NetBuilder can be identified by a unique number that also indicates whether one NetBuilder comes 'before' or 'after' another, for example using the MAC address. The assignment of numbers to nodes is assumed to have been done at start of day by a network administrator.

6.3 VPN Creation

This section describes how distributed VPN creation takes place in the replicated NetBuilder. Section 6.3.1 is concerned with the VPN design process, and Section 6.3.2 with the reservation of resources at Dividers.

6.3.1 VPN Design

The VPN design process requires the NetBuilder to have current knowledge about the state of resource reservations throughout the network. After a VPN is a created, the new network state is advertised to every NetBuilder node. The mechanics of determining a *consistent* view of global network state from the collation of messages sent by different nodes at different times is a well studied problem in the field of distributed systems. There are a number of protocols for distributed snapshots [Babaoğlu93], which are often used for distributed deadlock detection or debugging. The replicated NetBuilder makes use of one of these solutions.

When the NetBuilder is replicated, an out-of-date view of the global network state may result in a poor quality or even invalid VPN topology being determined. This issue is important due to the high cost of VPN design. However the demands for system responsiveness and scalability mean that locking the entire network for the duration of a VPN creation request, i.e. serialising the requests, is not a practical approach. Therefore an additional broadcast message is introduced. Before commencing a search a NetBuilder broadcasts a `searching` message containing the desired VPN topology and resource allocation, together with the transaction identifier. Transaction identifiers represent global timestamps, and are discussed in Section 6.3.2.

On receipt of the `searching` message from a remote node, a currently searching NetBuilder has a number of options:

- If the transaction identifier is greater (i.e. the remote node started searching more recently), then the message can safely be ignored. The ordering introduced by the use of transaction identifiers gives priority to the older request.
- If the transaction identifier is smaller and there is a potential conflict of resources, then the NetBuilder has three choices:
  - It may optimistically proceed with its search in the hope that the earlier search may not succeed.
  - It can restart the search using a modified network state that reflects the anticipated future resource allocation in anticipation that the other search will succeed.
  - It can abort the search, possibly restarting after receipt of the state update that is broadcast once the other search is completed, provided it is successful.

The choice of action to take will depend on whether the NetBuilder has sufficient knowledge to *estimate* the likelihood of a remote search succeeding and whether it is able to *predict* the probable resource usage of the remote VPN at relevant places in the network. In addition, considerations such as the duration or complexity of the desired local VPN will play a part in deciding whether to abort an ongoing search.

Given sufficient real world experience, in particular a model of incoming requests, it would be possible to refine this procedure and improve the VPN admission control decision. Similar to the case of circuit-switched networks, the blocking probability for a call request depends on factors such as the duration of the call and the routing algorithm used [Walrand96]. The problem is harder in VServ where VPN topologies are more complex than single connections, but VServ has the advantage that subsequent reconfiguration of a VPN's topology and resource allocation is possible in a non-disruptive manner through the resource revocation described in Chapter 7.

6.3.2 Resource Reservation

Allowing multiple VPN creation requests to be serviced in parallel not only results in the possibility of VPN design taking place with imperfect knowledge of current state, but may also cause attempts to reserve resources at the Dividers to conflict. The resource allocation at multiple points on behalf of one VPN must be *atomic*, that is, it must be an 'all or nothing' event. A partially created VPN is unlikely to be useful and wastes resources. This requires fairness of servicing requests: some notion of global ordering is needed, and preemption of ongoing requests may be necessary.

The atomicity requirement for VPN creation is addressed in the replicated NetBuilder by implementing the operation as a distributed transaction. The two-phase commit protocol (2PC), referred to by Gray as "contract law applied to computers" [Gray93] is a well known technique for enforcing atomicity. In the first phase participants in the transaction vote on whether the transaction should be committed or not, and the votes are collected by the designated transaction manager. In the second phase, if all the votes are yes then the manager instructs all the participants to commit the transaction, and otherwise to abort. The distributed 2PC makes use of a timeout period to ensure that node or network failure does not compromise the atomicity of the transaction. If any of the transaction participants vote against, or do not respond within the timeout, then the manager unilaterally causes the transaction to be aborted.

Costs that result from using 2PC are caused by the number of messages sent between NetBuilders and the time to process them. Transport of NetBuilder messages takes place using the control VPN as described in Section 3.2.1. This gives the replicated NetBuilder a dedicated share of network resources, and hence communication between NetBuilder nodes does not affect the QoS received by customer VPN traffic. However, there is still an overhead associated with the transmission, receipt and processing of those messages. 2PC is a blocking protocol, with resources being 'pinned down' during the voting phase, and held until the transaction is aborted, or, if the transaction completes, until explicitly released at a later time. This may prevent other transactions from making progress. Section 6.4 discusses the costs associated with NetBuilder replication.

The receipt of a VPN creation request from a customer implicitly designates the receiver as the *coordinator* for this request. The first task of the coordinator is to establish a globally meaningful start time for the transaction. This can be thought of as 'taking a ticket' for the request, and is necessary to ensure that a transaction is not starved and will eventually be serviced. The transaction timestamp is combined with the coordinator's own unique identifier to produce a transaction identifier referred to as *tid*. The transaction identifier also serves as a globally unique VPN identifier once the VPN is in existence.

The exact nature of the mechanism used for global timestamping is not significant. A possible alternative is a global real-time clock, such as that obtained with a synchronization protocol like NTP [Mills91]. Another is to use logical or vector clocks [Lamport78, Babaoğlu93] which are based on the observation that within a distributed system a partial ordering of events exists, in that some events are known to happen before others because there is a causal effect. Which scheme is preferable depends not only on the availability of a reliable reference time source, but also on the frequency of VPN creation requests versus the frequency of clock synchronisation updates. It is in the VSP's interests to minimise the network resources used for management and control operations, which in this case refers to the number of messages used to exchange timing information.

An overview of the steps performed by the coordinator is as follows:

1. Receive VPN creation request.
2. Get transaction identifier *tid*.
3. Broadcast `searching` message.
4. Perform search, giving the switchlet specification list.
5. Run 2PC for the switchlet specification list.
6. Broadcast the local state change, if any.

The version of 2PC used by the NetBuilder extends the classical 2PC protocol with an extra round of messages in which NetBuilder nodes send switchlet handles to the coordinator for the newly created switchlets after the transaction is committed. If the switchlet handles are not received within the timeout period the coordinator will abort the transaction, causing all the earlier switchlet creation operations to be be undone.

A further consideration for distributed VPN creation is the allocation of overlapping label spaces at adjacent Dividers where required. Recall from Section 3.2 that the resource partitioning of a switch or router in the underlying network is handled by a single Divider. Within the Divider, the switch or router is modelled as a set of incoming ports and a set of outgoing ports. Each port may correspond to a single neighbour (as in an ATM switch), or may be associated with more than one neighbour (e.g. in a router on a broadcast medium such as Ethernet). This model forms the basis of the label space partitioning method: a switchlet comprises a subset of the available ports, and for each port it has a number of labels. Connectivity between adjacent switchlets is enabled by having the label space of the outgoing port overlap with the label space of the incoming port on the neighbouring switchlet, therefore two NetBuilders responsible for creating adjacent switchlets in a VPN must agree on a shared label space.

A nested transaction is used for this adjacency negotiation involving client-server style communication between the NetBuilder pairs which are responsible for switchlets that require overlapping label space. Since each NetBuilder node has a unique, countable identifier, an arbitrary convention of the lower-numbered node acting as client and hence initiating the transaction is used to avoid deadlock. Every node may be involved in multiple node-pairs, and may be acting as both client and server in the different negotiations.

6.4 Discussion

This section discusses the cost of replicating of the NetBuilder with some observations based on experimental results.

The degree of replication in the NetBuilder refers to the number of Dividers managed by each NetBuilder. This affects not only the robustness and availability of VServ, but also the amount of concurrency that the system supports.

An implementation of 2PC was made in order to concretely consider the effects of various factors. In this simulation of VServ, resource capacity is assumed to be infinite, with the focus being on the costs of communication and locking. The physical network topology used was the realistic ISP's topology, introduced for the searching examples of

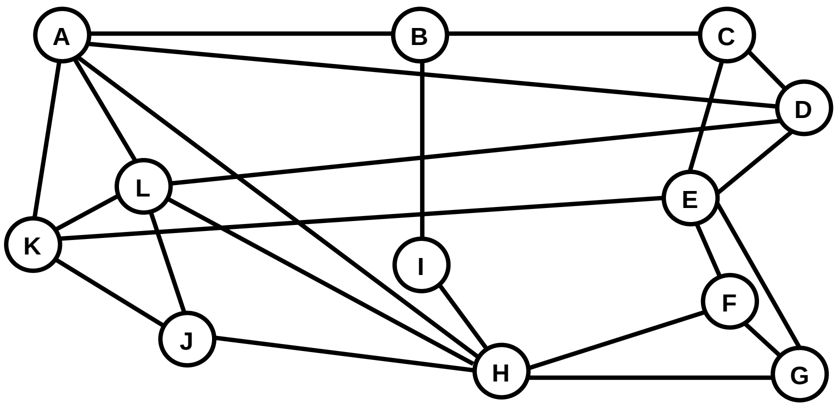


Figure 6.2: Example network topology.

Section 5.3. For convenience, the number of nodes used was 12 (rather than 13). The topology is shown in Figure 6.2.

The NetBuilders for the simulation were written in Python, and the execution environment comprised four 500 MHz Pentium PCs running Linux.

The simulation architecture consists of a *request server* which hands out VPN switchlet specifications from a pre-determined list, cycling through the list as needed. Each NetBuilder acquires a VPN request from the server, carries out the appropriate local resource locking, and then initiates the 2PC protocol for the other NetBuilders involved in the request. When a transaction completes, the coordinator sends a message to the request server. Each NetBuilder logs its own actions, and the request server also logs the overall durations of transactions.

The graph of Figure 6.3 shows that, as expected, there is a non-trivial cost associated with replication. On the y axis is the time taken so far, and the transaction number is on the x axis. The graph shows three different amounts of replication for the NetBuilder operating with the same set of VPN creation requests, serviced in the same order. For this experiment each requested VPN encompasses every single node in the physical network. This means that every NetBuilder is involved in every request, even in the case of maximum replication when 12 NetBuilders are running.

It can be seen that the request throughput for a single NetBuilder is markedly higher. However it should be emphasised that this experiment is only assessing the overheads for resource reservation and does not take into account the improvements that may be gained from concurrent executions of the VPN search process.

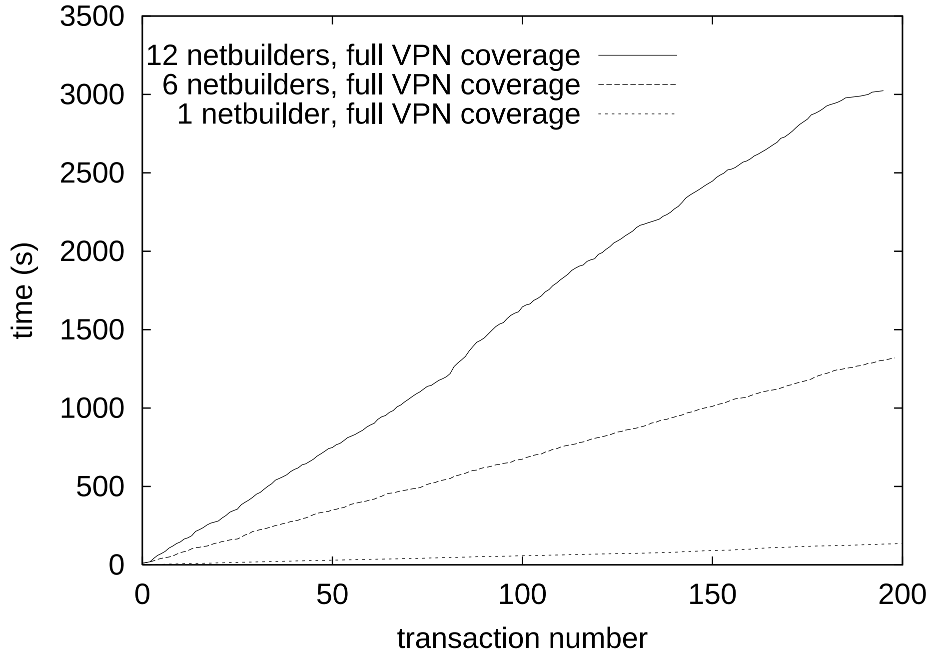


Figure 6.3: Increasing transaction completion time for different amounts of NetBuilder replication.

