Guaranteeing QoS in an IP Network

Developing a Distributed SLA Admission Controller

by

Timothy R. Ducharme, B.Sc.
University of Victoria, 1999

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science
University of Victoria

© Tim Ducharme, 2004
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part,
by photocopy or other means, without the explicit permission of the author.

Supervisors: Dr. G. C. Shoja and Dr. E. G. Manning

ABSTRACT

This thesis deals with guaranteeing Quality of Service in an IP network using an SLA Admission Controller. Our research has shown that current MPLS implementations are not as agile or robust as required for an automated admission controller – one that processes highly dynamic fine-grained SLAs. Thus, we have designed and implemented a frame-scheduling algorithm that can support such requests and we have introduced a new technology that provides the granularity and agility needed to make an SLA Admission Controller useful in the real world. In the process, we have moved the notion of an optimal admission controller several steps further along the path from mathematical concept to a working software / hardware co-implementation.

In this thesis, we analyze the customer interface to an SLA Admission Controller – we work through the manual commissioning and provisioning of an MPLS network – and we discuss examples of an IPFS network and related protocols. We also provide specific details on how IPFS technology works and how QoS support is integrated. We conclude with developing both the IPFS frame scheduling algorithm and the signaling for a distributed SLA Admission Controller.

TABLE OF CONTENTS

ABSTRACT II

TABLE OF CONTENTS III

LIST OF TABLES V

LIST OF FIGURES VI

GLOSSARY OF TERMS VIII

ACKNOWLEDGEMENTS X

1 INTRODUCTION 1
- 1.1 The Purpose 1
- 1.2 The Problem 1
- 1.3 The Solutions: A Brief Outline 2
  - 1.3.1 QoSNET 3
  - 1.3.2 IPFSNET 4
- 1.4 Key Aspects 4
  - 1.4.1 Service Level Agreements 5
  - 1.4.2 Fixed-path Routing 5

2 BACKGROUND REVIEW 7
- 2.1 Communication Protocols 7
  - 2.1.1 IP 8
  - 2.1.2 Ethernet 9
  - 2.1.3 SONET/SDH 10
  - 2.1.4 ATM 11
  - 2.1.5 MPLS 12
- 2.2 Signaling Protocols 14
  - 2.2.1 LDP 14
  - 2.2.2 CR-LDP 14
  - 2.2.3 RSVP 15
  - 2.2.4 RSVP-TE 16
- 2.3 Admission Control 16
  - 2.3.1 Ad Hoc Method 17
  - 2.3.2 IntServ 17
  - 2.3.3 DiffServ 18
- 2.4 SLAOpt 19
  - 2.4.1 The Utility Model 19
  - 2.4.2 The Simulator 19

3 QOSNET 21
- 3.1 Admission Process 21
- 3.2 The Controller (SLACtl) 23
  - 3.2.1 Analysis 23
  - 3.2.2 External (Customer ↔ Controller) Interface 25
  - 3.2.3 Internal (Controller ↔ Network) Interface 28
- 3.3 The Network 32
  - 3.3.1 Implementation: A Brief Chronology 32
  - 3.3.2 Physical Architecture 33
  - 3.3.3 Commissioning / Configuring 33
  - 3.3.4 IP Architecture 35
  - 3.3.5 MPLS Architecture 38
  - 3.3.6 Packet Classification 41
- 3.4 Summary 42
  - 3.4.1 Lessons Learned 42
  - 3.4.2 Next Step 43

4 IPFSNET 44
4.1 IPFS 44
4.1.1 HRN 45
4.1.2 Node Configurations 47
4.1.3 ERP 47
4.1.4 Frame-Switching 48
4.2 Node Implementation 51
4.2.1 Data/Frame Flow 53
4.2.2 FPGA Design 55
4.2.3 Router/Microprocessor 56
4.2.4 Device Driver 57
4.2.5 Segmentation & Reassembly 59
4.3 QoS Support 59
4.3.1 Priority Scheme 60
4.3.2 Scheduling Scheme 61
4.3.3 Scheduling Granularity 64
4.4 Summary 66

5 THE CONTROLLER (DSLACTL) 67
5.1 Analysis 67
5.2 Architecture 68
5.2.1 Scheduling System 69
5.2.2 Signaling System 74
5.2.3 SLA Processing 80
5.3 Design 81
5.3.1 slad Daemon 82
5.3.2 pathd Daemon 83
5.3.3 Interfaces & Considerations 85
5.4 Admission Process 86
5.5 Summary 87

6 CONCLUSIONS & FURTHER WORK 89
6.1 Synopsis 89
6.2 Main Contributions 90
6.3 Further Work 90

REFERENCES 92

APPENDIX A 96

APPENDIX B 106

APPENDIX C 107

APPENDIX D 108

APPENDIX E 111

APPENDIX F 115

LIST OF TABLES

Table 3.1 – External Interface Requests/Responses 26
Table 3.2 – XML Messages → QOSMgr Methods 27
Table 4.1 – Example Schedule Table (partial) 63
Table 4.2 – IPFS Line / Data rates 64
Table 4.3 – IPFS Scheduling for Common Payloads 65

LIST OF FIGURES

Figure 2.1 – QoSNET Protocols ... 7
Figure 2.2 – IPFSNET Protocols ... 8
Figure 2.3 – IPv4 Datagram Structure ... 8
Figure 2.4 – ToS Field Definition ... 9
Figure 2.5 – Ethernet Frame Structure ... 10
Figure 2.6 – SONET OC-3c Frame Structure ... 11
Figure 2.7 – ATM UNI Cell Structure ... 12
Figure 2.8 – MPLS Label ... 12
Figure 2.9 – MPLS Network ... 13
Figure 3.1 – QoSNET Admission Process ... 22
Figure 3.2 – SLAOpt Main Class ... 23
Figure 3.3 – SLAOpt Structure Diagram - Main Modules ... 24
Figure 3.4 – SLAOpt Sequence Diagram - Initialization ... 24
Figure 3.5 – QoSNET External Interface ... 25
Figure 3.6 – nnMessageServer Class ... 26
Figure 3.7 – nnHandleSocket Class ... 27
Figure 3.8 – QoSNET Internal Interface ... 28
Figure 3.9 – Simple 9-node Network ... 29
Figure 3.10 – A Typical SLA for the 9-node Network ... 29
Figure 3.11 – A Revised SLA for the 9-node Network ... 31
Figure 3.12 – QoSNET Physical Architecture ... 33
Figure 3.13 – QoSNET Administration Plane ... 35
Figure 3.14 – QoSNET ATM Data Subplane ... 36
Figure 3.15 – QoSNET Ethernet Data Subplane ... 36
Figure 3.16 – QoSNET Control Plane ... 37
Figure 3.17 – UDP Traffic Test ... 38
Figure 3.18 – MPLS Paths ... 39
Figure 3.19 – LSPG Setup ... 40
Figure 3.20 – LSPs in an LSPG ... 40
Figure 4.1 – HRN Topology ... 45
Figure 4.2 – HRN Address ... 46
Figure 4.3 – G50 Node Configurations ... 47
Figure 4.4 – IPFS Frame Layout ... 47
Figure 4.5 – IPFS Frame Aggregation in an STS-3 SPE ... 48
Figure 4.6 – Frame-Switching Pseudo Code ... 49
Figure 4.7 – Example IPFS HRN for Unicast Frame-Switching ... 49
Figure 4.8 – Example ERP Frame with Prepended IPFS Header ... 50
Figure 4.9 – Extending an IPFS HRN Using a Legacy SONET Network ... 51
Figure 4.10 – G50 Block Diagram ... 52
Figure 4.11 – G50 Data Flow ... 53
Figure 4.12 – G50 IPFS Frame Flow ... 55
Figure 4.13 – G50 FPGA Design ... 56
Figure 4.14 – TxSched/TxFrame Pseudo Code ... 62
Figure 4.15 – Example HRN Sub-ring for Ring Scheduling ... 63
Figure 4.16 – Example STS-3 SPE with Scheduled IPFS Frames ... 63
Figure 5.1 – dSLACtl Architecture ... 68
Figure 5.2 – Scheduler Pseudo Code ... 70
Figure 5.3 – Network Layout: Generalized Multi-Ring Scheduling ... 71
Figure 5.4 – Network Layout: Schedule Aggregation ... 72
Figure 5.5 – Network Layout: Schedule Aggregation – adding a DS0 ... 73
Figure 5.6 – Request-Reply Protocol ... 74
Figure 5.7 – Two-Phase Commit Protocol ... 75
Figure 5.8 – Reserve, Commit, Free Scheduler Pseudo Code ... 76

Figure 5.9 – addPath Pseudo Code 77
Figure 5.10 – removePath Pseudo Code 78
Figure 5.11 – Network Layout: Generalized Multi-Ring Signaling 79
Figure 5.12 – addSLA Pseudo Code 80
Figure 5.13 – removeSLA Pseudo Code 81
Figure 5.14 – SLA/Path Request Statechart 82
Figure 5.15 – addSLA Statechart 83
Figure 5.16 – removeSLA Statechart 83
Figure 5.17 – addPATH Statechart 84
Figure 5.18 – removePATH Statechart 84
Figure 5.19 – dSLACtl Sequence Chart 85
Figure 5.20 – IPFSNET Admission Process 86
Figure 5.21 – dSLACtl Architecture with 'plug-in' Modules 87
Figure A.1 – Network Layout: IP datagram Host A → Host B 108
Figure A.2 – Network Layout: Multi-Ring Scheduling 111
Figure A.3 – Network Layout: Same Ring Scheduling 112
Figure A.4 – Network Layout: Sub-ring → Super-ring Scheduling 112
Figure A.5 – Network Layout: Super-ring → Sub-ring Scheduling 113
Figure A.6 – Network Layout: Single-level HRN 115
Figure A.7 – Single Ring ACK Sequence Chart 117
Figure A.8 – Single Ring NACK Sequence Chart 117
Figure A.9 – Network Layout: Two-level HRN 119
Figure A.10 – Sub-ring → Super-ring ACK Sequence Chart 121
Figure A.11 – Sub-ring → Super-ring NACK Sequence Chart 121
Figure A.12 – Super-ring → Sub-ring ACK Sequence Chart 123

GLOSSARY OF TERMS

ACK	Positive Acknowledgement
ADM	Add-Drop Multiplexer
AF	Assured Forwarding
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
ATM IP FP	OC-3 ATM IP Function Processor
BGP	Border Gateway Protocol
CAM	Content Addressable Memory
CBS	Committed Burst Size
CDR	Committed Data Rate
CoS	Class of Service
CP	CP2 Control Processor
CR-LDP	Constraint-Based Routing Using LDP
CSMA/CD	Carrier-Sense Multiple Access with Collision Detection
DiffServ	Differentiated Services
DSCP	DiffServ Code Point
dSLACtl	Distributed SLA Admission Controller
EF	Expedited Forwarding
egress	Destination Gateway
EMS	Element Management System
ER	Explicit Route
ER-MPLS	Explicitly Routed MPLS
ERP	Encapsulated Routing Protocol
Eth IP FP	Ethernet 100BASE-TX Function Processor
FEC	Forwarding Equivalence Class
FFL	Frame-Forwarding Logic
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
GUI	Graphical User-Interface
HRN	Hierarchical Ring Network
IEEE	Institute of Electrical & Electronics Engineers
IETF	Internet Engineering Task Force
ingress	Source Gateway
IntServ	Integrated Services
IP	Internet Protocol
IPFS	IP Frame Switching
ISP	Internet Service Provider
L2	Layer-2
L3	Layer-3
LAN	Local Area Network
LDP	Label Distribution Protocol
LER	Label Edge Router
LOH	Line Overhead
LSP	Label Switched Path
LSPG	Label Switched Path Group
LSR	Label Switched Router
LUT	Look Up Table
MAC	Medium Access Control

MAN	Metropolitan Area Networks
MDM	Multiservice Data Manager
MMKP	Multidimensional Multiconstraint Knapsack Problem
MPLS	Multiprotocol Label Switching
MSC	MPLS Service Category
NACK	Negative Acknowledgement
NAT	Network Address Translation
NIC	Network Interface Card
NMS	Network Management System
OC-n	Optical Carrier-level n
OSPF	Open Shortest Path First
PBS	Peak Burst Size
PDR	Peak Data Rate
PDU	Protocol Data Unit
PHB	Per-Hop Behavior
POH	Path Overhead
PP7K	Passport 7440 Router
QME	QoS Management Engine
QoS	Quality of Service
RARP	Reverse Address Resolution Protocol
RFC	Request For Comment
RIP	Routing Information Protocol
RR	Request-Reply Protocol
RSVP	Resource Reservation Protocol
RSVP-TE	RSVP-Traffic Engineering
RTOS	Real-Time Operating System
SDH	Synchronous Digital Hierarchy
SDS	Software Distribution Site
SLA	Service Level Agreement
SLACtl	SLA Admission Controller
SLAOpt	SLA Optimizer
SNMP	Simple Network Management Protocol
SOH	Section Overhead
SONET	Synchronous Optical Network
SPE	Synchronous Payload Envelope
STD	Standard
STS-n	Synchronous Transport Signal-level n
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TE	Traffic Engineering
TE tunnel	Traffic Engineered Tunnel
TLV	Type-Length-Value
TOH	Transport Overhead
ToS	Type of Service
UDP	User Datagram Protocol
UM	Utility Model
UNI	User-Network Interface
VCI	Virtual Circuit Identifier
VPCI	Virtual Path-Channel Identifier
VPI	Virtual Path Identifier
WAN	Wide Area Network

ACKNOWLEDGEMENTS

I would like to acknowledge some very special people who have helped me tremendously with this work. First and foremost is my bride, LJ; we connected, courted, and committed during the time of this research. She continues to be an inspiration and an encouragement – a true blessing from God and the love of my life.