As the number of NetBuilders increases, not only do the transactions tend to take longer, but the amount of variability between transaction completion times increases. This is due to the blocking that occurs because of contention for locks, and becomes more unpredictable with more NetBuilders. The time taken for each transaction to complete for the case with 12 NetBuilders from the same experiment is shown in the scatter plot of Figure 6.4, where the y axis gives the duration of the transaction and the x axis the transaction number. An analysis of the causes of the peak at transaction 82 reveal that it occurred because the NetBuilder acting as coordinator for that request also dealt with 11 other transactions (in response to requests from other transaction coordinators) while carrying out the processing for transaction 82. The average number of transactions carried out by other NetBuilders concurrently with their 'own' transaction was between two and three.

The pattern of network coverage of the VPN requests also affects the performance achieved by the replicated NetBuilder. VPNs that require the participation of fewer NetBuilders are created faster than those involving more communications and remote locking. In the graph of Figure 6.5, which shows the case for 3 NetBuilder instances, it can be seen that the 'clustered' VPN topologies which involve fewer of the NetBuilder nodes tend to

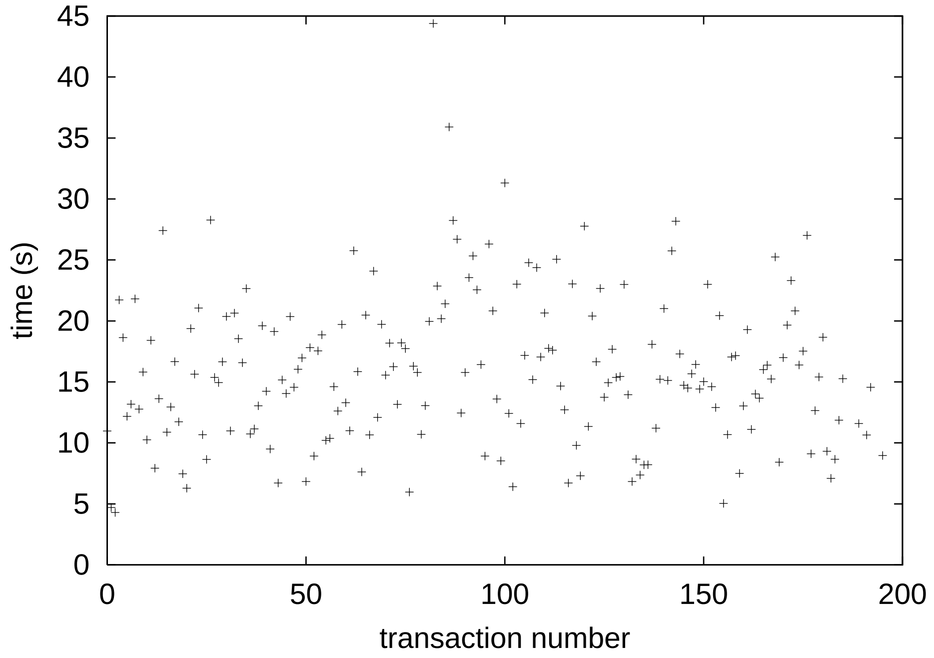


Figure 6.4: Transaction durations for 12 replicated NetBuilders.

give better performance.

Although these results clearly show that a trade-off exists between service rate and degree of replication, it is very hard to generalise to an optimal NetBuilder replication strategy. For some instances of VServ, high availability and robustness may be more important than the throughput of VPN resource reservation requests, while for others the converse may be true. It may be advantageous not to distribute NetBuilder load evenly—for example, popular physical network nodes could be grouped under a single NetBuilder to optimise for frequent operations. The scheduling of incoming NetBuilder requests in order to balance the load and minimise communication costs is an interesting area for future study.

6.5 Related Work

There is a great deal of literature in the field of distributed systems [Andrews91, Gray93, Mullender93, Lynch96], much of which applies to the replicated NetBuilder as to any distributed system. Of particular relevance for the design presented in this chapter are distributed transactions, which have been used for many years [Gray93] in applications

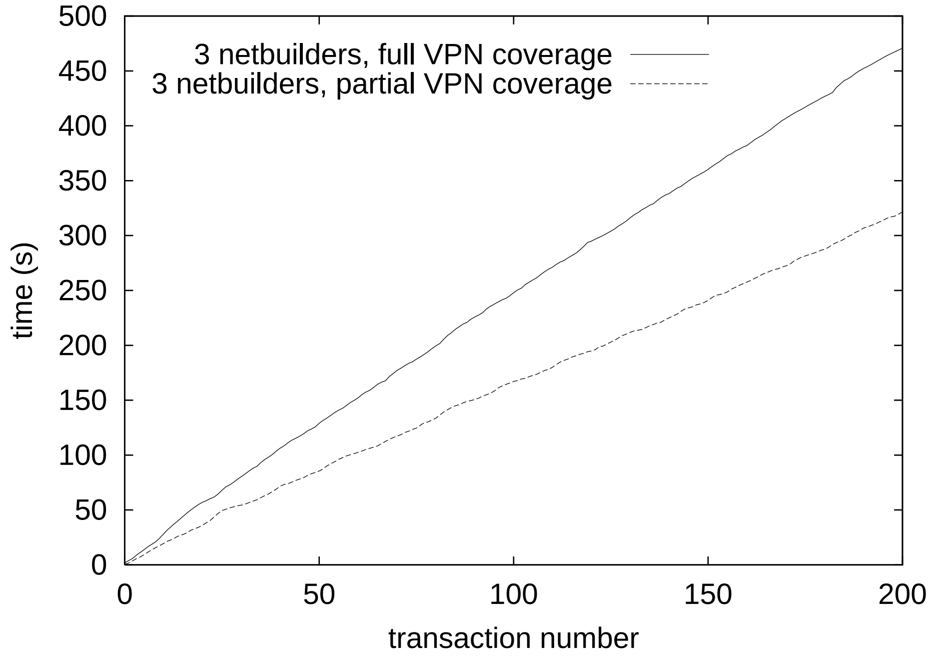


Figure 6.5: Effects on VPN creation time for different patterns of coverage of the physical topology.

such as distributed database systems. A standard for distributed transaction processing has been defined by the Open Group [X/Open96] which enables interoperability between distributed transaction management systems.

Use of distributed transactions has previously been considered in the context of connection setup by a Tempest control architecture [Rooney98a]. It is observed that the overheads of protocols such as 2PC make it unsuitable for the performance demands of connection establishment. The control architecture—the Hollowman—is therefore implemented with a lightweight transaction system that divides control operations into sets of sub-operations. When a sub-operation fails, the encapsulating operation is responsible for undoing the effects of the transaction. Similar to the conclusion drawn in this chapter with regard to the NetBuilder, the point is made that individual control architectures should be able to trade simplicity against robustness.

Transaction *routing* in the context of distributed database systems involves selecting a node at which an incoming transaction is processed according to current resource availability and the characteristics of existing transaction workloads [Nikolaou96]. A self-tuning load control method [Weikum94] that adapts tuning parameters according to current workload dynamics and predicted performance trends may be appropriate for improving replicated NetBuilder performance. In particular, automated tuning of coordinator placement could be used to improve the performance of VPN creation transactions.

6.6 Summary

This chapter presented a proposal for the design of the replicated NetBuilder. It was observed that locking the entire network during the VPN topology search would not be desirable due to its high cost, but that the consequent reduction in the degree of concurrency that can be supported is not acceptable for a dynamic VPN service. A compromise has been suggested, in which timestamped broadcast messages advertise impending changes in resource allocation state. The 2PC protocol used for VPN creation ensures transactional semantics for resource allocation within the network, and nested transactions enable overlapping label spaces to be negotiated by neighbouring nodes.

The trade-offs between complexity, performance and flexibility for the replicated NetBuilder have been explored. This chapter has shown that these trade-offs must be considered when deploying a replicated NetBuilder, and has suggested potential areas where improvements in performance may be found. A realistic request model is required to fully address the problem of VPN admission control, and to ensure the effectiveness of a replicated NetBuilder.

Replication of the NetBuilder not only improves its availability, but also its robustness. The flexible design described in this chapter allows one NetBuilder to take over responsibility for the Dividers of another if necessary. The mechanisms for doing so are beyond the scope of this dissertation, and this is another area for future work.

In the next chapter a resource revocation protocol is described that exploits the dynamic resource management capabilities of VServ to support a highly flexible VPN design creation process.

Chapter 7

Resource Revocation and Adaptive Control

This chapter focuses on the incorporation of managed resource revocation and adaptive control into VServ by exploiting its dynamic resource management capabilities.

The VSP is concerned with inter-VPN resource reservations: in general, it has no knowledge of how a VPN's resource allocation is actually used by its control architecture. This substantially reduces the management burden in the control plane, since each Divider need only track the aggregated resource *commitment* of each VPN to ensure that it remains within the limits of that VPN's resource *allocation*. A resource commitment is made by a control architecture when it establishes a connection with the desired QoS parameters. A further advantage of domain-wide resource management is that the dynamic reallocation of resources from one VPN to another is possible. An important motivation for reallocation is that it gives the VSP a great deal of flexibility, for example permitting it to differentiate between offering 'guaranteed' and 'best effort' VPN service. Section 7.1 describes other reasons for supporting dynamic resource reallocation.

The mechanism by which resource reallocation takes place is essentially the same as is used for the initial resource allocation to a VPN. Therefore resources can be shifted between VPNs with the same degree of granularity with which they are initially allocated, and it is possible to identify precisely which resources are being reallocated. Examples might be a specific label range, or the bandwidth currently in use by a particular connection. This information can be exploited by a control architecture to its advantage:

- The control architecture can respond to advance notification of an impending resource revocation.

- The control architecture can suggest to the VSP a specific set of resources for the revocation.

A *revocation protocol* is proposed in Section 7.2, which allows a control architecture to participate in the reallocation process using either or both of these approaches. Modifications within the Tempest to support resource revocation and the revocation protocol are described in Section 7.3.

The term *adaptive control* is used in this chapter to refer to compensatory action taken in the control plane prior to a resource revocation event. Adaptive control may not necessarily affect the application layer, and does not require the control architecture to monitor for loss of service. Furthermore, because it is a proactive response to an impending occurrence rather than a reaction to an earlier event, the control architecture can act more intelligently than otherwise possible. Finally, by allowing a control architecture to nominate resources for the revocation through the revocation protocol, the choice of resources can be tailored for the particular needs of the VPN suffering the revocation.

Two adaptive control architectures were designed and implemented to explore the notion of adaptive control, and these are described in Section 7.4.

Section 7.5 describes and compares related work concerning revocation, adaptive applications and adaptation techniques in the network.

7.1 Motivation

There are two possible scenarios in which dynamic resource reallocation can take place in VServ:

1. A control architecture may request the reallocation, for example to increase its label space at certain nodes, or to change the topology of its VPN.
2. Reallocation may be instigated by the VSP, in order to alter the resource allocation of an existing VPN, either with or without the participation of the affected control architecture.

Although the underlying mechanisms in the Tempest for achieving the reallocation are the same in these two scenarios, this chapter is only concerned with the latter situation, and in particular with the removal of resources from a VPN—termed *resource revocation*. A number of benefits of resource revocation can be envisaged:

Service Differentiation

A guaranteed and exact amount of resource is not always appropriate for a VPN user. Some users may prefer to take a resource allocation that is 'best effort'[1] and consequently cheaper. Indeed, when a VPN is being used to sell a network service to other parties, the best effort service can be realised simply by purchasing the appropriate best effort set of resources within the VPN. The ability of the VSP to repossess previously allocated resources at any time allows it to make this mechanism explicit, and to charge appropriately.

A further advantage for the VSP of providing service differentiation by resource revocation is that it can help to soften the strict requirements of the VPN design process. Chapter 6 discussed the trade-off between the certainty of a resource allocation found through a single 'oracle' VPN designer with global influence and complete knowledge of the current network state, and that obtained from one of several designers sharing the processing burden and increasing concurrency, but with potentially inaccurate views of current resource allocations. If resources can be repossessed from one VPN in order to assure the SLA of another, then the requirement for absolute certainty about the network-wide resource guarantees at the time of VPN creation is relaxed. Thus revocation permits a more flexible design process with potentially less overhead for lightweight VPNs, and also facilitates the operation of multiple NetBuilders.

Maximisation of Global Resource Usage

Revocation may be used to increase global resource usage by temporarily moving resources from one VPN to another. This allows good use to be made of network resources over long time periods, while also catering for short term surges in demand. For example, the predictable Mother's Day peak in telephone usage for the telephone company owner of one VPN could be handled by temporarily revoking resources allocated to a corporate VPN that is known to be lightly loaded on weekends.

Another way for the VSP to achieve good global resource usage is through an iterative allocation strategy, in which an initial resource allocation to a set of VPNs is gradually improved by dynamically reallocating resources between the VPNs. This is a further example of the flexibility in the VPN design process provided using revocation.

[1]Note that the meaning of 'best effort' in this context is somewhat richer than the typical interpretation of the term, as a best effort resource can be either *shareable*, such as bandwidth, or *non-shareable*, for example label space. Best effort is traditionally associated only with shareable resources, where performance degrades according to how many are sharing the resource and how they are using it. In this case, a best effort resource can also refer to something that may be given up completely.

SLA Violations

The resource partitioning of the Tempest relies upon in-band shaping and policing functions to ensure adherence to the traffic contracts established on connection setup. However there may be situations when the built-in capabilities of a switch or router are insufficient to enforce the partitioning on the data path. In these cases revocation of label space together with traffic monitoring could be used to punish SLA transgression, albeit in a very coarse-grained manner.

Equipment or network failure may cause the VSP itself to violate a VPN's SLA. In such a situation, it may be preferable to revoke resources from one VPN in order to fully satisfy the SLA of another, rather than have partially fulfilled SLAs for two networks. Resource revocation gives the VSP a way of responding to SLA violations according to local circumstances and policy.

Revocation is an important mechanism within VServ, particularly for the flexibility it brings to the VPN design process. The next section describes a revocation protocol which provides scope for control architecture participation in the revocation event.

7.2 The Revocation Protocol

The revocation of resources is coordinated by the NetBuilder, which instructs individual Dividers to make the required changes to local switchlets. A control architecture participating in the revocation protocol communicates with the NetBuilder for this purpose, much as it does to request modifications to its VPN.

The revocation protocol defines the messages exchanged between the NetBuilder and control architectures. This section gives an overview of this functionality, which enables control architectures to adapt and even participate in the revocation process. Resource selection techniques and the role played by the Divider is described in Section 7.3.

The protocol assumes that reliable and ordered delivery of messages takes place, and, as for all other control and management communication, that this takes place in a separate control VPN with its own resource allocation.

There are two components of the revocation protocol:

1. The interface exported by a control architecture to enable the NetBuilder to give notification of impending revocation events.

2. Extensions to the NetBuilder interface for the maintenance of a *revocation specification* for each control architecture, indicating the preferred resources for revocation.

A revocation notification message is issued in advance of the revocation event to all control architectures that possess resources selected for revocation and that have registered to receive such notification. Non-adaptive control architectures that do not implement the protocol are not affected. The message contains the set of resources that have been selected by the NetBuilder for repossession from the control architecture. The registration by a control architecture to receive this notification entails the control architecture informing the NetBuilder of the location of the interface through which it will listen for such messages.

Figure 7.1 shows the messages exchanged in the deployment of revocation notification. The acknowledgement message is not essential for the operation of the protocol, and is therefore shown with a dashed arrow in the diagram. The use of this acknowledgement message subtly changes the semantics of the protocol, by allowing the control architecture to *delay* the revocation event until the NetBuilder has received the acknowledgement. The duration of delay must be limited by a timeout period to prevent rogue control architectures from delaying the revocation indefinitely. Nevertheless, the use of the acknowledgement provides a breathing space for the control architecture in which it can prepare for the revocation as necessary to avoid loss of service to its applications.

Figure 7.2 illustrates the messages exchanged when a control architecture not only registers for notification of an impending revocation event, but also maintains a revocation specification at the NetBuilder in which it details *which* resources it would prefer to be revoked. The format of this data structure is described in Section 7.3.

The optional `update revocation specification` step shown in the diagram illustrates how the control architecture can take advantage of the revocation delay to modify its revocation specification *after* receiving notification, but before replying to that notification. Here the breathing space given to the control architecture is used to fine-tune the resources selected for revocation.

This opportunity to change the resource specification as close to the revocation event as possible may be useful for a number of reasons. The revocation notification sent to the control architecture contains the set of resources selected by the NetBuilder for revocation. Ideally this will be a subset of the resources detailed in the revocation specification maintained by the control architecture. However there may be situations when it is a superset, or even disjoint, and the specification may also be stale due to more recent changes in circumstance of the control architecture. The opportunity for an update immediately prior to the revocation event itself may be the best way of providing a relevant resource

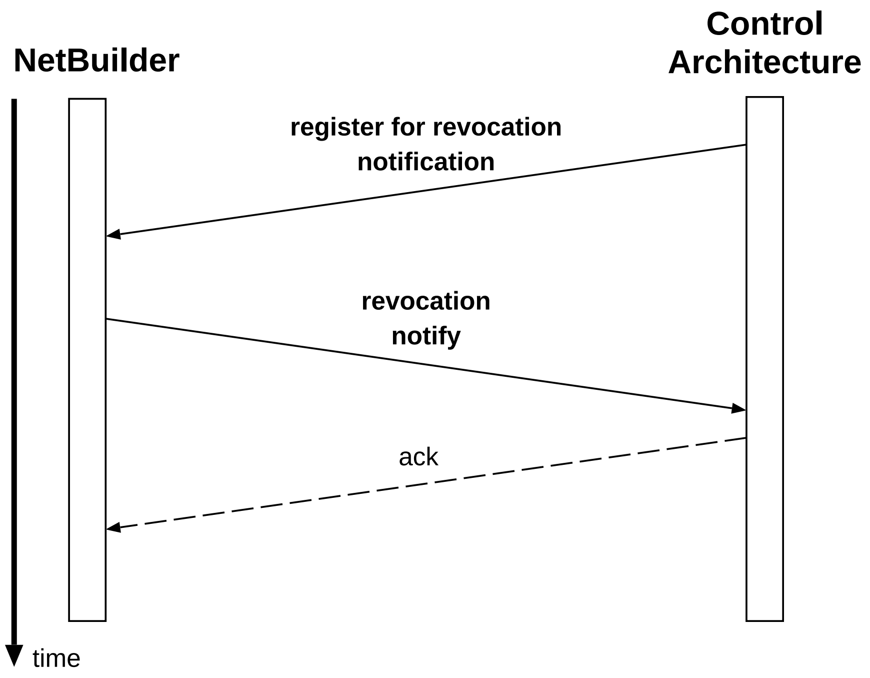


Figure 7.1: Exchange of messages for revocation notification. The dashed arrow indicates an optional step.

specification. Furthermore, the determination of such information may be an onerous task that the control architecture prefers to perform only when necessary.

A change to the revocation specification when a revocation is pending will always prompt the NetBuilder to issue a new revocation notification. Once again a timeout period should limit the size of the delay given to a control architecture, and in this case care must be taken to guard against denial of service attacks by malicious control architectures constantly updating their specification data in order to prevent the revocation from taking place.

Note that there is no requirement for a control architecture participating in the revocation specification part of the protocol to also register for revocation notification from the NetBuilder. In particular, it may be useful to employ an agent to maintain a resource specification on behalf of a legacy, non-adaptive control architecture. The agent keeps a resource specification up-to-date, perhaps using customised monitoring to observe the control architecture's behavioural patterns, but will not have any use for advance notification messages.

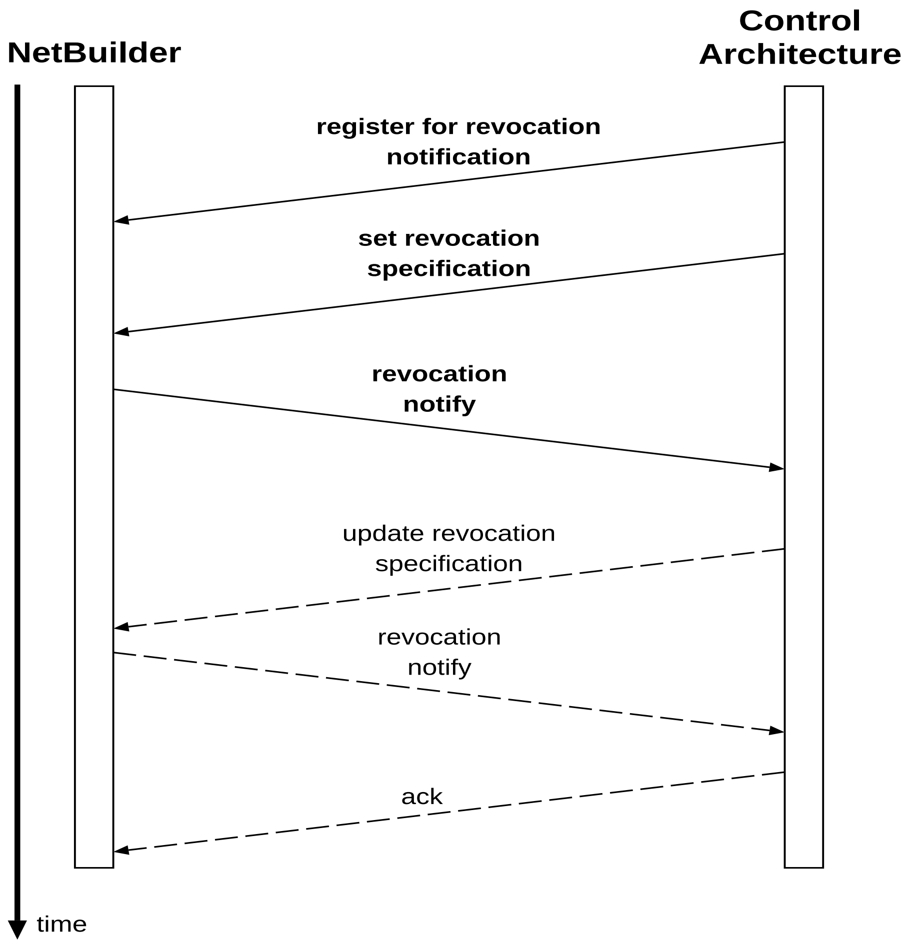


Figure 7.2: Exchange of messages for revocation notification and revocation resource specification. Dashed arrows indicate optional steps.

7.3 Implementation

In Figure 7.3 a possible sequence of steps in a revocation event are shown in order to illustrate the roles played by each of the components.

As indicated in the diagram, the NetBuilder acts as the coordinator of the revocation, and has three main tasks:

1. Decide when a revocation is required, and how much of each resource must be reclaimed.
2. Select one or more 'victim' VPNs.

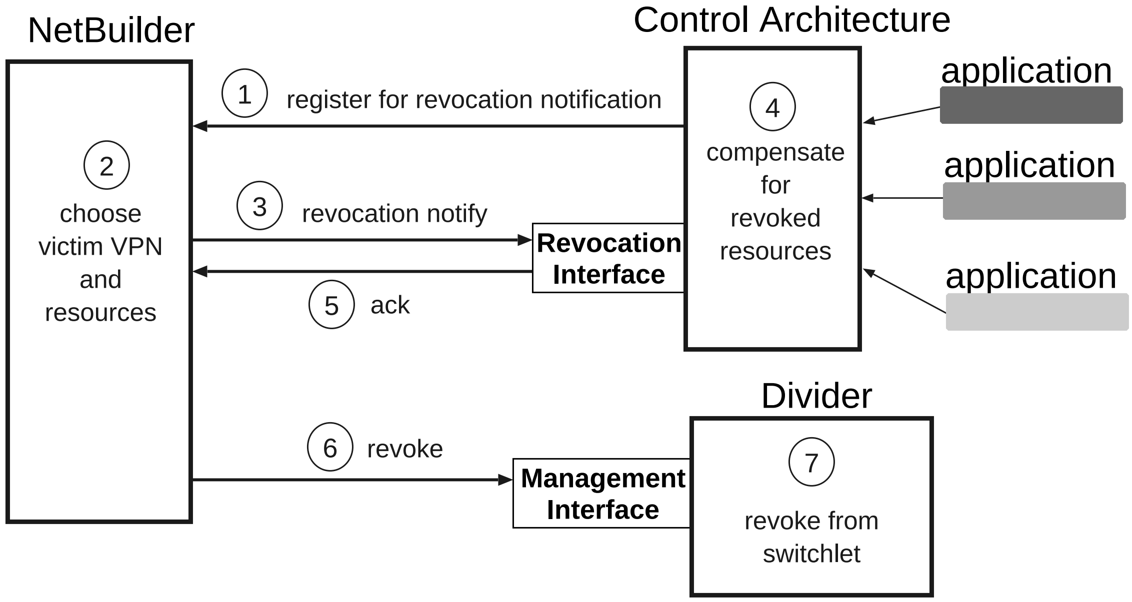


Figure 7.3: Sequence of steps in a revocation, making use of revocation notification.