I also wish to express my sincerest gratitude to my supervisors Dr. Gholamali C. Shoja and Dr. Eric G. Manning. Their insight and knowledge have provided focus for my work. I would like to thank the members of my committee: Dr. John Muzio, Dr. Kin Li, and Dr. Dale Shpak for their time and efforts.

I would also like to express my deepest appreciation to my parents Bob and Brenda Ducharme for always believing in me and urging me to do whatever my heart desires. To my friend Frank Schnurr, thank you for listening to me and continually prompting me. I also thank my Lambrick family and my breakfast club for their prayers and accountability.

I am exceedingly grateful for the considerable funding and support that was provided by the Natural Sciences and Engineering Research Council of Canada, University of Victoria, New Media Innovation Centre, Nortel Networks, and Syscor R&D. I would like to thank Tom Getty and Jeff Taylor, the Nortel support engineers who helped out with the acquisition and setup of the Passport 7440s.

Many people at Syscor R&D were involved in developing IPFS technology including Nick Tzonev, Dale Shpak, Pei-Chong Tang, David Sime, Grace Lin, Dilian Stoikov, Ron St. Pierre, Mike Gabelman, Sumio Kiyooka, Doreen Dinsdale, and Derek Heidom. I thank all of these individuals for their willingness to listen to my suggestions and for working together as hardware / software co-design and co-implementation teams.

To God

on whose strength I depend daily.

Πάντα δἰ αὐτοῦ ἐγένετο,
καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν.

John 1:3

All things through Him came into being,
and without Him came into being not even one.

Chapter 1
Introduction

In this chapter we introduce the purpose of our work, the problem as it exists today in backbone networks, and our solutions to this problem – QoSNET and IPFSNET.

1.1 The Purpose

This research started with an implementation of the *Utility Model* (UM) as the basis for an admission controller in an MPLS-enabled IP Network. The purpose was to demonstrate that an admission controller based on the Utility Model can and will guarantee effective fulfillment of QoS constraints. During this research, we identified several difficulties in our original approach and sought solutions that would ultimately address the core problem in today's backbone networks. Hence, the purpose changed to demonstrating how a distributed admission controller that is tightly integrated into a backbone network is able to guarantee fulfillment of QoS constraints.

1.2 The Problem

Over the last few years the demand on networks and inter-networks to transfer large amounts of data with real-time constraints has increased considerably and this trend, we believe, will continue. New applications that reflect the evolving needs of end-users,

such as streaming media, depend more and more on solid, stable, high-bandwidth, low-latency data communication channels. As a result, 'best-effort' datagram communication is no longer acceptable; rather, network *Internet Service Providers* (ISP – e.g. UUNet) are facing the challenge of providing guaranteed *Quality of Service* (QoS) to their customers. *Service Level Agreements* (SLA) provide a means for customers to contract with ISPs for their communication requirements. By accepting a particular agreement, the ISP is bound unconditionally to providing the level of service detailed in that agreement or face various penalties ranging from refunding fees to lawsuits; SLAs are not a new concept to ISPs nor to their customers. SLAs are actively used today as a means of contracting between the parties involved, specifying the allocation of large data pipes through backbone networks – i.e. large-grained static SLAs. Typically, the details of an SLA are agreed upon by representatives of each party sitting around a boardroom table – the customer's representative requests needed resources and the ISP representative offers various options along with price points. This process is tedious and time-consuming. It is because of the length of this process that only relatively few large-grained static SLAs are considered; typically, those with contract lifetimes in the order of months or even years.

On top of that, the very idea of guaranteeing QoS through a 'best-effort' datagram network is ludicrous; guarantees of any sort are contrary to the definition of 'best-effort'. *Best-effort*, as its name implies, only means that the network will do its best to deliver the traffic – datagrams might be delayed or dropped at any point – there are NO guarantees whatsoever. The only way QoS can be guaranteed in current backbone networks is to rely on the allocation schemes of lower layer protocols (such as ATM) requiring circuit provisioning – a time-consuming, labor intensive and costly process.

1.3 The Solutions: A Brief Outline

In the course of agreeing to a particular SLA, the ISP contracts to provide the level of service specified in the SLA. This is achieved by binding network resources to the SLA. One such method of binding network resources to an SLA is by pre-selecting a path (or circuit in the case of ATM) through the network, which all data packets associated with the SLA will follow.

Looking at this SLA process, we determined that there was a more time-efficient method. We could automate the contracting of SLAs and provide for more-dynamic finer-grained SLAs along the way. Our solution implements a network admission controller that simplifies the contracting of SLAs between ISPs and their customers. Our admission controller integrates with an IP network and automates the SLA process. Based on the allocation scheme of our controller, we then allocate the corresponding resources in the network to carry the IP datagrams.

We make the following initial assumptions:

1. *restricted access* – all data traffic is subject to our admission control
2. *centralized control* – the admission controller runs at one location only
3. *a failure-free network* – datagrams are not corrupted by node or link failure

All of the assumptions are unnecessary in our final solution using an IPFS network.

1.3.1 QoSNET

The combination of an actual MPLS-enabled IP network and our admission controller is the basis of QoSNET – a prototype network. This network was set up and the controller designed so that our conjectures could be verified. That is, we built QoSNET with the express purpose of demonstrating that our controller could automate the SLA process and guarantee QoS in an actual IP network.

Toll Expressway Analogy

QoSNET can be thought of as a toll expressway running parallel to an existing freeway. Admission onto this toll expressway is subject to paying a toll and being granted permission to use the expressway. The main benefit of paying a toll, from the customer's point of view, is that since access is limited by the controller, the controller determines how congested the expressway may become. Therefore, the controller can ensure that the expressway is not overly congested so traffic flows quickly – you can get where you are going without being slowed by traffic jams. Our first assumption – *restricted access* – translates into only authorized vehicles can actually use the expressway. The second assumption – *centralized control* – translates into one central authority that is responsible for determining who gets access (a central HQ, for example). Note this does not mean there is only one access point, only that requests for access must all go to central HQ. The third assumption – *a failure-free network* – translates into no traffic accidents.

1.3.2 IPFSNET

Building on the lessons learned from QoSNET, our final solution uses an *IP Frame Switching* (IPFS) network tightly integrated with our re-designed admission controller as the basis for IPFSNET – a development network. This testbed is used in the development of a commercial switching / routing product, of which our admission controller is an integral part. During the development process, this testbed was used to demonstrate both the functionality of the IPFS protocol in a real product and to assist in researching QoS issues.

Toll Expressway Analogy

IPFSNET can similarly be thought of as a toll expressway and it could be constructed parallel to an existing freeway, however unlike QoSNET, IPFSNET carries non-toll traffic as well. Admission to IPFSNET is not restricted to those paying a toll, but those paying will enjoy greater benefit than those traveling for free. The free traffic is subject to certain restrictions, such as preemption by toll-paying traffic to the point of being kicked off the expressway without reaching their destination. The main benefit in paying to access IPFSNET is that you are given priority over those traveling for free. The controller contracts with you that the expressway will not become overly congested so that you can get where you are going quickly. Our first assumption – *restricted access* – is relaxed with IPFSNET since free traffic can use it as well, but there are still restrictions to priority resources. The second assumption – *centralized control* – is relaxed with the design of a distributed admission control algorithm. The third assumption – *a failure-free network* – is also relaxed, as failures do happen in the real world.

1.4 Key Aspects

There are four key aspects to this work:

1. mapping SLAs → IP connections
2. fixed routing (how to achieve this)
3. consistent routing (same routing in the controller and in the real network)
4. forced routing (the controller specifies the path completely)

These can be grouped into two broad concepts, which are vital to understanding this thesis – *Service Level Agreements* and *Fixed-path Routing*.

1.4.1 Service Level Agreements

A *Service Level Agreement* (SLA) is a binding contract between a customer and a service provider. By agreeing to the terms of an SLA, the service provider guarantees that the agreed-upon service level will be met for the duration of the agreement; in return, the customer provides some monetary or other compensation to the service provider. This implies that an SLA consists of both technical specifications and business considerations. To be semantically correct, a request becomes an agreement only after both parties have agreed to the terms; we broaden the definition to include requests. SLAs can be *static* or *dynamic*. The large-grained SLA discussed above is an example of a static SLA. A dynamic SLA is one where a customer requests resources right when they need them and only for the period of time needed.

For this thesis, we use the term SLA as an abbreviation for a *dynamic, small-grained request for service*. Thus, an SLA is used to request resources from the admission controller. The SLA consists of a specification of endpoints, such as source and destination IP addresses, and a specification of QoS constraints, such as *bandwidth*, *latency*, and/or *jitter*. These QoS constraints must be met at every point as the traffic flows through the network in order for the SLA to be met. This is what we mean by *guaranteed* QoS – the network will meet all of the QoS constraints at every point in the path.

1.4.2 Fixed-path Routing

We assume that given the appropriate path routing specification, the network will set up this path, return some form of path identifier, and subsequently route all traffic marked with this path identifier through the associated path. In other words, SLAs assume *fixed-path routing*. To this end, the Internet standard routing protocol – hop-by-hop *Open-Shortest Path First* (OSPF) [28] – does not suffice as the underlying routing protocol in these IP networks; we need some type of fixed-path routing for our work. As such, we chose *Explicitly Routed Multiprotocol Label Switching* (ER-MPLS) [17] for QoSNET. For IPFSNET, fixed-path routing is implicit in its design so no special considerations are necessary. However, when multiple-path routing is provided in IPFS networks we will need a fixed-path routing algorithm.

The remainder of the thesis is organized as follows: Chapter 2 provides relevant background on the protocols used in our work, and reviews work in the area of admission control, such as IntServ, DiffServ, and ad hoc methods. Chapter 3 concentrates on our first solution – QoSNET – while Chapter 4 discusses the foundation for our second solution – IPFSNET. Our *distributed SLA Admission Controller* (dSLACtl) is developed in Chapter 5, and we conclude this thesis in Chapter 6 with a synopsis of our research, our main contributions to the field, and a look forward into ongoing work.

Chapter 2
Background Review

In this chapter we provide background information on communication and signaling protocols necessary for understanding this work. We also review related work in the area of admission control techniques, and we conclude with a brief discussion on SLAOpt and the Utility Model.

2.1 Communication Protocols

The communication protocols discussed in this section are *IP*, *Ethernet*, *SONET*, *ATM*, and *MPLS*. Excellent sources for protocol details are [40], [22], [23], and [5].

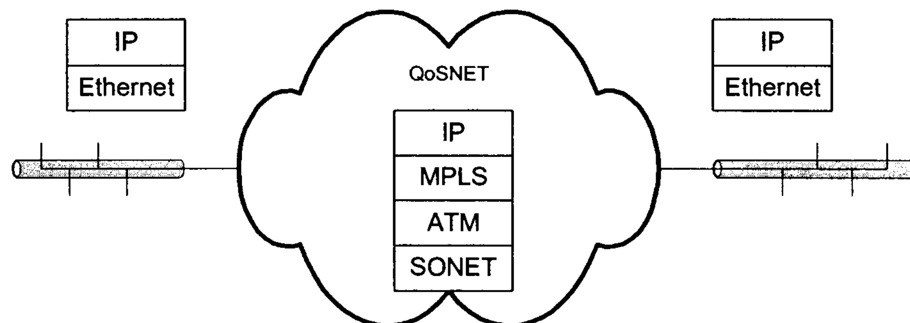

Figure 2.1 – QoSNET Protocols

Figure 2.1 shows the communication protocols used in QoSNET – at the edge: *IP* over *Ethernet*, in the core: *IP* over *MPLS* over *ATM* over *SONET*.