3. Determine how much of each resource to revoke from switchlets belonging to each victim VPN.

The methods employed in order to carry out each of these steps are very much dependent upon the VSP's local policies and mechanisms. For example, the decision to perform a resource reallocation may be in response to the violation of an SLA, and the selection of victim VPNs may be influenced by how much individual VPN customers have been charged.

Figure 7.3 shows revocation notification being employed: the first step is the registration by the control architecture to receive notification if the decision to revoke its resources is taken. The registration involves informing the NetBuilder of the location of the control architecture's *revocation interface*, through which it will listen for notification messages. The interface definition is shown in Figure 7.4 with two versions of the notification method. When an acknowledgement in the form of a reply from the control architecture is not permitted, the notification includes a parameter indicating the approximate number of seconds until the revocation event. Note that this advance warning is on a relatively slow timescale. The task of revocation is part of global inter-VPN resource management, and is therefore handled on the same slow timescales as VPN creation or modification.

As shown in Figure 7.3, when a NetBuilder decides to reallocate resources, it chooses

```
// ack version, control architecture can delay revocation
boolean notifyRevocation(in SwitchSpec spec);

// no ack, num_secs gives a hint as to when revocation will occur
void notifyRevocation(in SwitchSpec spec,
                      in long num_secs);
```

Figure 7.4: Alternative revocation notification methods in IDL.

a victim VPN and determines how much of each resource to revoke from each switchlet belonging to that VPN (step 2). If the control architecture has opted to receive revocation notification, then the NetBuilder will notify the control architecture with a `SwitchSpec` constructed from the selected resources (step 3). At this point the control architecture can take action to mitigate any ill effects of the impending revocation, before replying with an `ack` (steps 4 and 5). The NetBuilder proceeds with the revocation by instructing the Dividers to deduct the previously selected resources from the VPN's switchlets (steps 6 and 7).

The implementation of the resource specification part of the revocation protocol involves the maintenance of a data structure at the NetBuilder that indicates which resources the owning control architecture would prefer to be revoked. Figure 7.5 shows how this specification is defined, and also the method added to the NetBuilder interface for creating or modifying this structure.

The `VciRevSpec` structure shown in Figure 7.5 specifies label space that is preferred for revocation. The `BwRevSpec` structure can identify the bandwidth associated with a particular connection (to indicate unused bandwidth the connection information can be set to "any"). This allows a control architecture to express policies such as "take no more than x bandwidth from connection y". Ideally, when part of the committed bandwidth is removed from a connection the in-band policing on that connection is dynamically 'dialled down'. In practice some Tempest implementations, such as on ATM, may require the connection resources to be remapped to a new connection, whereas others, for example using MPLS on Linux, can allow the policing parameters to be modified on-the-fly.

In this implementation, the notion of priority within the revocation specification is very simple. When the NetBuilder is selecting resources to revoke, it examines each element in the appropriate list in turn until the required amount of resource is found. It is conceivable

```
// Vci Revocation Spec
// Values of -1 mean "any"
struct VciRevSpec {
     SwitchId switch_id;
     PortId port_id;
     short vpi;
     short vci;
};

// Bandwidth Revocation Spec
// Values of -1 mean "any"
struct BwRevSpec {
     SwitchId switch_id;
     PortId port_id;
     short vpi;
     short vci;
     long bandwidth;
};

typedef sequence<VciRevSpec> VciRevSpec_seq;
typedef sequence<BwRevSpec> BwRevSpec_seq;

// Revocation Spec
// Set up by control arch
struct RevocationSpec {
     VciRevSpec_seq vci_rev_specs;
     BwRevSpec_seq bw_rev_specs;
};

// Set up resource revocation spec
void setRevocationSpec(in RevocationSpec revocation_spec)
     raises ( NetBuilderInvalidRevocationSpec,
              NetBuilderNotImplemented );
```

Figure 7.5: NetBuilder revocation specification methods.

that some control architectures might benefit from the ability to express more sophisticated schemes. This is an area for further work.

7.4 Adaptive Control

An adaptive control architecture is able to respond gracefully to impending changes in its resource allocation. An advantage of using the revocation protocol is that a control architecture is able to proactively prepare for an imminent revocation, either by opting to receive advance notification, or by deciding itself which of its resources it would prefer to be revoked.

In this section the nature of adaptive control is explored and two realistic examples that make use of the revocation protocol are described. There is a great deal of related work on both network-level adaptation techniques and on adaptive applications, and this is reviewed within Section 7.5.

The nature of an adaptive response will largely depend upon what kind of control architecture is involved: *general purpose* or *service-specific*.

- General purpose control architectures provide a generic interface over which applications can transfer data across the network. Implementations of standard protocols on the Tempest, for example TCP or ATM signalling, result in general purpose control architectures.
- Service-specific control architectures are designed to support a particular network service or application using the fine-grained resource control of the Tempest to optimise network resource usage to the specific needs of the application. An example is Videoman [van der Merwe98a], which supports a video conferencing service that minimises the number of connections by means of a gather and distribution tree topology.

General purpose control architectures have two options for reacting to service degradation. They can try to compensate in the control plane, for example by re-routing calls or rejecting some call requests, or they can do nothing and rely on adaptation by the application. Service-specific control systems may in addition be able to explicitly invoke application-level adaptation techniques such as compression or a decrease in the transmitted traffic.

The type of resource being revoked will also have an impact on the reaction of the control architecture. Reduction of label space can not, in general, be handled by applications, and will require the control architecture to take compensatory action in the control plane. An example of such a response is described in Section 7.4.1, where an existing connection is re-routed.

In contrast, the revocation of bandwidth (or any other shareable resource) will affect the QoS offered by the control architecture, and existing traffic may experience a deterioration in performance. A reduction in available traffic quality without a corresponding reduction in label space can be compensated for by the control architecture by reducing the label space available to its users. On the other hand it may be acceptable in that VPN for quality to simply degrade (in particular for a control architecture like IP that only ever assumes best effort service). A service-specific control architecture has the additional flexibility of being able to directly manipulate the service to respond to such degradations. The control architecture described in Section 7.4.2 makes use of application level knowledge and the revocation protocol to ensure that the ill effects of such service degradation are minimised.

The following adaptive control architectures demonstrate some of the mechanisms available for deploying adaptive control in VServ.

7.4.1 Label Space Revocation from CRACA

CRACA (CoRny Adaptive Control Architecture) is a service-specific control architecture for a security guard type application. CRACA was designed and implemented for the purposes of demonstrating adaptive control. Figure 7.6 shows a setup in which CRACA is deployed on a network topology of three switches.

The VPN for CRACA in this setup consists of three video sources and a single video sink. CRACA sets up and tears down connections on a single switch in order to cycle between the sources, thus giving a 'security guard' a rotating view of each source. This style of connection manipulation demonstrates the advantages of the fine grained access to physical resources available with the Tempest—the capability to dynamically remap just one endpoint of a connection is not supported in the standard signalling APIs that are currently available.

In order to demonstrate adaptive control in CRACA, label space is revoked at the switch where the 'floor control' is taking place, such that there is now insufficient label space in that switchlet for all three sources. Since CRACA is adaptive, it is able to re-route one of the source streams and thus maintain service to the sink. If CRACA was not adaptive, the video stream that previously used the revoked label would be lost.

CRACA also makes use of the revocation protocol to prevent any break in service being discernible at the sink. It registers for revocation notification, and then takes advantage of the advance warning it receives to re-route the connection before the label is revoked.

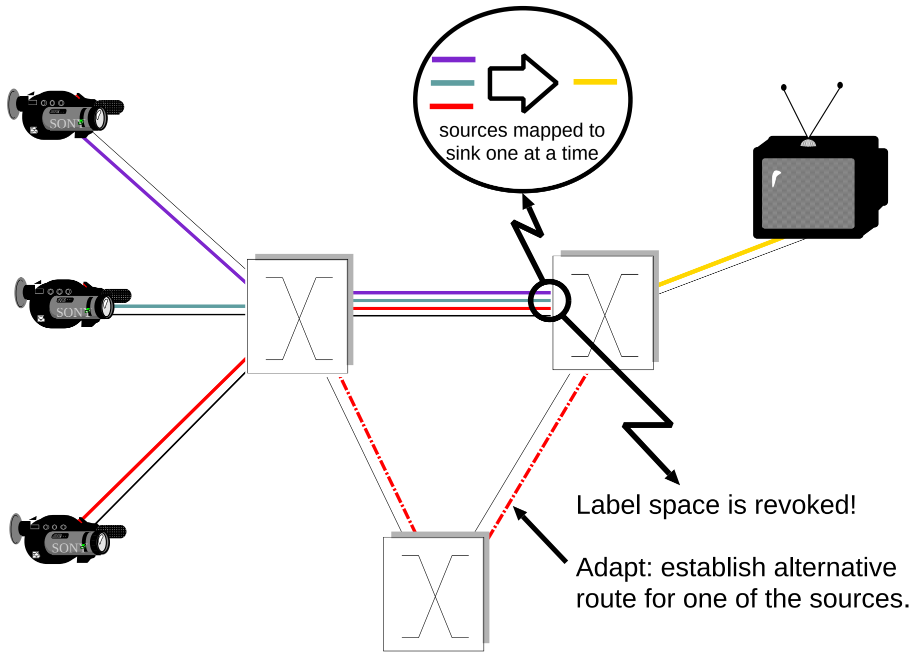


Figure 7.6: Label space revocation in CRACA.

7.4.2 Bandwidth Revocation from VACANT

Another control architecture that was devised in order to demonstrate adaptive control is VACANT (Video Adaptive Control with A kNown Topology). The setup for a demonstration of how VACANT makes use of a revocation specification to adapt to a reduction in its allocated bandwidth is shown in Figure 7.7.

In Figure 7.7, four connections are shown. The connections shown with solid lines belong to the VPN of VACANT. All of the VPN's resource is used for the three traffic streams: two video streams from video recorders to televisions, with the third being constant rate data traffic from a traffic generator, sent to an unspecified destination. The connection drawn with a dotted line is a single unspecified traffic stream that traverses the same link between the two switches as all three of the VACANT streams. It has no allocated bandwidth at all, but simply sends data at the line rate and soaks up bandwidth unused by any other connection. This type of best effort flow does not belong to a VPN and is therefore not usually allowed in VServ. In general, the Tempest does not allow a VPN with no bandwidth to set up connections or send traffic. However the ATM implementation used here is amenable to supporting this behaviour, and an exception was made in order to graphically display the effect of bandwidth revocation.

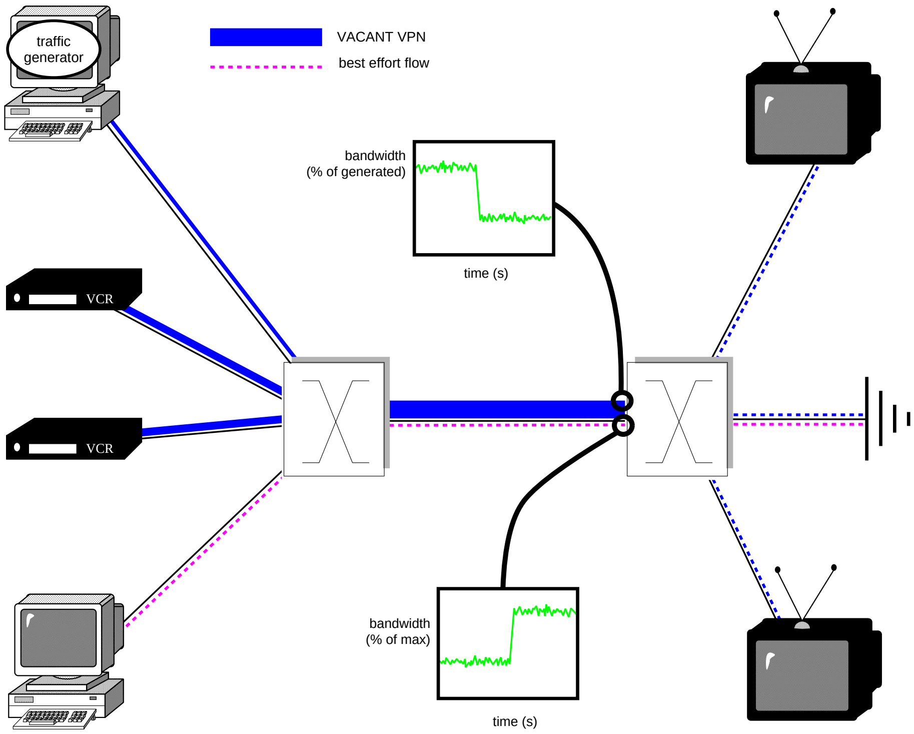


Figure 7.7: Bandwidth revocation in VACANT.