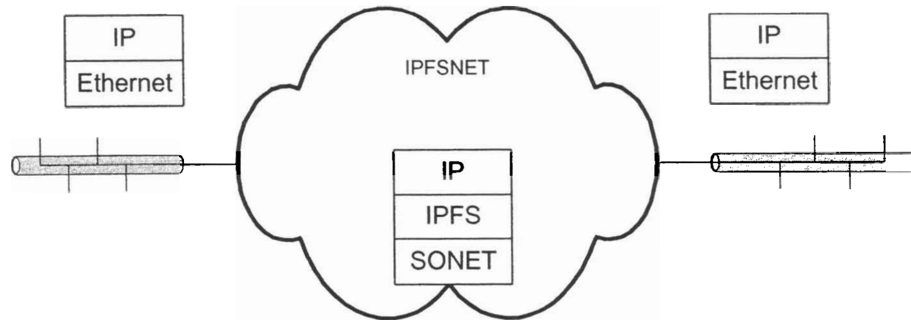

Figure 2.2 – IPFSNET Protocols

Figure 2.2 shows the communication protocols used in IPFSNET – at the edge: *IP* over *Ethernet*, in the core: *IP* over *IPFS* over *SONET.*

2.1.1 IP

The *Internet Protocol* (IP) [39] is the routing layer datagram service of the *Transmission Control Protocol* (TCP/IP) and the *User Datagram Protocol* (UDP/IP). IP is used to route frames from host to host over a network (e.g., the Internet). The IP frame header contains the routing and control information for IP datagram delivery. There are two versions of IP – IPv4 and IPv6. The IPv4 header is illustrated in Figure 2.3. IPv6 has larger addresses and other modifications that are irrelevant to our research.

4	8	16		32
Ver.	IHL	ToS	Total Length	
Identification			Flag	Fragment Offset
Time To Live		Protocol	Header Checksum	
Source Address				
Destination Address				
Option + Padding				
Data (0~65,530 Bytes)				

Figure 2.3 – IPv4 Datagram Structure

The important fields in the IP header for our discussion are the source/destination address fields and the ToS field.

IP Address

The source and destination addresses of IPv4 are 32 bits each and represent interconnected hosts – each host having a unique *IP address*, although this is not necessarily the case anymore with the advent of *Network Address Translation* (NAT). IP addresses are expressed in dotted decimal notation, e.g. 192.168.1.144. High-level modules map between host names and IP addresses, while lower level modules map between IP addresses and local network addresses. Routers and gateways map between local net addresses and routes, where a route indicates how to get to a host.

ToS

The *Type of Service* (ToS) [2] field indicates the type of service a datagram desires from a network. Networks may or may not consult this field when making routing or packet dropping decisions. The original intent of this field was to enable networks to offer service precedence – in times of high load, a network could consult this field to decide which datagrams to drop and which to forward. The bit definition of this field is given in Figure 2.4.

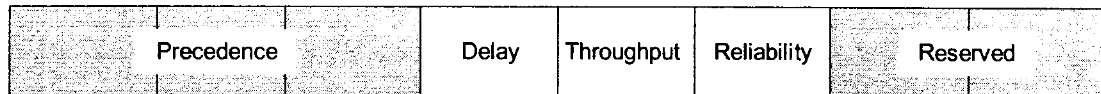

Figure 2.4 – ToS Field Definition

The *delay*, *throughput*, and *reliability* bits (3~5) are redefined as IP *Class of Service* (CoS). Most networks either ignore or re-mark the ToS field.

2.1.2 Ethernet

Ethernet refers to the family of local area network products covered by the IEEE 802.3 [21] standard that defines what is commonly known as the *Carrier-Sense Multiple Access with Collision Detection* (CSMA/CD) protocol.

Three common data rates for Ethernet are:

- 10 Mbps – 10BASE-T Ethernet
- 100 Mbps – Fast Ethernet (100BASE-TX, 100BASE-FX)
- 1000 Mbps – Gigabit Ethernet

We are primarily interested in 10BASE-T and 100BASE-TX as edge/access interfaces between *Local Area Networks* (LAN) and *Metropolitan Area Networks* (MAN) or *Wide Area Networks* (WAN). The Ethernet frame is illustrated in Figure 2.5.

6	6	2		4
Destination	Source	Len	*Data + padding (46~1500 Bytes)*	FCS

Figure 2.5 – Ethernet Frame Structure

MAC Address

The destination and source addresses are 48-bit Ethernet addresses, also called *Medium Access Control* (MAC) addresses. A MAC address is also known as a unicast address because it refers to a single device and is assigned by the *Network Interface Card* (NIC) manufacturer from a block of addresses allocated by the *Institute of Electrical & Electronics Engineers* (IEEE). Group addresses identify end stations in a workgroup and are assigned by the network manager, and a special group address (all 1s – the broadcast address) indicates all stations on the network. MAC addresses are expressed in hex format with either the digits separated by a colon, 00:40:05:1C:0E:9F, or separated by a dash, 00-40-05-1C-0E-9F. Ethernet MAC addresses are important for our discussions on IPFSNET.

2.1.3 SONET/SDH

Both *Synchronous Optical NETwork* (SONET – North American) [9] and *Synchronous Digital Hierarchy* (SDH – International/European) define a means for carrying many signals of different capacities through a synchronous, flexible, optical hierarchy. This is accomplished using a byte-interleaved multiplexing scheme which is based on integer multiples of the *Synchronous Transport Signal-level 1* (STS-1). STS-1 has a transmission speed of 51.84 Mbps and the STS-1 frame contains 810 octets (nine rows by 90 columns) – an octet is equivalent to an 8-bit byte. The *Transport Overhead* (TOH) contains the *Section Overhead* (SOH) and *Line Overhead* (LOH). The TOH uses the first three columns of the STS-1 frame and contains framing, error monitoring, management and payload pointer information. The *Synchronous Payload Envelope* (SPE) uses the remaining 87 columns, of which the first column is used for *Path Overhead* (POH), leaving 86 columns for data. A pointer in the TOH identifies the start of the payload.

Optical Carrier-level 3 (OC-3) and *Synchronous Transport Module-level 1* (STM-1) rates are an extension of the basic STS-1 speed and operate at 155.52 Mbps, carrying three interleaved STS-1 frames. OC-3c is the same size as OC-3 (three STS-1 frames), however it carries a single STS-3 frame. Thus, the OC-3c frame has nine rows and 270 columns. Nine of these columns are TOH and one column of the remaining 261 columns is used for POH. The SONET OC-3c frame is illustrated in Figure 2.6.

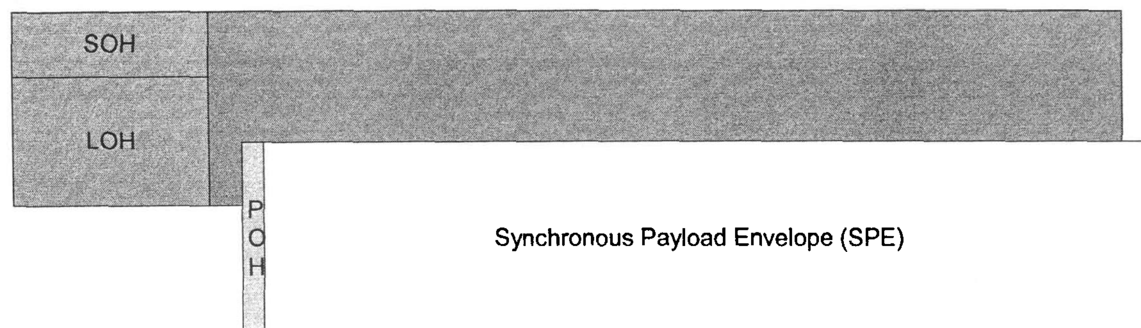

Figure 2.6 – SONET OC-3c Frame Structure

In order to account for clock skew and wander between the header and the payload, SONET/SDH allows the SPE to float inside the OC-3c frame. Pointers in the LOH point to the start of the SPE. There are no addresses in the SONET/SDH hierarchy as it is used to carry point-to-point *Time Division Multiplexing* (TDM) traffic over optical fibers. We are very interested in the SPE and the timing of SONET for IPFSNET, particularly at the OC-3c line rate.

2.1.4 ATM

Asynchronous Transfer Mode (ATM) [6] is a cell-switching and multiplexing technology that uses fixed-length cells – 53 bytes – to carry different types of traffic. ATM creates pathways between end nodes called virtual circuits, which are identified by *Virtual Path Identifier/Virtual Circuit Identifier* (VPI/VCI) values. The basic ATM *User-Network Interface* (UNI) cell structure is illustrated in Figure 2.7. Although there are many parts to ATM, we are only interested in the VPCI.

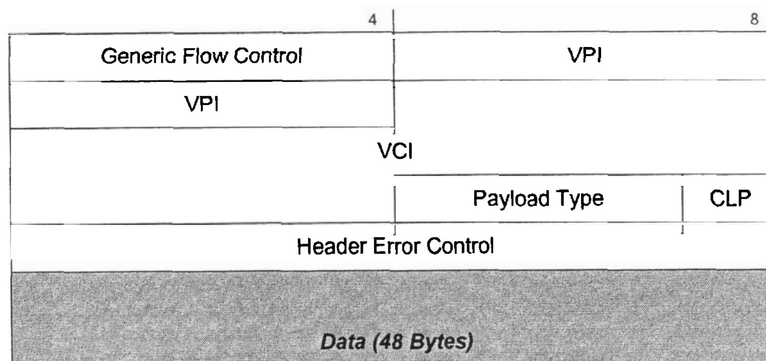

Figure 2.7 – ATM UNI Cell Structure

VPCI

Together, the VPI and VCI comprise the *Virtual Path-Channel Identifier* (VPCI) which represents the routing information for the ATM cell and identifies an end-to-end circuit through the ATM network. The VPCI is re-mapped at each ATM switch.

2.1.5 MPLS

Multiprotocol Label Switching (MPLS) [43] is an end-to-end forwarding technique which uses label-swapping to rapidly switch data traffic from *ingress* (source gateway) to *egress* (destination gateway) through a network. MPLS is layer-2 and layer-3 independent, meaning that various layer-3 routing techniques can interface to multiple layer-2 switched media through MPLS technology. For example, MPLS traffic can include IP, Frame Relay, ATM, and optical waveforms. Most significant is that layer-3 routing occurs at the edge of the network, and layer-2 switching takes over in the core. The MPLS labels are simple and fixed length (20-bits) and can be mapped easily to IP addresses; an MPLS label is illustrated in Figure 2.8. Labels can be 'stacked' in front of each other and in this case specify an *explicit-routed* path; the S bit is set to 1 to indicate the bottom of the stack.

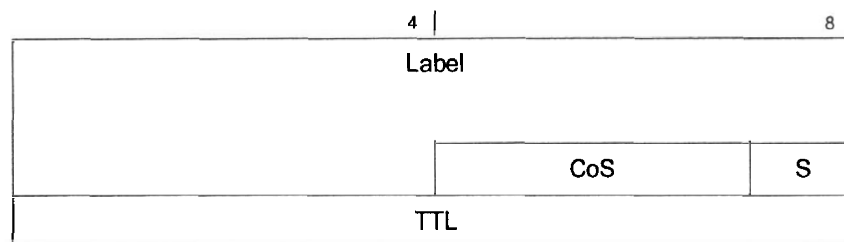

Figure 2.8 – MPLS Label

MPLS uses the concept of a *Forwarding Equivalence Class* (FEC) which is a partition of the address space. An ingress router determines which FEC a data packet belongs to and then prepends the appropriate MPLS label(s) to the front of the packet. As the packet moves through the network, MPLS swaps the label at each node on the route, according to a pre-defined label database at that node. The egress router decapsulates the packet and forwards it using the IP routing protocol. An MPLS network is shown in Figure 2.9. The edge nodes are called *Label Edge Routers* (LER) and provide ingress and egress functions for IP traffic; the core nodes are called *Label Switched Routers* (LSR) and provide high-speed switching functions for the network. The path of data between the MPLS nodes is a *Label Switched Path* (LSP), which is a unidirectional tunnel through the network.