Within VACANT, the data streams have been allocated a share of the available bandwidth, roughly indicated by the thickness of the lines from the sources. The video streams each have just over twice the bandwidth given to the traffic generator, with one video stream having a slightly higher peak rate than the other.

Two traffic monitors are installed at the ingress to the second switch. One is observing the amount of traffic from VACANT's traffic generator and plotting this as a percentage of the amount that was actually generated against time. The other is monitoring the best effort flow's share of traffic and plotting this as a percentage of the line rate against time. When bandwidth is revoked from VACANT this causes the best effort flow's share of the bandwidth to increase, and is reflected in the graph generated by the traffic monitor on that connection.

In this demonstration, VACANT makes use of a revocation specification structure in the NetBuilder to tailor the revocation as appropriate for the applications running over it. Without a revocation specification, bandwidth is revoked from VACANT according to some policy of the VSP over which VACANT has no control. In the current revocation implementation, bandwidth is proportionally revoked from each connection in the VPN[2]. The bandwidth that VACANT has associated with each of the two video streams was determined by the motion content and hence burstiness in the respective videos. Thus the smaller of the streams is disproportionately affected, resulting in severe deterioration of the picture.

The situation is significantly improved when VACANT participates in the revocation protocol and sets up a revocation specification indicating that it would prefer bandwidth from the traffic generator's connection to be revoked in preference to either of the video streams. Now when the revocation occurs, both video streams retain acceptable picture quality with correspondingly less bandwidth consumed by the traffic generator.

In Figure 7.8, the effects of revocation on the bandwidth of the streams in VACANT, as observed by the traffic monitor at the ingress to the second switch, is shown. The graph shows the two video streams and the constant rate data stream produced by the traffic generator. Video1 is produced by footage from a talk show interview, whereas Video2 contains the recording of a motor race, hence is substantially more bursty. The experiment was run over the ATM implementation of the Tempest, using ATM codecs for the video streams.

Two bandwidth revocation events take place. In the first, which occurs after 30 seconds, the revocation protocol is not used and bandwidth is revoked by the VSP from each connection proportional to its allocated bandwidth. This results in an unwanted loss of quality in the video streams. VACANT then uses the revocation protocol to encourage the removal of bandwidth from that the traffic generator's connection in preference to the video streams. Resources are restored at 60 seconds, during which time VACANT populates its revocation specification at the NetBuilder. The second revocation event at 90 seconds shows the result—the same amount of bandwidth is revoked from the VPN in total, but the two video streams are significantly less degraded.

7.4.3 IP with Load Sensitive Routing

Both CRACA and VACANT are examples of service-specific control architectures that make use of application level knowledge to tailor their adaptive behaviour. This section presents an example of adaptive control in a general purpose control architecture.

One form of adaptive control is the division of resources into guaranteed and best

[2] The implementation also supports equal revocation of bandwidth from each connection. The choice of which default resource selection policy to use is a matter for the VSP.

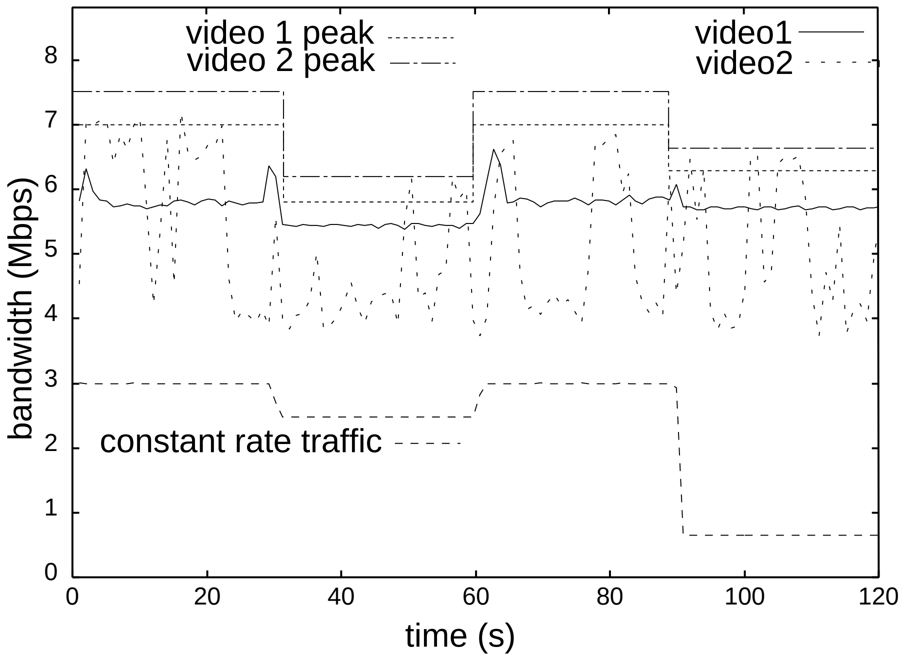


Figure 7.8: Experiment demonstrating the use of the revocation protocol in VACANT.

effort. For example, a content distribution architecture could use its guaranteed resources to get its data into the network, and best effort for delivering it to participants, where it is assumed that individual receivers can tolerate glitches and degradation in quality of the stream.

Another situation where this form of adaptation may be appropriate is in a control architecture based on load-sensitive routing of IP [Shaikh99]. In this version of IP short-lived flows are transmitted on static pre-provisioned connections that are created with best effort QoS, while long-lived flows are dynamically routed with allocated quality of service resources. The routes for these flows are determined according to current load, i.e. according to the current bandwidth allocation of existing long-lived flows. The aggregate bandwidth used by long-lived flows is kept below the total available to ensure availability for short-lived flows. Any slack caused by over provisioning for long-lived flows is taken up by the short-lived flows.

As an adaptive control system, this version of IP would purchase sufficient guaranteed resources from the VSP for the long-lived flow bandwidth allocation, with the remainder being best-effort, i.e. preferred for revocation. On receiving notification of an impending reduction in bandwidth, the control architecture changes the maximum capacity of long-lived flows accordingly to compensate for the loss of bandwidth available to the short-lived flows. If the resource revoked is label space, then the pre-established connections for short-lived flows are re-routed, and the routing algorithm for long-lived flows modified as required.

7.5 Related Work

Related work for this chapter falls into two categories. In Section 7.5.1 other contexts in which revocation occurs are reviewed. Section 7.5.2 considers some examples of adaptive network behaviour.

7.5.1 Revocation

To the author's knowledge, no other work addresses the deliberate use of revocation as a technique for network resource management. This section reviews some instances of resource revocation in other systems.

7.5.1.1 The Exokernel

The Exokernel [Engler95] is a minimal operating system kernel that multiplexes hardware resources to application-level library operating systems, which then provide the standard higher level abstractions to applications.

Since physical resources are directly exported to the library operating systems, a revocation protocol is defined for when physical resources must be reclaimed from an application. Explicit notification of revocation is used to allow the library operating systems to remap bindings to the revoked resources as necessary (for example, to update page table entries). The revocation protocol gives library operating systems the opportunity to respond to revocation requests, and defines a procedure to be followed when a library operating system does not cooperate. This second stage, known as the *abort protocol*, entails forceable repossession of the resources, which are recorded in a *repossession vector*, and then an exception sent to the library. The vector may also contain details of other resources to which any state associated with the revoked resources has been transferred, and in addition, a library may load the vector in advance with resources that can be used for this purpose.

Although the Exokernel's revocation protocol provided the inspiration for the revocation protocol presented in this chapter, there are some key differences. The repossession vector of the Exokernel holds details of resources that have already been revoked, whereas VServ's revocation specification indicates preferences of the control architecture for a future event, and is not updated by any entity other than that control architecture (or an agent acting on its behalf). In the Exokernel, each library operating system is expected to free resources itself according to instructions from the kernel. Because VServ supports legacy, non-participating control architectures, it is not feasible to expect every control architecture to free up resources itself. Furthermore, the repossession of resources in VServ may entail taking allocated but unused resources—an operation that is carried out simply by reducing switchlet size in the Divider, without needing the control architecture to take any action at all.

7.5.1.2 Other Operating Systems

McNamee proposes an extension to the Mach operating system to support user-level page replacement policies [McNamee90]. An interface between the kernel and an external pager is defined through which the pager specifies which pages are to be relinquished. This permits optimisation of the paging scheme according to an application's particular patterns of access. A similar approach exists for external page management [Harty92].

The philosophy behind these approaches is very similar to that of the Tempest. Application-specific page replacement policies can be compared with support for multiple, service-specific network control architectures. Allowing a control architecture to define its own revocation policies according to the needs of its higher-level applications not only parallels the use of an application-specific page replacement policy, but is also another instantiation of the same overall idea.

Memory management in the Nemesis operating system is a further example [Hand99]. A frame allocator deals with the allocation and revocation of physical memory to and from applications. The approach is similar to the division of resources into guaranteed and best effort that is provided in VServ using the revocation protocol. In Nemesis applications can be guaranteed a certain number of physical frames, and may also have a number of so-called *optimistic* frames which may be revoked by the frame allocator when necessary. Similar to the revocation protocol and other operating system schemes for user-level page replacement, each application maintains a frame stack which contains an ordered list of physical frame numbers, indicating its preference for revocation.

7.5.1.3 Policy-based Preemption

The COPS (Common Open Policy Service) protocol [Durham00] is a framework for the exchange of policy information between a client (the enforcer of policy) and a server (responsible for taking policy decisions). The protocol defines the format of messages used for client requests and state updates, and for server policy decisions. It has been proposed that COPS be used to provide policy based admission control [Yavatkar00]. The proposal requires that there be support for preemption of previously installed state, in particular previously reserved resources, in order to allow a new call admission request. A related proposal [Herzog00] describes preemption priority policy elements for use in QoS-based signalling (for example with COPS). It details strategies for determining the priority of merged flows, and suggests that once admitted, to ensure stability flows are assigned a different priority that is at least as great as their preemption priority.

The ability to alter preemption priorities of objects according to policy, within the context of a general policy framework such as COPS, is a mechanism that could prove useful for resource revocation in VServ. Current proposals limit their scope to per-flow QoS signalling, and hence to admission control decisions. There has not as yet been consideration of dynamic resource reallocation between flows, nor of preemption of specific resources belonging to an individual flow.

7.5.1.4 Capabilities

Mechanisms for revocation are most often encountered in the context of capability-based operating systems. [Karger88] discusses the difficulties of immediate revocation in such a system. A capability can be thought of as "a ticket granting access to an object", and it is both necessary and sufficient to be in possession of the appropriate capability in order to access an object. The ability to transfer and copy capabilities from one holder to another makes the search to find all instances of a particular capability time consuming, or even impractical in the case of a distributed system. A large number of schemes have been proposed to address this problem, for example [Gligor79, Wilkes79, Wulf81, Karger88, Shapiro99]. However proposals for revocation of capabilities tend to be cumbersome and it is notable that no capability-based systems are in common use today.

The style of revocation used in VServ has different properties to that in a capability-based system, hence avoiding many of the difficulties. The means of using the resources of a switchlet is through the Ariel interface reference for that switchlet: the interface reference is merely a handle and does not contain any information or implicit access permissions for a specific set of resources. Thus there are no overheads associated with performing the revocation, as it is simply a matter of changing the parameters of the switchlet in the Divider. When bandwidth or any other shareable resource is revoked from a switchlet, the in-band policing function of the switch or router is employed to prevent further usage of the revoked resources. On the other hand, revocation of label space, where a label may be considered conceptually closer to an access permission, is enforced by policing on the control path—a label can not be used by a control architecture to transmit data unless a connection has previously been established through the Ariel interface.