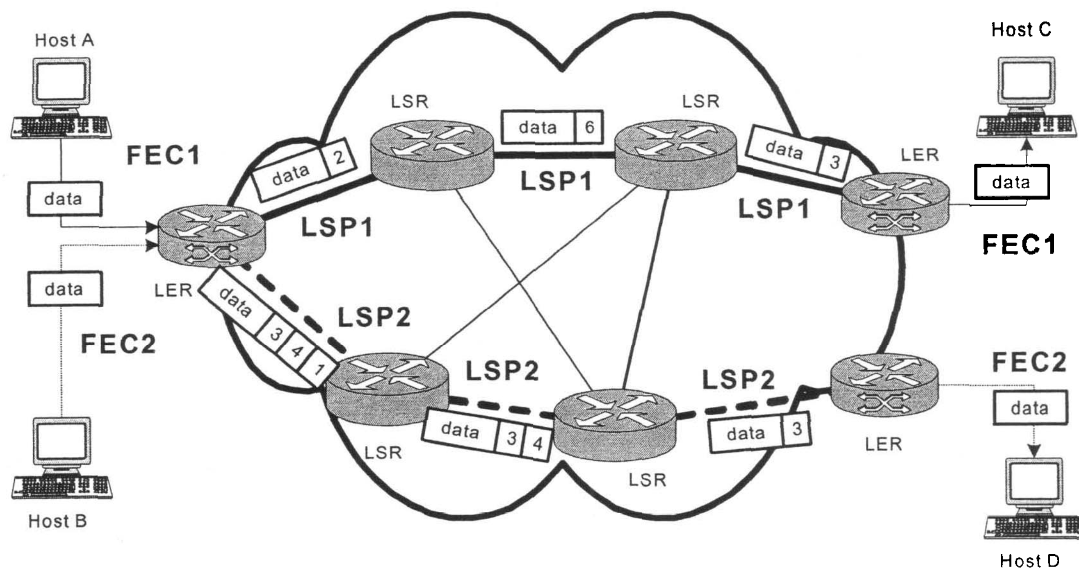

Figure 2.9 – MPLS Network

In Figure 2.9, FEC1 is mapped to LSP1, which is a *hop-by-hop* LSP – each LSR replaces the label before sending the packet to the next LSR. LSP2 is *explicit-routed* (in this case, *strict* ER) – the ingress LER prepends the complete label stack to the packet and each LSR pops off one label before forwarding the packet. MPLS also allows *loose* ER LSPs in which the LSP label stack is only partially specified. We use *strict Explicitly Routed MPLS* (strict ER-MPLS) in QoSNET.

2.2 Signaling Protocols

A signaling protocol is used to set up paths/circuits through a network. The signaling protocols discussed in this section are *LDP*, *CR-LDP*, *RSVP*, and *RSVP-TE*.

2.2.1 LDP

LSR/LERs must agree on the meaning of the labels used to forward traffic between and through them. *Label Distribution Protocol* (LDP) [4] defines a set of procedures and messages by which one LSR informs another of the label bindings it has made. The LSR uses this information to establish LSPs through a network. Two LSRs that use LDP to exchange label mapping information are known as *LDP peers* and they have an *LDP session* between them. A session is bi-directional, meaning that both LDP peers are able to learn about each other's label mappings.

LDP messages use a *Type-Length-Value* (TLV) encoding scheme; the value of a TLV-encoded object, may itself contain one or more TLVs. Messages are sent as LDP *Protocol Data Units* (PDU), and each PDU can contain more than one LDP message.

2.2.2 CR-LDP

Constraint-based Routing using LDP (CR-LDP) [24] extends LDP to allow for forwarding on the basis of constraints such as explicit routes or traffic parameters. CR-LDP adds four TLVs to the LDP protocol which are each very important to our research. The *Explicit Route* (ER) TLV contains a list of nodes that defines the path of an ER-LSP and is made up of one or more ER-hop TLVs. Each ER-hop TLV defines one hop in the ER, using an IP address prefix or a router identifier, and specifies whether the ER is strict or loose. The Traffic Parameters TLV defines the required characteristics of a constraint-based LSP using the following fields:

- *Peak Data Rate* (PDR) and *Peak Burst Size* (PBS) define the maximum rate at which data can be sent on the ER-LSP
- *Committed Data Rate* (CDR) and *Committed Burst Size* (CBS) define the rate at which the MPLS domain commits to being available to the ER-LSP
- *frequency* constrains the amount of variable delay that the network can introduce into the CDR

When an LSR receives a Traffic Parameters TLV with a label request, MPLS negotiates with the layer-2 software to reserve the requested bandwidth, if available. The LSP Identifier TLV provides a unique identifier for the LSP within the MPLS network. This TLV is needed in the case of ER-MPLS as the base MPLS implementation does not use network-wide unique identifiers, only peer-to-peer unique labels. CR-LDP permits *Traffic Engineering* (TE) to help manage large networks.

2.2.3 RSVP

Resource reSerVation Protocol (RSVP) [14] is a signaling protocol used to reserve resources in a network. Through the use of PATH and RESV messages, a flow is setup and resources are reserved. The PATH message is initiated by the sender of the flow, and can contain a *FlowSpec* – a specification of the required traffic flow characteristics. This PATH message travels through the network, being forwarded by each intermediate router until it reaches the receiver. Resource reservation does not occur in response to the PATH message; instead, it is through the use of a RESV message (containing a FlowSpec) which the receiver sends back to the sender, that the required resources are actually reserved. When an intermediate router receives the RESV message, it tries to reserve the resources specified in the FlowSpec. If the RESV request fails at any point then the receiver is notified and the RSVP signaling stops. If the RESV request is successful then bandwidth and buffer space are allocated by the router and the RESV request is sent to the next upstream router.

RSVP is a soft-state protocol, meaning that PATH messages must be periodically resent from the sender and RESV messages must be periodically resent from the receiver to maintain the reservation of resources. These PATH/RESV messages normally follow the same path as those used initially, from the sender through each intermediate router to the receiver and then back through each router to the sender of the flow. If a node does not receive a RESV message before a specified timeout then the resources are freed and the reserved flow is lost. RSVP has been extended to support aggregation of flows [8] and traffic engineering.

2.2.4 RSVP-TE

The *RSVP-Traffic Engineering* (RSVP-TE) [7] protocol is an addition to the RSVP protocol with extensions for setting up LSPs through an MPLS network. The ingress node for an LSP assigns a particular label to a set of packets; this label defines the flow through the LSP. Such an LSP is called an *LSP Tunnel* because the traffic flow through it is transparent to intermediate nodes. The LSP Tunnel object implies that traffic belonging to the LSP tunnel can be identified solely on the basis of packets arriving from the previous hop with the particular label value(s) assigned by this node to upstream senders to the session. For traffic engineering applications, sets of LSP tunnels can be associated for reroute operations or for spreading a traffic trunk over multiple paths; such sets are called *Traffic Engineered tunnels* (TE tunnel). Two identifiers are used, a tunnel ID and an LSP ID, to identify a TE tunnel.

CR-LDP and RSVP-TE both essentially perform the same function in regards to MPLS. They both contain a specification of traffic parameters and provide similar signaling to setup an LSP. RSVP-TE, as the most-widely implemented by industry, has been adopted by the *Internet Engineering Task Force* (IETF) as the signaling protocol of choice for MPLS [3]. However, we use CR-LDP in QoSNET because that is what the manufacturer of our MPLS nodes supports. For IPFSNET, we developed a proprietary signaling protocol as part of this work.

2.3 Admission Control

An excellent resource which brings together most of the work on QoS issues is "Internet QoS: A Big Picture" [48]. This work is summarized here as it relates to our research. There is much contention of whether quality of service really needs to be guaranteed. The argument is that fiber is cheap and with new technologies, such as *Dense Wave Division Multiplexing* (DWDM), network communications will soon become a relatively unlimited resource – there will be enough cheap bandwidth available that QoS will be delivered naturally. Our contention is that no matter how big a network becomes – how much bandwidth is provided – it will still be a limited resource. That is, as the supply increases the demand will also increase. There will still be a need to allocate the limited

resources so customers can be guaranteed their desired quality of service. Thus, the focus of this work – guaranteeing Quality of Service in an IP network – is achieved by controlling access to the fixed resources in the network – i.e. by *Admission Control.*

2.3.1 Ad Hoc Method

In this method there is minimal admission control. The network is setup initially to provide interconnectivity; then as usage dictates, it is re-configured to match the latest communication characteristics. If a customer wants some form of assured bandwidth, for example, the network is manually re-configured to support the request; this is usually done by over-allocating and leaving resources under-utilized. With the advent of technologies that provide provisioning capabilities, such as ATM, a large-pipe (VPCI) is provisioned so that a particular customer's data traffic will flow inside. This large-pipe is allocated at or near the expected peak data rate; otherwise the service degrades as the traffic rate exceeds the provisioned level of the pipe. However when the traffic rate drops below the provisioned rate of the pipe, valuable network resources become under-utilized. Overall the network becomes less and less efficient as more and more large-pipes are provisioned. Even today, this method is largely employed throughout backbone networks.

2.3.2 IntServ

Integrated Services (IntServ) [13] has three service classes: *Best Effort service*, *Controlled-Load service*, and *Guaranteed service.* With Controlled-Load service [46] a traffic flow receives the same level of service as that experienced in a lightly loaded network. This is really an enhanced and more reliable Best Effort service; however, it breaks down as the network becomes loaded. Guaranteed service [44] is for flows requiring fixed delay bounds and uses a leaky bucket approach at each router to regulate flows. For Guaranteed service or Controlled-Load service, an application must set up the path and reserve resources before transmitting its data.

IntServ has four main components: the signaling protocol, the admission controller, the classifier, and the scheduler. IntServ relies on reserved resources and call setup; it uses RSVP as its signaling protocol. The admission controller decides whether a request for

resources will be granted; the classifier determines in which queue to place an incoming packet; and the scheduler schedules the packet for transmission according to its QoS requirements.

IntServ has been criticized for its high demands on routers to maintain state information on a per flow basis and for processing overhead to support RSVP, admission control, classification, and scheduling. Although incremental deployment of IntServ is possible with Controlled-Load service, for Guaranteed service, every router in the network must be IntServ capable. This combined with the fact that RSVP uses soft-state – which places extra signaling demand on the network – translates into a non-scalable architecture.

2.3.3 DiffServ

Because IntServ is difficult to implement and deploy, and because it doesn't scale well, *Differentiated Services* (DiffServ) [11] was introduced. DiffServ is motivated by scalability, flexibility, and 'better than best-effort service' without RSVP signaling.

DiffServ aggregates flows into classes, where each class receives a certain 'treatment'; the IP ToS field is renamed the *DiffServ Code Point* (DSCP) [29] and is used to mark a packet as belonging to a certain class. DiffServ also defines a base set of packet forwarding 'treatments' – *Per-Hop Behaviors* (PHB) [20]. Two common PHBs are *Expedited Forwarding* (EF) and *Assured Forwarding* (AF). EF specifies that the departure rate of the class of traffic from the router must equal or exceed a configured rate independent of the traffic intensity of any other classes – i.e. a minimum guaranteed bandwidth. AF divides traffic into four classes where each AF class is guaranteed some minimum resources (bandwidth, buffering). Within each class, packets are partitioned into one of three 'drop preference' categories. Congested routers then drop/mark based on these preference values – i.e. a relative-priority scheme.

There are two significant differences between DiffServ and IntServ. First, since there are only a limited number of service classes and since service is allocated in the granularity of a class, the amount of state information is proportional to the number of classes rather than the number of flows; therefore, DiffServ is more scalable. Second, complex classification, marking, policing and shaping functions are only necessary at the edges of

the network – core routers only need to implement *behavior aggregate* classification; therefore, DiffServ is easier to implement and deploy than IntServ.

Since the DSCP (ToS field) is ignored by non-DiffServ routers, AF can be incrementally deployed; the AF packets will be treated as best effort by non-DiffServ routers and will have better overall performance since they are less likely to be dropped. However, EF requires that all routers be DiffServ-capable.

DiffServ requires customers to have an SLA with their ISP, which specifies the service classes supported and the amount of traffic allowed in each class. These SLAs can be static or dynamic; static SLAs are negotiated on a regular basis, whereas dynamic SLAs must use a signaling protocol, such as RSVP [10], to request services as needed.

2.4 SLAOpt

2.4.1 The Utility Model

In [25] Khan considered the problem of optimal allocation of the resources of a single multimedia server, while meeting the QoS requirements of individual sessions. He then showed how the problem could be mapped onto a variant of the combinatorial knapsack problem, with server utility as the quantity to be optimized and with QoS requirements expressed as constraints on resource allocation. Both optimal but slow (algorithmic) and fast but suboptimal (heuristic) methods were given as solutions to the *Multidimensional Multiconstraint Knapsack Problem* (MMKP) – we refer to these methods as the *Utility Model* (UM).

2.4.2 The Simulator

In [45] Watson applied the UM to the problem of optimal allocation of the resources of a packet network, with overall network utility as the quantity to be optimized and with the QoS requirements of bandwidth and latency as constraints on network resource allocation. Furthermore, he constructed a simulator, *SLA Optimizer* (SLAOpt), to demonstrate the feasibility of such an approach. The result of his work showed that network utilization in excess of 80% (77 400 units attained vs. 82 000 units maximum possible ≅ 94.4%) could be attained in a simulated environment. With such encouraging

results, as compared to a typical utilization in the range of 20~30%[*] for a real-world network, we wanted to extend this work to an actual implementation – integrating Watson's simulator into an admission controller that would interface and control a real IP network. This extension formed the foundation for QoSNET, which we discuss in the next chapter.

[*] This figure comes from personal correspondence with industry professionals.

Chapter 3 QoSNET

QoSNET is a private prototype network used to research *Quality of Service* (QoS) guarantees between content providers and content users. The QoS parameters we are concerned with are bandwidth and latency. We do not consider jitter at this time. The QoSNET concept is to use an *SLA Admission Controller* (SLACtl) to allow/disallow traffic streams through an IP network. A customer requests admission by specifying the stream parameters (endpoints, BW, latency) using *Service Level Agreements* (SLA). SLACtl determines if it can admit the new stream while respecting current SLAs and routes appropriately.

There are two parts to QoSNET: the *controller*, and the *network*. We detail each of these in the following sections. However, before we discuss the inner-workings of QoSNET, it is important to understand how QoSNET interfaces to the real world – i.e. how a customer interacts with QoSNET.

3.1 Admission Process

As illustrated in Figure 3.1, a customer wishing to be granted access to QoSNET first determines their QoS needs – step 1. From these needs, the customer creates an SLA, and requests admission to QoSNET using this SLA – step 2. SLACtl processes this SLA

admission request and determines if this request can be fulfilled without interfering with the currently admitted streams in the network. During this process, SLACtl will find the optimal route through the network for the new stream, if one is possible – step 3. If the SLA request cannot be fulfilled, then a *negative acknowledgement* (NACK) is returned to the customer and the process ends. However, if a suitable path is found through the network, then SLACtl instructs the network to setup this path by completely specifying the path routing – step 4a. The network builds the path and *acknowledges* (ACK) success back to SLACtl – step 4b. SLACtl then instructs the customer that the SLA request has been granted– step 4c. The customer can then start sending data on this path – step 5 – and be assured that as long as they stay within their requested QoS parameters, their data will get through the network at the agreed upon service level.

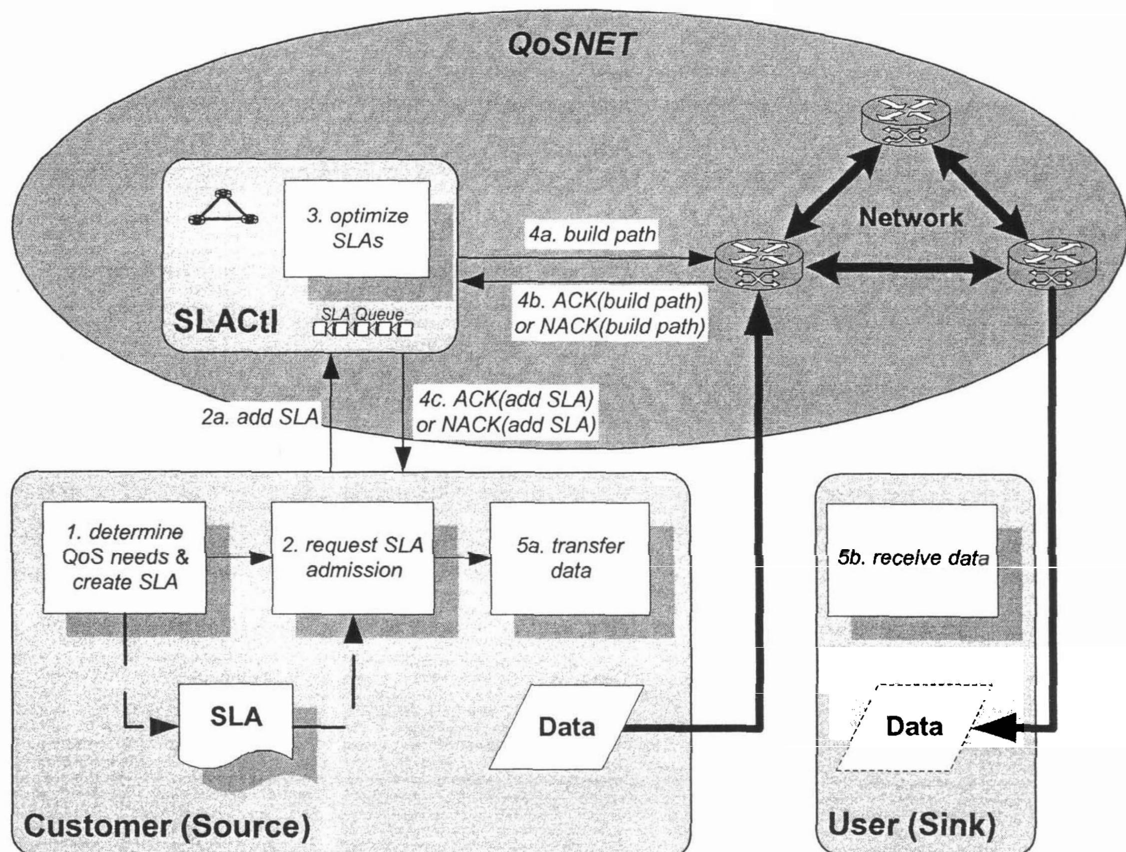

Figure 3.1 – QoSNET Admission Process

With this overview of how the admission process works, we can now discuss our admission controller – SLACtl.

3.2 The Controller (SLACtl)

Our first attempt at a solution to the guaranteed QoS problem as described in Section 1.2 was to adapt and modify Watson's SLAOpt simulator [45] to interface with a real IP network, so that we could control the admission of traffic into the IP network. With the goal of integrating the SLAOpt *QoS Management Engine* (QME) into SLACtl, we started with analyzing the simulator created by Watson. In this analysis, we looked at potential differences in concept between SLAOpt and SLACtl. We also investigated how SLAOpt worked, what were its main components, and how we could extract the basic modules that we needed from the framework used to operate in the simulated environment.

3.2.1 Analysis

SLAOpt was written in Java, so our first task in reverse engineering was to look at the main class, which is given in Figure 3.2.

```
public class SLAOpt {
    private nnNetwork myNetwork;
    private nnQOSMgr  myQOSMgr;
    private int port = 5060;

    public SLAOpt() throws FileNotFoundException, IOException {
        System.out.println("Starting up...");

        System.out.println("Reading network...");
        myNetwork = new nnNetwork("nodes.dat", "links.dat");
        myQOSMgr  = new nnQOSMgr(myNetwork);

        nnGUI myGUI = new nnGUI(myNetwork, myQOSMgr, ...);
        myQOSMgr.setGUI(myGUI);

        nnMessageServer myMessageServer = new nnMessageServer(myQOSMgr, port);
        System.out.println("Starting server on port " + port + "...");
        new Thread(myMessageServer).start();
    }
}
```

Figure 3.2 – SLAOpt Main Class

The main class shows the interaction between the main modules. From this class we see that SLAOpt contains a network (nnNetwork), a QoS manager (nnQOSMgr), a graphical user-interface (nnGUI), and a message server (nnMessageServer). The QoS manager is what we have referred to above as the QME. The structure diagram in Figure 3.3 illustrates the relationships between these classes.

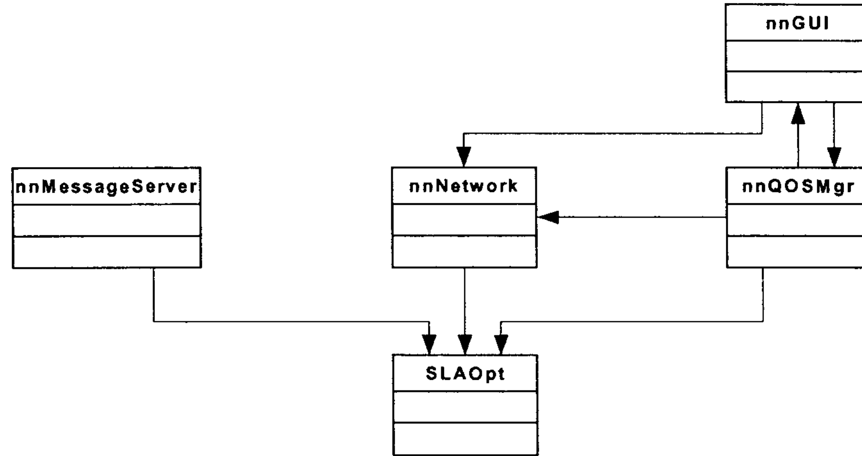

Figure 3.3 – SLAOpt Structure Diagram - Main Modules

SLAOpt
nnNetwork
nnQOSMgr
new(nodes:String, links:String)
readFile("nodes.dat")()
readFile("links.dat")()
return()
new(myNetwork:nnNetwork)
return()
nnGUI
new(myNetwork:nnNetwork, myQOSMgr:nnQOSMgr)
return()
setGUI(myGUI:nnGUI)
return()
nnMessageServer
new(myQOSMgr:nnQOSMgr, port:int)
return()
start()

Figure 3.4 – SLAOpt Sequence Diagram - Initialization

The interaction diagram in Figure 3.4 shows the initialization process of SLAOpt; by stepping through this initialization process, we can see that the main modules are not separated from each other. First, the simulated network is constructed from two files: nodes.dat and links.dat. These two files contain the description of the architecture of the simulated network on which SLAOpt operates. Next, the QME is initialized with the simulated network. Then, the *graphical user-interface* (GUI) is initialized with both the network and the QME; this is so that the GUI can display the changing network state as SLAs are admitted, and also for accepting manually entered SLAs. Next, the QME is told about the GUI so that the QME can change the display as SLAs are admitted. Finally, the message server is initialized and interfaced to the QME. This message server listens for TCP/IP connections on the given port and processes SLA admission requests.

3.2.2 External (Customer ↔ Controller) Interface

In a previous section we discussed the admission process – i.e., how a customer requests access to QoSNET, how a decision is made concerning that request, and then how the customer is informed about the decision. The interaction between a customer and SLACtl describes the basic external interface to QoSNET; this interface is shown in Figure 3.5.

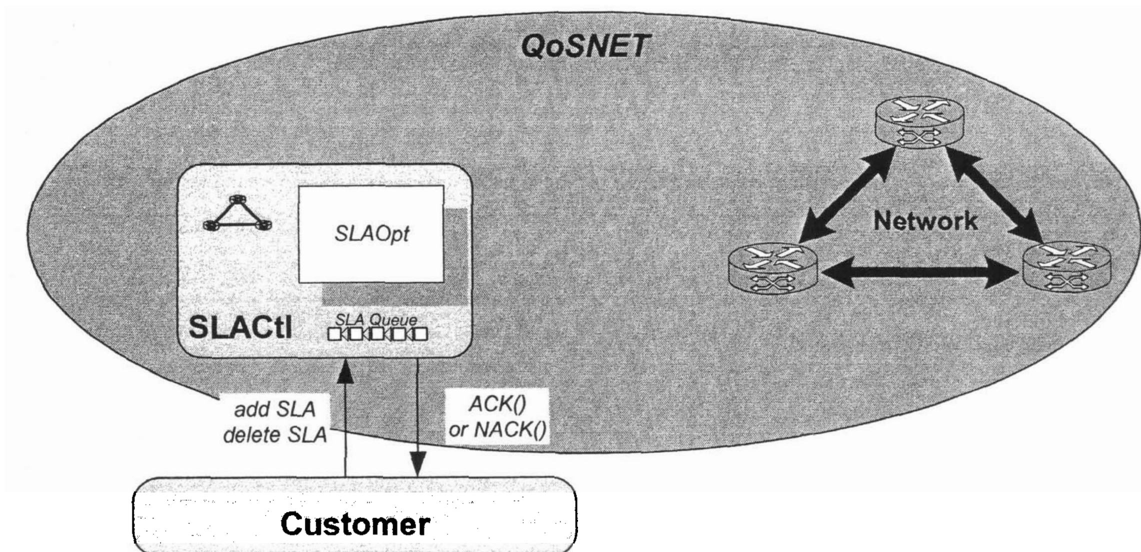

Figure 3.5 – QoSNET External Interface

The foundation for this interface was already in place – using the message server – although we need to modify it. The external interface will have to address the requests/responses listed in Table 3.1.