7.5.2 Adaptive Network Behaviour

Adaptive behaviour in the network can be contrasted with the mechanisms available to adaptive control architectures in VServ. This section describes adaptive techniques used by QoS architectures and network protocols.

7.5.2.1 QoS Architectures

QoS architectures are middleware systems that deliver resource guarantees to applications. They are often specifically aimed at multimedia applications, and usually deal with the provision of both network and operating system resources.

An example of a QoS architecture is GARA (Global Architecture for Reservation and Allocation) [Foster00]. GARA supports adaptive applications using QoS resource managers which allow applications to modify resource reservations dynamically. The resource managers use sensors to monitor network state (for example, packet loss), and notify applications of changes. The dynamic feedback received by an application allows it to either adapt, request a resource reservation change, or use a combination of both techniques. Other architectures based on dynamic feedback include the middleware control architecture [Bolliger98] and a framework for a closed-loop control system [Li99].

AQUA [Lakshman96a] is a QoS architecture which provides a generic QoS specification for all resources, similar to the network QoS abstractions used in Ariel. The same authors also examine the implementation of QoS renegotiation by extending the standard ATM signalling protocol [Lakshman96b]. The mechanisms incorporated into the switch to achieve this could be used in the Tempest to provide a method of elegant renegotiation of QoS for existing connections, both as directed by the control architecture, and for use in revocation.

In general, QoS architectures support adaptive applications by providing a feedback mechanism from the network and possibly some sort of resource reservation functionality. The strong resource guarantees available with the Tempest make it suitable for supporting QoS architectures. A control architecture that implements a QoS architecture can use revocation notification to remove the need for a monitoring function, hence making the feedback to applications more precise. This is also a form of adaptive control, in that the control architecture uses information about resource revocation to invoke application-level adaptation mechanisms.

The dynamic connection management (DCM) scheme [Parris94] uses connection re-routing to maintain QoS reservations. The lack of scalability of the DCM approach is addressed by trying to limit the extent of dynamic reconfiguration of a connection to a small number of network nodes [Bettati96]. These two connection management schemes are examples of techniques that may be employed by adaptive control architectures to prepare for a revocation event without disruption to applications.

7.5.2.2 Network Protocols

Many existing network protocols provide a limited form of adaptation in the control plane. For example, dynamic routing protocols such as Open Shortest Path First (OSPF) [Moy91] modify the routing table according to constantly updated information about the current state of the network. Another example is TCP's congestion control algorithm [Jacobson88] in which the sender modifies its transmission rate in response to packet loss in the network.

The existence of adaptive applications that do not have stringent real time requirements has resulted in a the standardization of a service category for ATM networks known as *Available Bit Rate* (ABR). ABR is intended to be used by applications which can modulate their data rate as a function of the network load, and hence avoid cell loss, as well as unnecessary over-commitment of resources. Such applications tend to transmit bursty traffic, but are not particularly sensitive to delay or delay variation. ABR provides rate feedback from the network to the sender using a rate indicator in resource management (RM) cells.

As is the case with QoS architectures, network protocols use observation of network conditions to trigger an adaptive response. A control architecture version of each network protocol could use similar techniques to provide adaptation. However the nature of the adaptation in these cases is limited; in VServ control architectures can make use of a much richer set of adaptation techniques, and, due to the relatively slow timescales, need not be as concerned with adverse effects such as oscillation which may result from dynamic feedback.

7.6 Summary

This chapter proposed resource revocation as a useful mechanism for VPN service differentiation, for improving physical resource usage, and as a tool for reacting to SLA violations by individual VPNs. A revocation protocol was presented in order to allow control architectures to participate in the revocation process, and thus be able to adapt more intelligently when resources are repossessed. An implementation of resource revocation and the revocation protocol in VServ was described. Adaptive control is defined to be compensatory action taken by a control architecture in anticipation of resource revocation, and three examples of adaptive control architectures were described.

The simple revocation mechanism described in this chapter is a key feature of VServ. Its dynamic resource management capabilities, which allow both inter-VPN *and* intra-VPN management, provide a powerful and flexible framework with which to support a dynamic VPN service.

Chapter 8

Conclusion

This dissertation described VServ, a service offering dynamic and resource-assured VPNs that can be acquired and modified on demand, and which supports customised network control for each individual VPN. This chapter contains a summary together with suggestions for future work and concluding remarks.

8.1 Summary

Chapter 1 presented the motivation for VServ. Three main considerations were identified:

- There is a need for VPNs that are both resource-assured *and* dynamic.
- The Tempest network control framework supports VPNs with customised control architectures and guaranteed resource allocations. A VPN service over the Tempest exploits this functionality to enable a *multiservice* network.
- Other efforts to offer QoS guarantees in the network face similar challenges with respect to optimising domain-wide aggregate resource allocations, thus the inter-aggregate resource management aspects of this work have a potentially broader application.

The context for the research was described, including the goals and scope of VServ, together with a brief introduction to the Tempest.

Motivation for investigating a dynamic and resource-assured VPN Service was explored further in Chapter 2. The various types of VPN that are currently used were described, highlighting the differences to those obtained with VServ. The provision of QoS in networks was also covered. ATM and MPLS were described, both of which are connection-oriented network technologies that have been used for implementations of the Tempest. Approaches to incorporating QoS in IP networks, and hence the support of the Internet as a multiservice network, were reviewed. The final section of Chapter 2 discussed open signalling and devolved control, which allows VServ to accommodate customised control of VPNs.

In Chapter 3 the motivation for, and architecture of the Tempest network control framework was described in more detail. The important distinction between inter- and intra-VPN resource management mechanisms in the Tempest was explained. This distinction makes the dynamic resource reallocation capabilities of VServ viable, whilst handling highly dynamic VPNs or a large number of VPNs. Extensions to the original implementation of the Tempest to support dynamically reconfigurable QoS were detailed. A bootstrap VPN was also discussed, as well as aspects of the Tempest implementation on MPLS, specifically the need to install packet filters at the ingress routers in order to allocate incoming packets to a VPN. This is an important consideration for enabling interoperability with other network domains, especially with the Internet.

Issues specific to VServ, rather than its support in the Tempest, were the subject of the next four chapters. Chapter 4 was concerned with automating the VPN design process. It described an outline of the process, broken down into three main stages. The first is the description of the desired VPN, with a `configure` utility proposed to provide a template for describing the requirements of particular control architectures. The `configure` wizard produces a VPN description in a specification notation called VANDAL, which is defined by a simple context-free grammar. The VANDAL specification is parsed to generate code for functions that prune links from the physical topology graph and check the properties of a (possibly partial) VPN topology with respect to the properties detailed in the VANDAL description. Some discussion of determining current resource availability and of assigning edge weights followed. A case study illustrated these steps of the VPN creation procedure. The chapter concluded by reviewing related efforts to dynamically provision VPNs.

Chapter 5 continued with the next stage of the VPN design process—the search for a reasonable topology and resource allocation for a VPN that satisfies the requirements of both customer and VSP. The general problem is intractable, except in very small networks, but it was observed that certain types of VPN topology are suitable for the application of polynomial time algorithms. Lightweight or highly dynamic VPNs can be initialised with a suboptimal topology, and, if needed, subsequently refined using dynamic resource reallocation. For a tree with no subgraph properties, a Steiner minimal tree (SMT) heuristic algorithm is appropriate. A QoS-constrained Steiner algorithm is used for a source-based tree with subgraph properties. All other types of topology warrant a brute-force approach.

It is proposed that when no suitable polynomial time algorithm exists, heuristics can improve performance sufficiently for automated VPN design to be practical in networks of sufficiently small size or low density. This claim is supported by comparing performance and quality of results obtained with a naïve search against that using a simple heuristic. A realistic example highlighted the advantages obtained with customisation of the search procedure, and a discussion of inter-domain resource reservation concluded the chapter.

In Chapter 6 it was argued that the NetBuilder should be replicated to improve its availability. When a VPN creation request is received by the NetBuilder, it first performs VPN design and then the corresponding reservation of resources in the network. Because the process of VPN design is quite heavyweight, it is not practical to disallow other simultaneous VPN design or resource allocation activities. However this can lead to inconsistent views of global state among the NetBuilder instances, resulting in suboptimal or invalid VPN topologies. This problem is partially mitigated with the broadcast of a `searching` message at the start of the VPN design process. Receipt of this message by a NetBuilder that is simultaneously searching may cause it to optimistically continue, or else to prematurely terminate the search on the assumption that there may be an impending conflict. The chapter also ensured transactional semantics for VPN creation using the standard two-phase commit protocol. In addition, the overheads of replication were discussed, supported by experimental observation.

In Chapter 7 the focus turned to making use of the dynamic resource management capabilities of VServ to provide support for resource revocation. Revocation allows the VSP to offer differentiated service, to maximise global resource usage and to respond to SLA violations. The chapter presented a protocol which can be used by a control architecture to prepare for an impending revocation, and to suggest to the VSP a preferred set of resources. This facility allows a proactive, fine-tuned response by an adaptive control architecture, rather than having it react some time later after observing a loss of network service. The messages exchanged in the revocation protocol were described, together with some examples of adaptive control. The chapter also reviewed a selection of related work, which included revocation in other contexts, and adaptive architectures and protocols.

8.2 Future Work

The design of VServ presented in this dissertation is exploratory, with emphasis on the investigation of potential problems and identification of ways in which the service can be made practical. This leaves a number of potential areas for future work.

Improvements to the procedure for VPN topology search could be explored, in particular the finding of a polynomial time heuristic algorithm that works reasonably well in practice. The initialisation of highly dynamic VPNs is another area that would benefit from further investigation; mechanisms for providing a topology and resource allocation that is acceptable to both VSP and customer in a volatile situation merit further examination.

The deployment of VServ on a network offering VPN services to 'real' users is the next step for ratifying the claim of this dissertation that the service is viable. In such a situation, considerations such as the trade-offs for the replicated NetBuilder of availability versus simplicity could be investigated further. Given a realistic set of VPN creation requests, the automated allocation of incoming VPN requests to NetBuilder nodes would be an interesting area to pursue. Such a set would also be useful for exploration of effective distributed VPN admission control.

Deploying VServ in a realistic environment would make the need for inter-domain SLA negotiation pressing: interoperability with other types of networks, as well as other instances of VServ is an area for future study. At the implementation level, it would be interesting to explore in greater depth the packet filtering mechanisms required by an MPLS implementation of the Tempest.

8.3 Conclusions

A key property of VServ is that resource allocation is explicitly separated from resource usage. The VSP divides 'chunks' of resource among VPNs, whilst individual control architectures deal with the association of resources with flows. This separation eliminates the need to keep track of per-flow state globally, giving a mechanism for scalable resource allocation and reallocation. It also allows each resource-assured VPN its own customised control architecture.

VServ supports the dynamic provisioning of resource-assured VPNs with a practical design process and availability through replication. A resource revocation protocol permits a high degree of flexibility in the creation of VPNs which further facilitates dynamic provisioning.

It is the thesis of this dissertation that VServ enables a multiservice network in a flexible and practical manner. Deployment of VServ in the real world, for example on an ISP's network, would seem plausible in the near future. It is hoped that the validity of the approach presented in this dissertation will be borne out by real world experience.