Table 3.1 – External Interface Requests/Responses

Request	Response
addSLA	ACK(addSLA)
	NACK(addSLA)
deleteSLA	ACK(deleteSLA)
	NACK(deleteSLA)

The deleteSLA request is important to our controller because it allows for releasing resources that are no longer needed by a customer. Also, the ACK and NACK responses provide feedback to the customer about whether or not they can use QoSNET.

```
public class nnMessageServer implements Runnable {
    nnQOSMgr  theQOSMgr;
    ServerSocket theServer;
    int port = 5060;
    boolean keepServing = true;

    public void serveMessages() {
        try {
            theServer = new ServerSocket(port);
            while (keepServing) {
                Socket myRequestor = theServer.accept();
                nnHandleSocket myHandler = new nnHandleSocket(
                                                    myRequestor, theQOSMgr);
                new Thread(myHandler).start();
            }
        } catch (Exception e) {...}
    }
}
```

Figure 3.6 – nnMessageServer Class

Looking at the message server class of SLAOpt given in Figure 3.6, we discover that it uses an nnHandleSocket object to process requests appearing on TCP/IP sockets. The nnHandleSocket object, given in Figure 3.7, processes XML documents presented on the input stream of the TCP/IP socket. The commands that nnHandleSocket can process are specified as the XML message type.

```
class nnHandleSocket implements Runnable {
    Socket        theRequestor;
    nnQOSMgr  theQOSMgr;

    public void handleMessage(Document theDoc) {
        Node message = theDoc.getDocumentElement();
        String messageType = ((Element)message).getAttribute("type");

        if (messageType.equals("addSLA")) {
            ...
            theQOSMgr.admit(mySLAVector);
        } else if (messageType.equals("deleteSLA")) {
            ...
            theQOSMgr.remove(mySLAVector);
        } else if (messageType.equals("changeSLAQOS")) {
            ...
        } else if (messageType.equals("addSLAQOS")) {
            ...
        }
    }
    public void run() {
        try {
            ...
            Document doc = docBuilder.parse (theRequestor.getInputStream());
            ...
            handleMessage(doc);
        } catch (Exception e) {...}
    }
}
```

Figure 3.7 – nnHandleSocket Class

The valid message types are:

- addSLA
- deleteSLA
- addSLAQOS
- changeSLAQOS

We are only interested in the addSLA and deleteSLA message types. For each of these message types, the nnHandleSocket object parses the SLA parameters from the XML document and invokes the QME to perform the action as listed in Table 3.2.

Table 3.2 – XML Messages → QOSMgr Methods

XML Message Type	nnQOSMgr method
addSLA	public void admit(Vector theSLAs)
deleteSLA	public void remove(Vector theSLAs)

Looking at these method signatures we see that the QME does not report on the success or failure of its admit or remove methods (both methods return void). In other words, the existing message server has no way of reporting an ACK or NACK back to the original caller. This response infrastructure needs to be added to the external interface, which will require changing the nnQOSMgr and nnHandleSocket objects to return the appropriate responses. Other than these modifications, the external interface to SLAOpt through the message server will suffice for SLACtl.

3.2.3 Internal (Controller ↔ Network) Interface

QoSNET also has an internal interface, which is between SLACtl and the IP network; this interface is shown in Figure 3.8. SLAOpt operates on an internal simulated network, which is contained in the nnNetwork object. Since this network model is internal to SLAOpt, we need a way of linking this model network to our real network. We need to build an interface object that can translate between commands presented to the simulated network and commands to mirror the results in the real network.

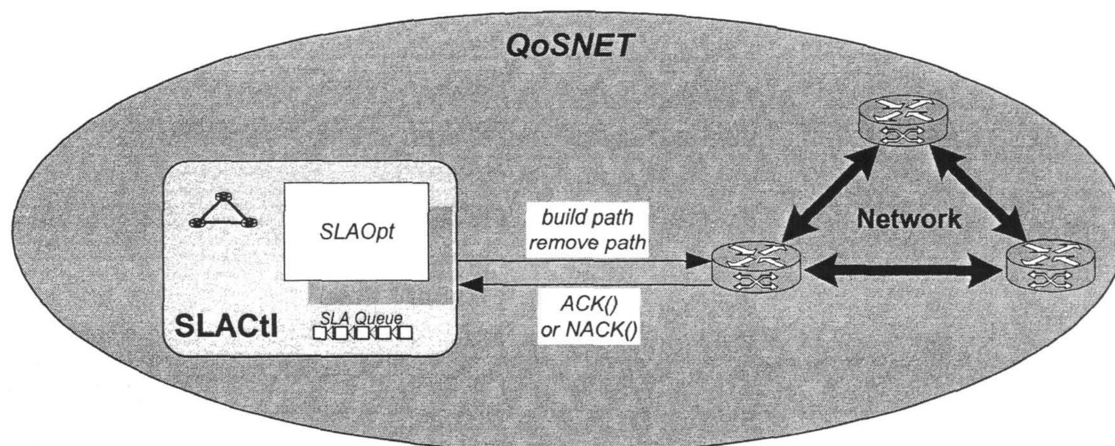

Figure 3.8 – QoSNET Internal Interface

The real network itself, including the configuration / commissioning process of the backbone routers, is detailed in the next section. However, there are some basic concepts that are inherently different between the simulated network and the real network. Looking at the model of the network used in SLAOpt, we identify that a simplified view of a network is used – nodes have names and the concept used is that these nodes produce and consume data traffic. This simplified idea is fine for a simulation, but in a real network, one in which our controller operates, the nodes belong to an autonomous

network and only route traffic between the edges of the network. The data traffic is produced and consumed external to the network. This may not seem to be a big difference, however in the following discussion we will see how important these differences are.

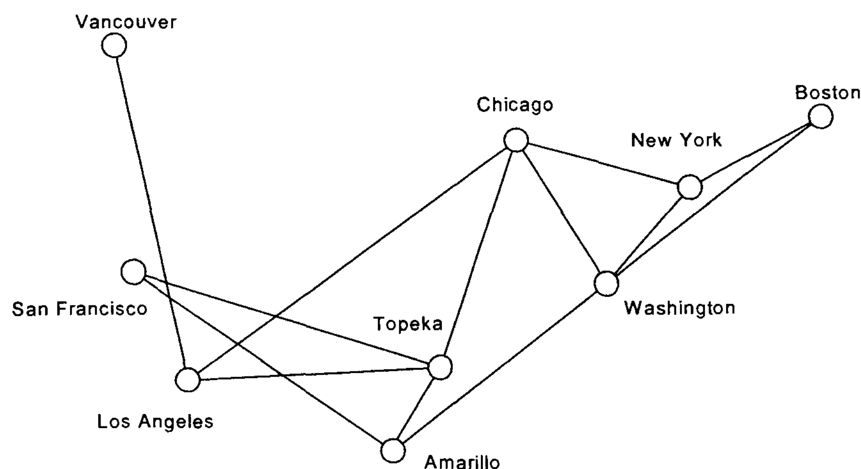

Figure 3.9 – Simple 9-node Network

Using the network presented in Figure 3.9, a typical SLA would be presented to SLAOpt as shown in Figure 3.10.

```
<?xml version = '1.0' ?>
<message type = "addSLA">
    <sla     name = "Rob's SLA"
             source = "Vancouver"
             destination = "New York"
             duration = "10000">
        <qos    capacity = "30000"
                delay = "0.6"
                utility = "30"> </qos>
    </sla>
</message>
```

Figure 3.10 – A Typical SLA for the 9-node Network

If this SLA was admitted, a circuit would be set up starting at the node named "Vancouver" and terminating at the node named "New York". SLAOpt would determine the routing between the internal nodes. For this example, we assume the routing is as follows: "Vancouver" → "Los Angeles" → "Chicago" → "New York". This is sufficient for a simulation, but for a real network with an associated IP addressing scheme this routing is missing some crucial information. First, the only identifier for this SLA is the name "Rob's SLA". How will the ingress router determine which traffic to send down

this circuit? We could use the source and destination nodes to identify this SLA as long as we can match these node names to nodes in our network, and as long as no other SLAs are presented with traffic originating at the ingress node and terminating at the egress node. However, does a customer know the architecture of our network? And should they? A customer might know that there is a node in "Vancouver" and they might even be able to supply an IP address for this node (likewise for the "New York" node). But even with this information, we restrict the number of pathways through the network to a maximum of 9*8 = 72 if we use the ingress/egress node address pairs to identify our pathways. We need a way to extend the architecture of the network. One method would be to include all the hosts connected to our network as sources and sinks. This solves the 72-pathway limitation, but makes our network concept very difficult to implement, as there could potentially be thousands or millions of hosts. Fortunately, clues as to how we can extend and simplify our network concept are available with some analysis of the traffic and the information available to the customer.

Gateways & Subnets

An IP packet arriving at the ingress router in our real network will have both source and destination IP addresses. These addresses are host addresses external to our network – i.e. they identify hosts which reside on subnets connected to our network via some IP pathway. By integrating the concepts of *gateways* and *subnets* into our network model we can then map these source/destination addresses into ingress/egress addresses and thus extend our network into the customer realm without making it unmanageable. A simple example best describes this idea.

Example

Suppose a customer wishes to send traffic through our network which originates at a host in "Vancouver"; this host has IP address 10.4.1.100. The customer wishes to send data to a host in "New York"; this host has IP address 10.4.11.103. The revised SLA that the customer would present to SLACtl would be as in Figure 3.11. The only difference between this revised SLA and the typical SLA presented in Figure 3.10 is that we now specify IP addresses for the source and destination hosts. These IP addresses are readily available to the customer.

```
<?xml version = '1.0' ?>
<message type = "addSLA">
    <sla    name = "Rob's SLA"
            source = 10.4.1.100
            destination = 10.4.11.103
            duration = "10000">
        <qos    capacity = "30000"
                delay = "0.6"
                utility = "30"> </qos>
    </sla>
</message>
```

Figure 3.11 – A Revised SLA for the 9-node Network

Next, how do we setup a circuit through our network? We need to map these IP addresses into node addresses in our network. For this example, we use the node names in the simple 9-node network. Assuming the source host belongs to subnet 10.4.1.x with its gateway being the "Vancouver" node, then we can know that traffic generated by the host at 10.4.1.100 will be presented to our network at the "Vancouver" node with IP address 10.4.1.1. Likewise, we can determine that the destination address of 10.4.11.103 is accessible from the gateway node "New York" with IP address 10.4.11.1. Now, we can setup routing through our network. Additionally, we can select data packets to traverse the pathway we have allocated between "Vancouver" and "New York" because we know that data packets arriving with a source address of 10.4.1.100 and destination address of 10.4.11.103 refer to traffic associated with "Rob's SLA". Thus, introducing the concepts of gateways and subnets into the simplified network model of SLAOpt extends this simulated model into a representation suitable for a real-world network.

Mapping commands

The second problem we need to address in this interface is how to map commands presented to the simulated network into commands presented to the real data network. The real data network must mirror the pathways and routing structure that SLACtl sets up in its internal network. This problem is addressed by including an object that interfaces the real network to the internal network. There are some issues associated with this, though. The QME manipulates the internal network extensively during its processing of SLA admission / deletion requests. In fact, it de-allocates previously allocated pathways temporarily so that it can determine the optimal allocation. This de-allocation is fine for a simulation, but if we were to strictly mirror the processing of the QME, we would end

up with major disruptions in service in our real network. Every time a customer requested admission would mean that all data traffic would be temporarily suspended – not a reasonable practice.

Separation of Concerns

The third issue that should be addressed in our design of SLACtl has to do with separation of concerns. As we have seen in the early discussion of our analysis, SLAOpt tightly integrates the main modules – in particular, the GUI with the other modules. This tight integration (or lack of separation) will make modification of SLAOpt difficult. As a starting point, the GUI should be separated from the rest of the modules. This will facilitate easier migration of SLAOpt into SLACtl.

We can see that modifying SLAOpt to address these issues, while not a trivial task, is none-the-less achievable. With the analysis phase of the controller complete, we move on to discuss the next part of QoSNET – the Network.

3.3 The Network

In order to integrate SLACtl into an IP network, we needed an operational IP network – our prototype network – QoSNET. This network would need to provide fixed-path routing, so we chose backbone routers with *Multiprotocol Label Switching* (MPLS) support. With the help of our partners at NewMIC and Nortel Networks, we acquired three Passport 7440 routers and began to build the network.

3.3.1 Implementation: A Brief Chronology

A lot of effort went into building QoSNET – the network. Starting in April 2001, with a purchase order for three *Passport 7440 routers* (PP7K); delivery of these routers in June 2001; purchase of Preside *Multiservice Data Manager* (MDM) in December 2001; dedication of a Nortel Networks support engineer in March 2002; and ending in July 2002 with a fully functioning MPLS-enabled IP network. After this, scripts were written to automate the interfaces to SLACtl, and testing began.

3.3.2 Physical Architecture

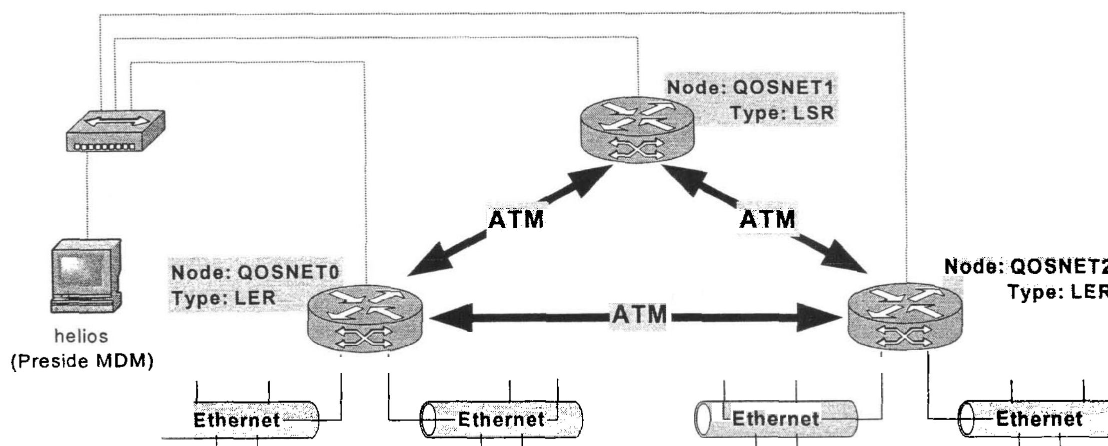

Figure 3.12 – QoSNET Physical Architecture

Figure 3.12 depicts the physical architecture of QoSNET. Two of the PP7Ks were designated as *Label Edge Routers* (LER) – QoSNET0, QoSNET2 – and one as a *Label Switched Router* (LSR) – QoSNET1. We configured the PP7Ks with *CP2 control processors* (CP) and *2-port OC-3 ATM IP function processors* (ATM IP FP). Additionally, we added a *2-port Ethernet 100BASE-TX function processor* (Eth IP FP) to each of the LERs. The ATM IP FPs form the backbone interconnections of our prototype network – (MPLS-labeled) IP traffic is carried inside ATM cells over SONET. The Eth IP FPs in the LERs enable us to inject IP traffic (wrapped in Ethernet frames) into the network at the ingress node and extract it at the egress node (loading it back into Ethernet frames). ATM over SONET interfaces interconnect the three nodes and 100BASE-TX Ethernet interfaces connect up external Ethernet networks to QoSNET. The third interconnect is a separate administration network which allows a Sun SPARC Ultra 5 to configure the individual QoSNET nodes through the appropriate CP's OAM Ethernet port (10BASE-T interface).

3.3.3 Commissioning / Configuring

Since the Passport routers were shipped with an old version of the Nortel-proprietary operating system and related firmware (one which does not support either the newer-technology ATM IP FPs or MPLS), the first step was to acquire and load a new version. The method employed in these routers is to use a customer-configured private ftp site called a *Software Distribution Site* (SDS).

The latest release of the software at the time (pcr 2.3GA – released June 2001) was acquired, an SDS was set up, and new software downloaded to one of the PP7Ks. However, the ATM IP FPs use a PowerPC processor and the CPs use an Intel i960 processor. Thus the software needed to be not only the correct modules (and their associated dependencies), but also designated for the appropriate processor type. A crash course in commissioning [30] and configuring [31] was now required.

We were not far into the commissioning process before we realized that we needed Preside MDM to help with the task. After acquiring it, we set about to learn this new tool [37]. Preside MDM is a workstation-based network management system that permits maintenance and monitoring of a complete network from a central or a decentralized network control center. Preside MDM has a full suite of applications and external interfaces to manage a number of different devices.

The initial stages in the commissioning process were very frustrating as we were faced with this unfamiliar product without much assistance, and the documentation was overwhelming [33]. After some ear bending, we managed to connect up with a Passport-certified support engineer who came on-site to help with the commissioning tasks and to train us along the way.

By this time, a new software release was available, which we soon discovered we needed. MPLS support in the Passport routers was only 'bleeding-edge' technology; it was being added incrementally to the software base during this time. So we acquired the latest release and started upgrading the PP7Ks again. Even with this latest release (pcr 3.1 GA – released November 2001), MPLS support on these backbone routers was still in its infancy and had limited capabilities. However, as we will show in the following discussion, we were able to bypass these limitations somewhat, and continue with our research into the feasibility of using SLACtl to control an MPLS-enabled IP network.

With the benefit of our previous experience of upgrading software, we quickly upgraded to the new software without any problems. QOSNET0 was finally at stage 0 and ready to be commissioned. Base configuration was performed and the PP7K recognized all of its components – green lights all around. A powerful feature of the PP7K is its ability to hot swap processor cards, including CPs. This feature automatically synchronizes the hard

drive on a spare CP with that of the main CP. We used this synchronization feature to automatically copy the software from the CP in QOSNET0 to the CPs destined for QOSNET1 and QOSNET2. With the latest software base on each of the PP7Ks, we connected up to Preside MDM for the rest of the commissioning process. The first step was to setup ATM Trunks [36] among the nodes so they could send ATM traffic over their SONET interfaces. After this, we configured the 100BASE-TX interfaces [34] so the two LERs could accept external Ethernet frames. With the commissioning completed to this stage, we then added IP [32] on top of the ATM configuration. We reached a major milestone, with the first successful ping command at this stage.

3.3.4 IP Architecture

As part of the commissioning process, we configured four groupings of IP subnets in QoSNET. These four groupings comprise three distinct planes – Administration, Data, and Control.

Administration Plane

The Administration Plane consists of 10BASE-T interconnections between each of the PP7K's CPs and Preside MDM running on the Sun Ultra 5. The Administration Plane is depicted in Figure 3.13. The subnet 10.4.0.x/24 groups the IP addresses used to pass configuration and administration data to and from the PP7K backbone routers.

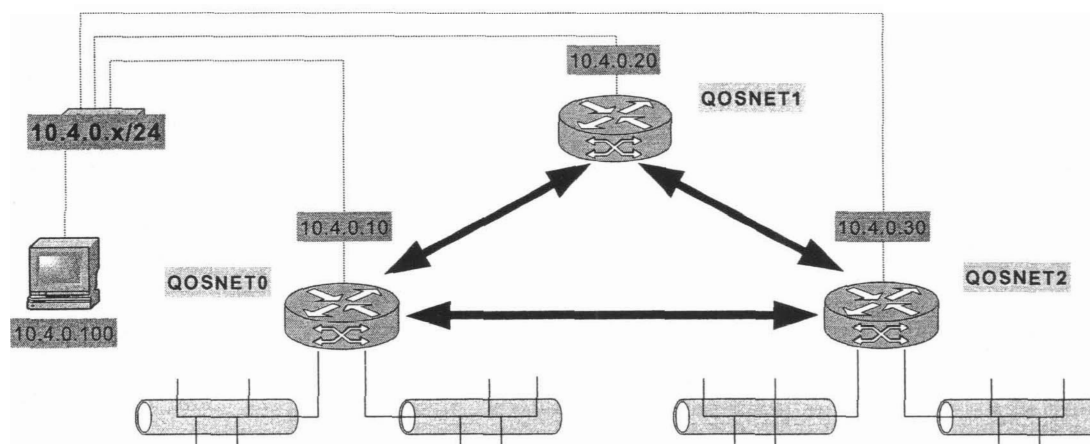

Figure 3.13 – QoSNET Administration Plane

Data Plane

The Data Plane is comprised of the ATM Data subplane and the Ethernet Data subplane. The ATM Data Subplane is depicted in Figure 3.14 and consists of OC-3 SONET interfaces between each of the PP7Ks over which ATM cells flow. The subnets 10.5.1.x/24, 10.5.2.x/24, 10.5.3.x/24 group the IP addresses used to identify OC-3 hardware ports on each of the PP7K backbone routers.

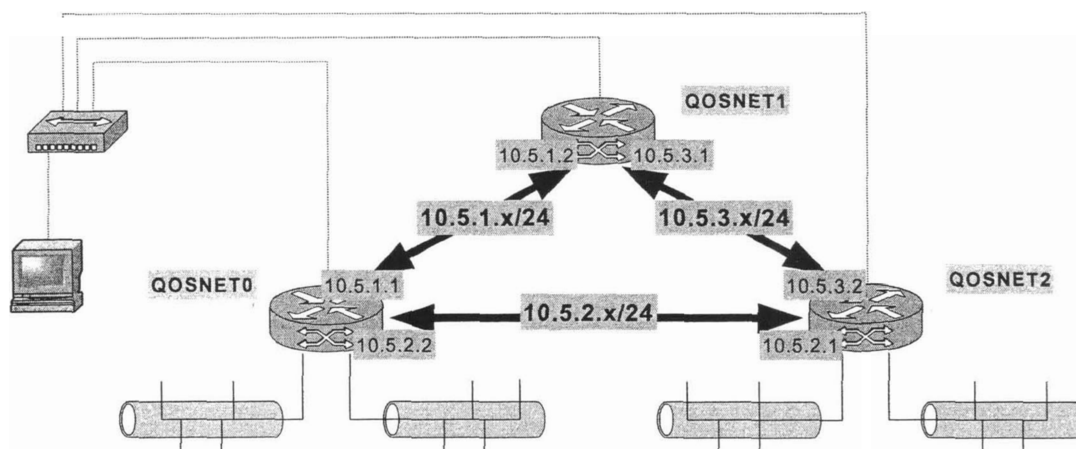

Figure 3.14 – QoSNET ATM Data Subplane

The Ethernet Data Subplane is depicted in Figure 3.15 and consists of 100BASE-TX interfaces between the two LERs and their externally connected Ethernet subnets. These subnets form the ingress and egress points of QoSNET. The subnets 10.4.1.x/24, 10.4.2.x/24, 10.4.11.x/24, 10.4.12.x/24 group the IP addresses used to identify Ethernet hardware ports on the two edge PP7Ks.

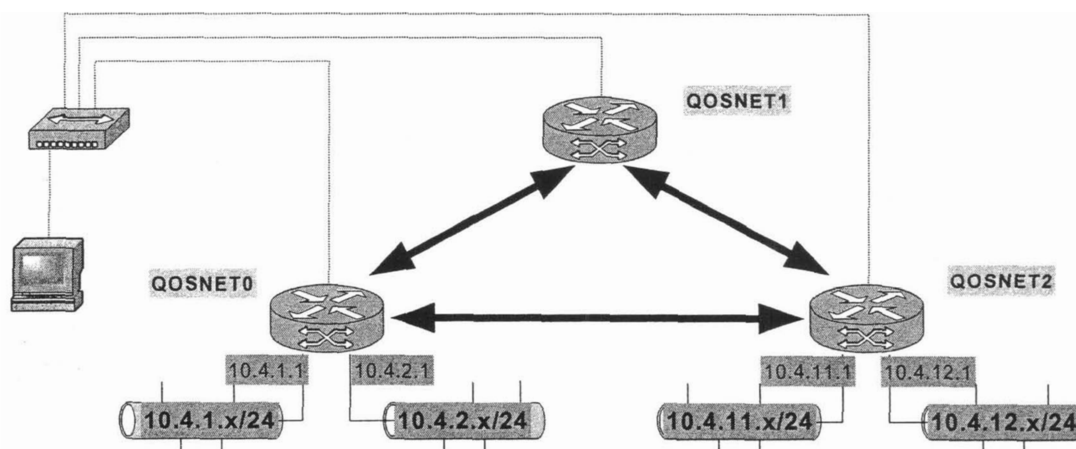

Figure 3.15 – QoSNET Ethernet Data Subplane

Control Plane

The Control Plane consists of abstract IP addresses used to identify the virtual routers in each of the PP7Ks. The virtual router IP addresses are used in the various routing protocols enabled on the PP7K; protocols such as *Border Gateway Protocol* (BGP) [42], *Open Shortest-Path First* (OSPF) [28], and the MPLS *Label Distribution Protocol* (LDP) [4]. The Control Plane is depicted in Figure 3.16. The router ids 10.5.10.1, 10.5.20.1, 10.5.30.1 correspond to QOSNET0, QOSNET1, and QOSNET2, respectively.