Bibliography

[Alles95] Anthony Alles. *ATM Internetworking.* Cisco Systems White Paper, May 1995. (pp 16, 89)

[Andersson00] Loa Andersson, Paul Doolan, Nancy Feldman, Andre Fredette, and Bob Thomas. *LDP Specification.* Internet Draft, August 2000. Work in progress. (p 17)

[Andrews91] Gregory R. Andrews. *Concurrent Programming.* Benjamin/Cummings, 1991. (p 103)

[ATMF96a] The ATM Forum. *ATM User-Network Interface (UNI) Signalling Specification Version 4.0*, July 1996. (pp 3, 16)

[ATMF96b] The ATM Forum. *Private Network-Network Interface Specification Version 1.0 (PNNI 1.0)*, March 1996. (pp 16, 89)

[Babaoğlu93] Özalp Babaoğlu and Keith Marzullo. *Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms.* In Mullender [Mullender93], chapter 4, pages 55–96. (pp 97, 99)

[Baker00] F. Baker, C. Iturralde, F. Le Faucheur, and B. Davie. *Aggregation of RSVP for IPv4 and IPv6 Reservations.* Internet Draft, March 2000. Work in progress. (pp 4, 19)

[Ballardie93] Tony Ballardie, Paul Francis, and Jon Crowcroft. *Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing.* Computer Communication Review, 23(4):85–95, October 1993. Proceedings of SIGCOMM September 1993. (p 72)

[Barr93] William J. Barr, Trevor Boyd, and Yuji Inoue. *The TINA Initiative.* IEEE Communications Magazine, 31(3):70–76, March 1993. (p 43)

[Bauer97] Fred Bauer and Anujan Varma. *ARIES: A Rearrangeable Inexpensive Edge-Based On-Line Steiner Algorithm.* IEEE Journal on Selected Areas in Communications, 15(3):382–397, April 1997. (p 73)

[Bernet00] Y. Bernet, R. Yavatkar, P. Ford, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. *A Framework for Integrated Services Operation Over DiffServ Networks.* Internet Draft, March 2000. Work in progress. (pp 2, 4)

[Berson98] Steven Berson and Subramaniam Vincent. *Aggregation of Internet Integrated Services State.* In 6th IEEE/IFIP International Workshop on Quality of Service (IWQoS'98), Napa, California, USA, May 1998. (pp 2, 19)

[Bettati96] Riccardo Bettati and Amit Gupta. *Dynamic Resource Migration for Multiparty Real-Time Communication.* In Proceedings of the 16th International Conference on Distributed Computing Systems, Hong Kong, May 1996. (p 126)

[Blake98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services.* RFC 2475, December 1998. (p 19)

[Bolliger98] J. Bolliger and T. Gross. *A Framework-Based Approach to the Development of Network-Aware Applications.* IEEE Transactions on Software Engineering, 24(5):376–390, May 1998. (p 125)

[Bos99] Herbert Bos. *Elastic Network Control.* PhD thesis, Cambridge University Computer Laboratory, UK, August 1999. Available as Technical Report No. 483. (pp 7, 23, 43)

[Braden94] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: An Overview.* RFC 1633, June 1994. (p 18)

[Braden97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource ReSeravtion Protocol – Version 1 Functional Specification.* RFC 2205, September 1997. (p 18)

[Brunner99] Marcus Brunner and Rolf Stadler. *Virtual Active Networks—Safe and Flexible Environments for Customer-Managed Services.* In 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'99), Zurich, Switzerland, October 1999. (p 66)

[Buckley98] William P. Buckley. *Virtual Switch Interface (VSI) Implementation Agreement.* Available from `http://www.msforum.org/`, November 1998. (p 24)

[Callon99] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. *A Framework for Multiprotocol Label Switching.* Internet Draft, July 1999. Work in progress. (pp 16, 17)

[Campbell99] Andrew T. Campbell, Michael E. Kounavis, Daniel A. Villela, John B. Vicente, Hermann G. De Meer, Kazuho Miki, and Kalai S. Kalaichelvan. *Spawning Networks.* IEEE Network Magazine, 13(4):16–29, July/August 1999. (p 65)

[Cetinkaya00] Coskun Cetinkaya and Edward W. Knightly. *Egress Admission Control.* In *Proceedings of INFOCOM'00* [IEE00]. (p 20)

[Chandra98] Prashant Chandra, Allan Fisher, Corey Kosak, T.S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. *Darwin: Customizable Resource Management for Value-Added Network Services.* In Sixth IEEE International Conference on Network Protocols (ICNP'98), Austin, Texas, USA, October 1998. (p 63)

[Clark92] David D. Clark, Scott Shenker, and Lixia Zhang. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism.* Computer Communication Review, 22(4):14–26, October 1992. Proceedings of SIGCOMM August 1992. (p 18)

[de Prycker95] Martin de Prycker. *Asynchronous Transfer Mode: Solution for Broadband ISDN.* Prentice Hall, third edition, 1995. (pp 15, 16)

[Doar96] Matthew B. Doar. *A Better Model for Generating Test Networks.* In Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'96), pages 86–93, London, UK, November 1996. (p 79)

[Duffield99] N. G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K. K. Ramakrishnan, and Jacobus E. Van der Merwe. *A Flexible Model for Resource Management in Virtual Private Networks.* Computer Communication Review, 29(4):95–108, October 1999. Proceedings of SIGCOMM September 1999. (p 66)

[Durham00] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. *The COPS (Common Open Policy Service) Protocol*. RFC 2748, January 2000. (p 124)

[Elek00] Viktoria Elek, Gunnar Karlsson, and Robert Ronngren. *Admission Control Based on End-to-End Measurements*. In *Proceedings of INFOCOM'00* [IEE00]. (p 20)

[Engler95] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr. *Exokernel: An Operating System Architecture for Application-Level Resource Management*. In Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95), pages 251–266, Colorado, USA, December 1995. (p 122)

[Foster00] Ian Foster, Alain Roy, and Volker Sander. *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*. In Eighth International Workshop on Quality of Service (IWQoS'2000), Pittsburgh, USA, June 2000. (p 125)

[Fotedar95] Shivi Fotedar, Mario Gerla, Paola Crocetti, and Luigi Fratta. *ATM Virtual Private Networks*. Communications of the ACM, 38(2):101–109, February 1995. (p 2)

[Fox99] B. Fox and B. Gleeson. *Virtual Private Networks Identifier*. RFC 2685, September 1999. (p 12)

[Fraser00] Keir Fraser. *MPLS on Linux*. Available from `http://www.cl.cam.ac.uk/Research/SRG/netos/netx/`, September 2000. (p 40)

[Garey79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979. (p 77)

[Garrett94] Mark W. Garrett and Walter Willinger. *Analysis, Modeling and Generation of Self-Similar VBR Video Traffic*. Computer Communication Review, 24(4):269–280, October 1994. Proceedings of SIGCOMM August 1994. (p 37)

[Gleeson00] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. *A Framework for IP Based Virtual Private Networks*. RFC 2764, February 2000. (pp 11, 14)

[Gligor79] Virgil D. Gligor. *Review and Revocation of Access Privileges Distributed Through Capabilities.* IEEE Transactions on Software Engineering, SE-5(6):575–586, November 1979. (p 124)

[Gray93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993. (pp 98, 103)

[Grossglauser97] Matthias Grossglauser, Srinivasan Keshav, and David Tse. *RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic.* IEEE/ACM Transactions on Networking, 5(6):741–755, December 1997. (p 38)

[Hall94] Jane Hall, Per Vorm, Jens Meinkohn, Michael Tschichholz, Lennart Bjerring, Peter Levinsky, and Michel Louis. *D6.4A, Virtual Private Networks Vol. I, VPN State of the Art.* PREPARE project deliverable. Available at `http://www.fokus.gmd.de/minos/prepare/entry.html`, August 1994. (pp 1, 11)

[Hamzeh99] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. *Point-to-Point Tunneling Protocol (PPTP).* RFC 2637, July 1999. (p 11)

[Hand99] Steven M. Hand. *Self-Paging in the Nemesis Operating System.* In Usenix Third Symposium on Operating Systems Design and Implementation (OSDI'99), February 1999. (p 123)

[Harty92] Kieran Harty and David R. Cheriton. *Application-Controlled Physical Memory using External Page-Cache Management.* In Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'92), September 1992. (p 123)

[Heinanen99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. *Assured Forwarding PHB Group.* RFC 2597, June 1999. (p 19)

[Herbert94] Andrew Herbert, Ian Leslie, Derek McAuley, and Cosmos Nicolaou. *Scalable Distributed Control of ATM Networks.* Project proposal, University of Cambridge Computer Laboratory, UK, 1994. (p 6)

[Herzog00] S. Herzog. *Signaled Preemption Priority Policy Element.* RFC 2751, January 2000. (p 124)

[Hutchinson91] Norman C. Hutchinson and Larry L. Peterson. *The x-Kernel: An Architecture for Implementing Network Protocols.* IEEE Transactions on Software Engineering, 17(1):64–76, January 1991. (p 39)

[IEE00] IEEE. *Proceedings of INFOCOM'00*, Tel Aviv, Israel, March 2000. (pp 134, 135)

[IEEE98a] IEEE. *IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks (802.1Q)*, 1998. (p 13)

[IEEE98b] IEEE. *Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks —Common specifications—Part 3: Media Access Control (MAC) Bridges (802.1D)*, 1998. (p 13)

[Isaacs00] Rebecca Isaacs. *Lightweight, Dynamic and Programmable Virtual Private Networks.* In Proceedings of the Third Conference on Open Architectures and Network Programming (OPENARCH'00), pages 3–12, Tel Aviv, Israel, March 2000. IEEE. (p i)

[Isaacs01] Rebecca Isaacs and Ian Leslie. *Support for Resource-Assured and Dynamic Virtual Private Networks.* IEEE Journal on Selected Areas in Communications, 19(3):460–472, March 2001. Special Issue on Active and Programmable Networks. (p i)

[Isaacs99] Rebecca Isaacs and Richard Mortier. *INCA: Support for IN Using the Tempest.* In IEEE Global Telecommunications Conference (GLOBECOM'99), pages 812–816, Rio de Janeiro, Brazil, December 1999. (p 51)

[ITU93] The International Telecommunication Union. *Recommendation Q.700: Introduction to Signalling System No. 7*, 1993. (p 3)

[Jacobson88] Van Jacobson and Michael J. Karels. *Congestion Avoidance and Control.* Computer Communication Review, 18(4):314–329, August 1988. Proceedings of SIGCOMM August 1988. (p 126)

[Jacobson99] V. Jacobson, K. Nichols, and K. Poduri. *An Expedited Forwarding PHB.* RFC 2598, June 1999. (p 19)

[Karger88] Paul Karger. *Improving Security and Performance for Capability Systems.* PhD thesis, Cambridge University Computer Laboratory, UK, October 1988. Available as Technical Report No. 149. (p 124)

[Kent98] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol.* RFC 2401, November 1998. (p 12)

[Kompella93] Vachaspathi P. Kompella, Joseph C. Pasquale, and George C. Polyzos. *Multicast Routing for Multimedia Communication.* IEEE/ACM Transactions on Networking, 1(3):286–292, June 1993. (p 76)

[Kou81] L. Kou, G. Markowsky, and L. Berman. *A Fast Algorithm for Steiner Trees.* Acta Informatica, 15:141–145, 1981. (p 74)

[Lakshman96a] K. Lakshman and Raj Yavatkar. *Adaptive Resource Management for Multimedia Applications.* In W. Effelsberg, O. Spaniol, A. Danthine, and D. Ferrari, editors, *High-Speed Networking for Multimedia Applications.* Kluwer Academic Publishers, 1996. (p 125)

[Lakshman96b] K. Lakshman and Raj Yavatkar. *An Empirical Evaluation of Adaptive QOS Renegotiation in an ATM Network.* In 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'96), Zushi, Japan, 1996. (p 125)

[Lamport78] Leslie Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM, 21(7):558–565, July 1978. (p 99)

[Lazar95] Aurel A. Lazar, Shailendra K. Bhonsle, and Koon-Seng Lim. *A Binding Architecture for Multimedia Networks.* Journal of Parallel and Distributed Computing, 30(2):204–216, November 1995. (p 43)

[Lazar96] Aurel A. Lazar, Koon-Seng Lim, and Franco Marconcini. *Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture.* IEEE Journal on Selected Areas in Communications, 14(7):1214–1227, September 1996. (p 43)

[Lazar97] Aurel A. Lazar. *Programming Telecommunication Networks.* IEEE Network Magazine, pages 8–18, September/October 1997. (p 21)

[Leslie96] Ian Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul Barham, David Evers, Robin Fairbairns, and Eoin Hyden. *The Design and Implementation of an Operating System to Support Distributed Multimedia Applications.* IEEE Journal on Selected Areas in Communications, 14(7):1280–1297, September 1996. (p 25)

[Li99] Baochun Li and Klara Nahrstedt. *Dynamic Reconfiguration for Complex Multimedia Applications.* In Proceedings IEEE International Conference on Multimedia Computing and Systems (ICMCS'99), Florence, Italy, June 1999. (p 125)

[Lo00] Sai-Lai Lo, David Riddoch, and Duncan Grisby. *The omniORB Version 3.0 User's Guide.* AT&T Laboratories Cambridge, May 2000. Available from `http://www.uk.research.att.com/omniORB`. (p 28)

[Lucent99] Lucent Technologies. *VPNWorX: VPN Quick Reference Guide*, October 1999. Available at `http://www.lucent.com/vpnworx/vpnworxresource.html`. (p 13)

[Lynch96] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996. (p 103)

[Ma99] Qingming Ma. *Explicit QoS Route Selection in a Multi-Service Network.* In Hot Interconnects 7, Stanford University, California, USA, August 1999. (p 60)

[Martillo90] Joachim Martillo. *Routing in a Bridged Network.* White Paper, Telford Tools, Inc., 1990. (p 13)

[McAuley90] Derek McAuley. *Protocol Design for High Speed Networks.* PhD thesis, Cambridge University Computer Laboratory, UK, January 1990. Available as Technical Report No. 186. (p 3)

[McNamee90] Dylan McNamee and Katherine Armstrong. *Extending the Mach External Pager Interface to Accommodate User-Level Page Replacement Policies.* In Proceedings of the USENIX Association Mach Workshop, October 1990. (p 123)

[Mills91] David L. Mills. *Internet Time Synchronization: The Network Time Protocol.* IEEE Transactions on Communications, 39(10):1482–1493, October 1991. (p 99)

[Mortier00] Richard Mortier, Rebecca Isaacs, and Keir Fraser. *Switchlets and Resource-Assured MPLS Networks.* Technical Report 510, University of Cambridge Computer Laboratory, UK, May 2000. (pp 6, 29, 40)

[Moy91] J. Moy. *OSPF Version 2.* RFC 1247, July 1991. (p 126)

[Mullender93] Sape Mullender, editor. *Distributed Systems.* Addison-Wesley, second edition, 1993. (pp 103, 132, 143)

[Newman97] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall. *Ipsilon's General Switch Management Protocol Specification Version 1.1.* RFC 1987, August 1997. (p 21)

[Newman98a] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching, T. Lyon, and G. Minshall. *Ipsilon's General Switch Management Protocol Specification Version 2.0.* RFC 2297, March 1998. (p 24)

[Newman98b] Peter Newman, Greg Minshall, and Tom Lyon. *IP Switching: ATM Under IP.* IEEE/ACM Transactions on Networking, 6(2):117–129, April 1998. (pp 17, 21)

[Nichols98] K. Nichols, S. Blake, F. Baker, and D. Black. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474, December 1998. (p 19)

[Nichols99] K. Nichols, V. Jacobson, and L. Zhang. *A Two-bit Differentiated Services Architecture for the Internet.* RFC 2638, July 1999. (pp 2, 4, 19, 20, 67)

[Nikolaou96] C. N. Nikolaou, M. Marazakis, and G. Georgiannakis. *Transaction Routing for Distributed OLTP Systems: Survey and Recent Results.* Technical Report TR96-0165, Institute of Computer Science, Foundation for Research and Technology (ICS-FORTH), Crete, Greece, 1996. (p 104)

[OMG00] Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification (CORBA 2.4)*, October 2000. (pp 28, 39)

[OMG95] Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification (CORBA 2.0)*, 1995. (p 28)

[OpenGroup94] The Open Group. *X/Open Transport Interface (XTI), Version 2*, October 1994. Open Group Technical Standard. (p 39)

[Pan00] Ping P. Pan, Ellen L. Hahne, and Henning G. Schulzrinne. *BGRP: Sink-Tree-Based Aggregation for Inter-Domain Reservations.* Journal of Communications and Networks, 2(2), June 2000. (pp 4, 91)

[Parris94] Colin Parris, Hui Zhang, and Domenico Ferrari. *Dynamic Management of Guaranteed Performance Multimedia Connections.* ACM/Springer-Verlag Multimedia Systems Journal, 1:267–283, 1994. (p 126)

[Perlman00] Radia Perlman. *Interconnections: Bridges, Routers, Switches and Internetworking Protocols.* Addison-Wesley, second edition, 2000. (p 61)

[Python] *The Python Language.* Available from `http://www.python.org/`. (pp 56, 79)

[Ramanathan96] S. Ramanathan. *Multicast Tree Generation in Networks with Asymmetric Links.* IEEE/ACM Transactions on Networking, 4(4):558–568, August 1996. (p 75)

[Rayward-Smith83] V. J. Rayward-Smith. *The Computation of Nearly Minimal Steiner Trees in Graphs.* International Journal of Mathematical Education in Science and Technology, 14(1):15–23, January-February 1983. (p 75)

[Rayward-Smith86] V. J. Rayward-Smith and A. Clare. *On Finding Steiner Vertices.* Networks: An International Journal, 16:283–294, 1986. (p 75)

[Rekhter95] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4).* RFC 1771, March 1995. (p 91)

[Rooney98a] Sean Rooney. *The Structure of Open ATM Control Architectures.* PhD thesis, Cambridge University Computer Laboratory, UK, February 1998. Available as Technical Report No. 451. (pp 1, 7, 23, 27, 30, 104)

[Rooney98b] Sean Rooney, Jacobus E. van der Merwe, Simon A. Crosby, and Ian M. Leslie. *The Tempest: A Framework for Safe, Resource-Assured Programmable Networks.* IEEE Communications Magazine, 36(10):42–53, October 1998. (pp 7, 24)

[Rooney99] Sean Rooney, Christian Hortnagl, and Jens Krause. *Automatic VLAN Creation Based on On-Line Measurement.* Computer Communication Review, 29(3):50–57, July 1999. (p 13)

[Rosen99] Eric C. Rosen, Arun Viswanathan, and Ross Callon. *Multiprotocol Label Switching Architecture.* Internet Draft, August 1999. Work in progress. (pp 16, 22)

[Salama96] Hussein F. Salama. *Multicast Routing for Real-Time Communication on High-Speed Networks.* Master's thesis, North Carolina State University, Department of Electrical and Computer Engineering, November 1996. (p 75)

[Salama97] Hussein F. Salama, Douglas S. Reeves, and Yannis Viniotis. *Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks.* IEEE Journal on Selected Areas in Communications, 15(3):332–345, April 1997. (p 77)

[Seaman00] M. Seaman, A. Smith, E. Crawley, and J. Wroclawski. *Integrated Service Mappings on IEEE 802 Networks.* RFC 2815, May 2000. (p 13)

[Shaikh99] Anees Shaikh, Jennifer Rexford, and Kang G. Shin. *Load-Sensitive Routing of Long-Lived IP Flows.* Computer Communication Review, 29(4):215–226, October 1999. Proceedings of SIGCOMM September 1999. (p 121)

[Shapiro99] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. *EROS: A Fast Capability System.* In Proceedings of the Seventeenth Symposium on Operating Systems Principles (SOSP'99), December 1999. (p 124)

[Shenker97] S. Shenker, C. Partridge, and R. Guerin. *Specification of Guaranteed Quality of Service.* RFC 2212, September 1997. (p 18)

[Simpson95] W. Simpson. *IP in IP Tunneling.* RFC 1853, October 1995. (p 2)

[Smith99] Jonathan M. Smith, Kenneth L. Calvert, Sandra L. Murphy, Hilarie K. Orman, and Larry L. Peterson. *Activating Networks: A Progress Report.* Computer, 32(4):32–41, April 1999. (p 42)

[Stoica99] Ion Stoica and Hui Zhang. *Providing Guaranteed Services Without Per Flow Management.* Computer Communication Review, 29(4):81–94, October 1999. Proceedings of SIGCOMM September 1999. (p 19)

[Sun95] Quan Sun and Horst Langendorfer. *Efficient Multicast Routing for Delay-Sensitive Applications.* In Proceedings of the Second Workshop on Protocols for Multimedia Systems (PROMS'95), pages 452–458, October 1995. (p 77)

[Takahashi80] Hiromitsu Takahashi and Akira Matsuyama. *An Approximate Solution for the Steiner Problem in Graphs.* Mathematica Japonica, 24(6):573–577, 1980. (p 75)

[Tanenbaum96] Andrew S. Tanenbaum. *Computer Networks.* Prentice-Hall International, third edition, 1996. (p 3)

[Tennenhouse96] David L. Tennenhouse and David J. Wetherall. *Towards an Active Network Architecture.* Computer Communication Review, 26(2):5–18, April 1996. (p 42)

[Townsley99] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B.Palter. *Layer Two Tunneling Protocol L2TP.* RFC 2661, August 1999. (p 11)

[van der Merwe97a] Jacobus E. van der Merwe and Ian Leslie. *Switchlets and Dynamic Virtual ATM Networks.* In Aurel Lazar, Roberto Saracco, and Rolf Stadler, editors, *Integrated Network Management V*, pages 355–368, San Diego, USA, May 1997. IFIP & IEEE, Chapman & Hall. (pp 3, 7)

[van der Merwe97b] J.E. van der Merwe. *Open Service Support for ATM.* PhD thesis, Cambridge University Computer Laboratory, UK, September 1997. Available as Technical Report No. 450. (pp 1, 6, 7, 8, 16, 23, 27, 30, 31, 32, 35)

[van der Merwe98a] Jacobus E. van der Merwe and Ian M. Leslie. *Service Specific Control Architectures for ATM.* IEEE Journal on Selected Areas in Communications, 16(3):424–436, April 1998. (pp 47, 116)

[van der Merwe98b] Jacobus E. van der Merwe, Sean Rooney, Ian Leslie, and Simon Crosby. *The Tempest—A Practical Framework for Network Programmability.* IEEE Network Magazine, 12(3):20–28, May/June 1998. (pp 7, 24)

[Walrand96] Jean Walrand and Pravin Varaiya. *High-Performance Communication Networks.* Morgan Kaufmann, 1996. (p 98)

[Weihl93] William E. Weihl. *Transaction-Processing Techniques.* In Mullender [Mullender93], chapter 13, pages 329–352. (p 96)

[Weikum94] Gerhard Weikum, Christof Hasse, Axel Monkegerg, and Peter Zabback. *The Comfort Automatic Tuning Project.* In Information Systems, volume 19(5), pages 381–432. Elsevier Science, 1994. (p 104)

[White97] Paul P. White and Jon Crowcroft. *The Integrated Services in the Internet: State of the Art.* Proceedings of the IEEE, 85(12):1934–1946, December 1997. (p 18)

[Wilkes79] M. V. Wilkes and R. M. Needham. *The Cambridge CAP Computer and its Operating System.* North Holland, 1979. (p 124)

[Woods00] Darrin Woods. *ISP Service Level Agreements: Fishy Business.* Network Computing, July 2000. Available from `http://www.nwc.com`. (p 2)

[Worster00] Tom Worster, Avri Doria, Fiffi Hellstrand, and Kenneth Sundell. *General Switch Management Protocol.* Internet Draft, January 2000. Work in progress. (p 32)

[Wroclawski97] J. Wroclawski. *Specification of the Controlled-Load Network Element Service.* RFC 2211, September 1997. (p 18)

[Wulf81] William A. Wulf, Roy Levin, and Samuel P. Harbison. *HYDRA/C.mmp: An Experimental Computer System.* McGraw-Hill Book Company, 1981. (p 124)

[X/Open96] The Open Group. *Distributed TP: Reference Model, Version 3*, 1996. X/Open G504. (p 104)

[Yavatkar00] R. Yavatkar, D. Pendarakis, and R. Guerin. *A Framework for Policy-Based Admission Control.* RFC 2753, January 2000. (p 124)

[Yemini96] Yechiam Yemini and Sushil da Silva. *Towards Programmable Networks.* In IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'96), L'Aquila, Italy, October 1996. (p 68)

[Yuhara94] Masanobu Yuhara, Brian N. Bershad, Chris Maeda, and J. Eliot B. Moss. *Efficient Packet Demultiplexing for Multiple Endpoints and Large Messages.* In Proceedings of the 1994 Winter USENIX Conference, pages 153–165, January 1994. (p 41)

[Zhang00] Zhi-Li Zhang, Zhenhai Duan, Lixin Gao, and Yiwei Thomas Hou. *Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services.* Computer Communication Review, 30(4):71–83, October 2000. Proceedings of SIGCOMM August 2000. (p 67)

[Zhang93] Lixia Zhang, Stephen Deering, and Deborah Estrin. *RSVP: A New Resource ReSerVation Protocol.* IEEE Network Magazine, 7(5), September 1993. (pp 17, 18)