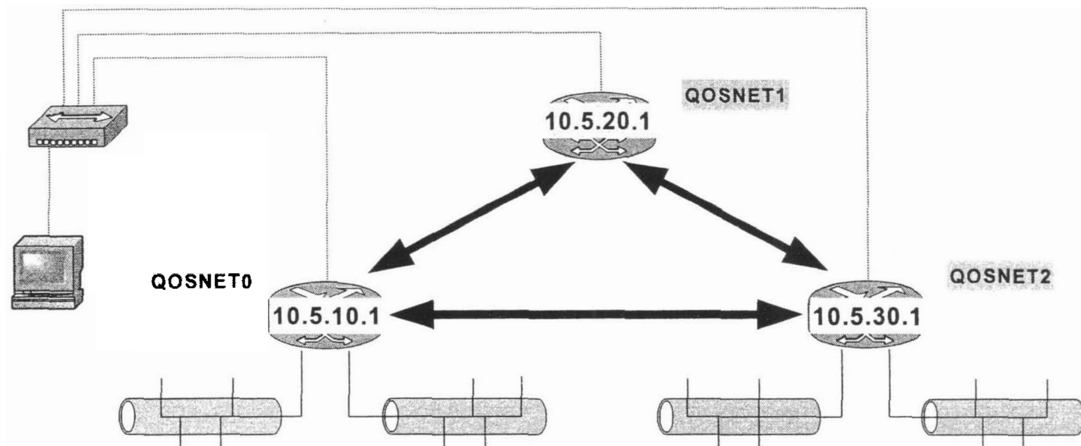

Figure 3.16 – QoSNET Control Plane

It is important to note that these IP addresses are entirely arbitrary as QoSNET was designed to be stand-alone for research purposes only.

A detailed configuration script for QOSNET0 is provided in Appendix A. The configuration for QOSNET1 and QOSNET2 are similar with the exception of different names and IP addresses.

With IP configured and running on QoSNET, we connected up the components of our testbed and for the first time had IP traffic flowing from a synthetic IP (UDP) traffic generator (Java-based) on one workstation (qosnet01), through Ethernet frames to the ingress router (QOSNET0), through ATM cells on SONET frames to the egress router (QOSNET2), back into Ethernet frames, and appearing at a packet sniffer on a different workstation (qosnet03). This UDP traffic test is illustrated in Figure 3.17.

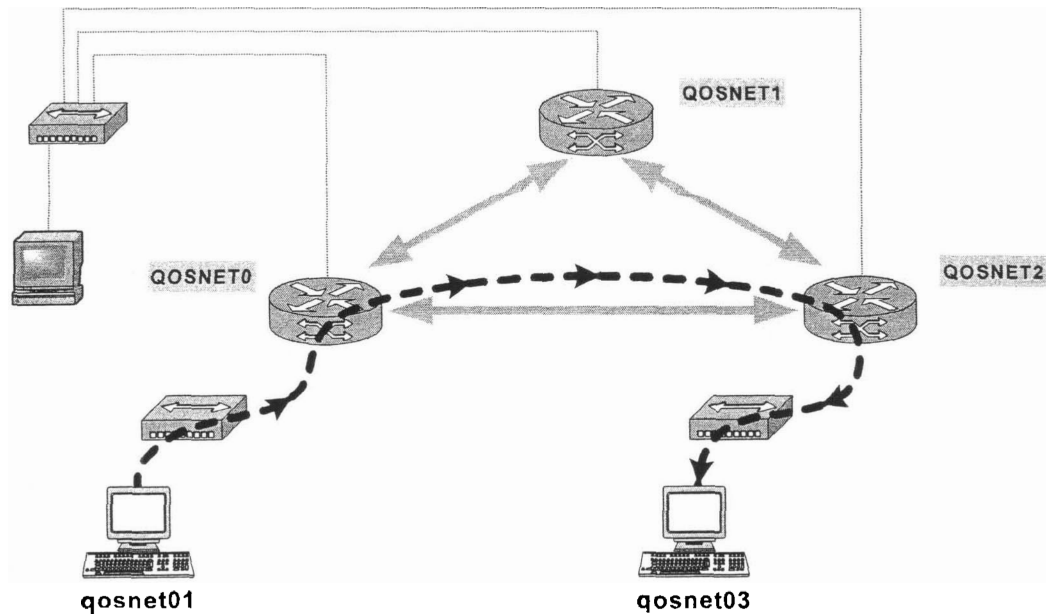

Figure 3.17 – UDP Traffic Test

The UDP traffic test was very simple and only proved that we had an operational *best-effort* datagram network. We could get datagrams through, but there was NO guarantee that prescribed service levels could be provided, nor did we have fixed-path routing; for this we needed MPLS.

3.3.5 MPLS Architecture

Our next task was to study the Nortel MPLS documentation [35] and determine how we could use the MPLS functionality provided.

At this stage in its maturity, MPLS on the PP7K is solely supplied as a *Traffic Engineering* (TE) [12] tool. That is, it is intended as a way of steering heavy traffic onto relatively unused links and away from overly-used links. The idea is that if a *Label Switched Path* (LSP) is available, then traffic will follow such a path. Depending on how you interpret the specification of MPLS, in particular how you are approaching an MPLS network, this can have two different meanings. From the perspective of a datagram network, an LSP is simply a set of next hop information which routers use to direct traffic through the network – there is no physical circuit set up. However, from a circuit-switched network point of view, an LSP would also entail setting up an actual circuit or pipe along which to send traffic.

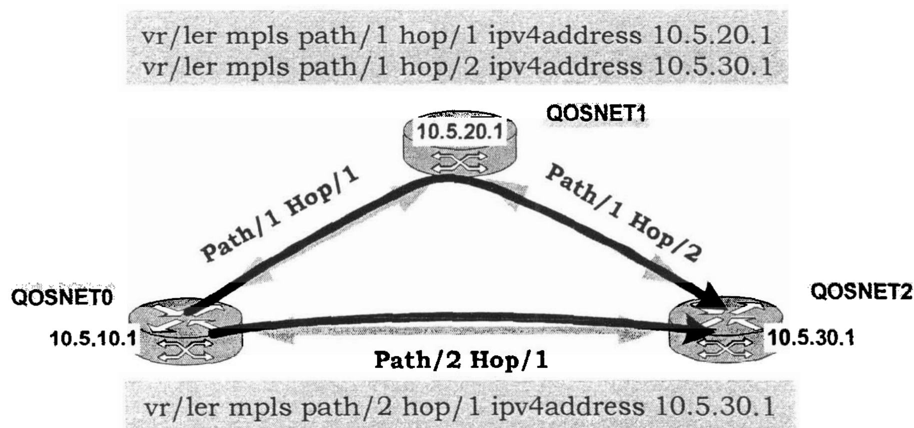

Figure 3.18 – MPLS Paths

The MPLS approach employed in the PP7K is a combination of these two perspectives. A Path contains strictly next hop route information specifying a list of LSR/LER router ids that the datagrams will follow. Figure 3.18 illustrates the setup of two LSPs in QoSNET from QOSNET0 → QOSNET2 – Path/1 has two Hops: 10.5.20.1, 10.5.30.1 – Path/2 has one Hop: 10.5.30.1 directly. An example Path provisioning script is given in Appendix B.

Label-Switched Path Groups

In the PP7K, there are additional entities that support the circuit-switched perspective. The one used by MPLS, called a *Label Switched Path Group* (LSPG), contains up to four LSPs which are mapped directly to a set of ATM VPCIs; thus enabling the allocation, reservation, and commitment of network resources along the LSP and forming a circuit or pipe.

By definition, LSPs contained within an LSPG will follow the pre-defined Path assigned to the LSP, and will terminate at the egress of the LSPG. It is then the job of the egress router to ensure the datagrams are correctly routed onto the external subnet. Figure 3.19 demonstrates the concept of an LSPG and Figure 3.20 shows the addition of two LSPs. An LSP is contained entirely within the backbone, but an external subnet (in this case containing a single IP address) can be specified as the final destination for an LSPG.

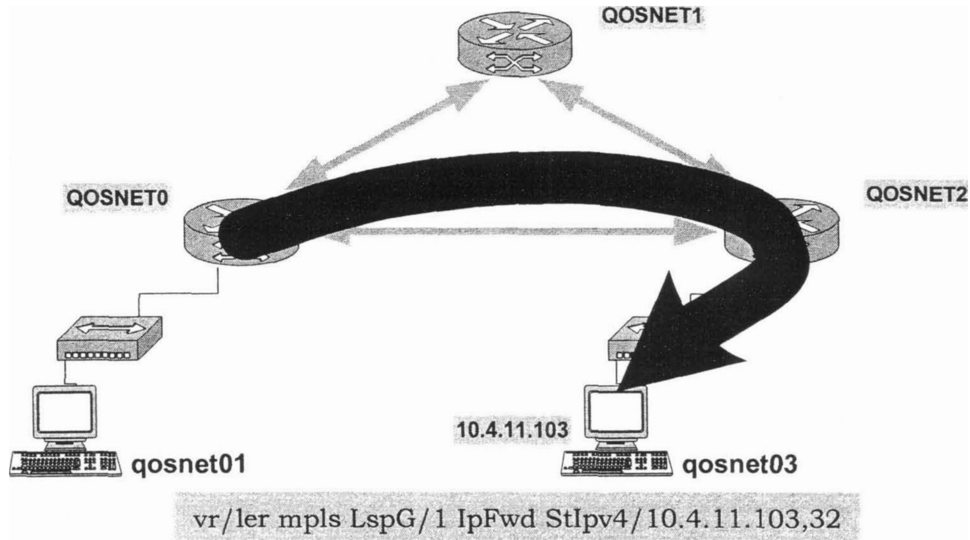

Figure 3.19 – LSPG Setup

In Figure 3.19 the LSPG illustrated will actually terminate at the egress LER (QOSNET3), but the ingress LER (QOSNET1) uses the LSPG destination IP address to determine which datagrams are routed through this LSPG (ingress packets with a destination IP of 10.4.11.103) and which will be routed a different way. LSPG/1 is shown as a solid arrow terminating at qosnet03 because 10.4.11.103 is the specified destination of the LSPG.

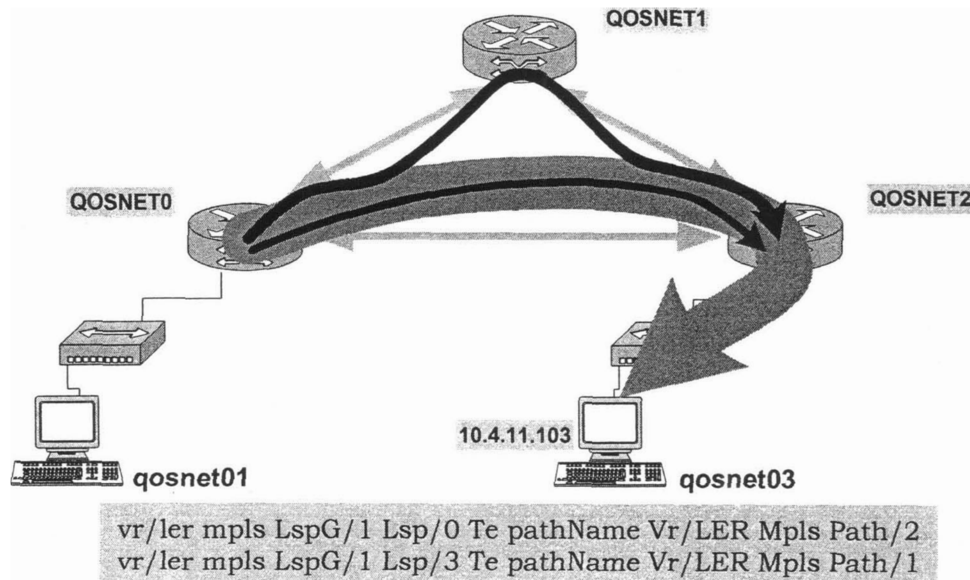

Figure 3.20 – LSPs in an LSPG

LSPG/1 in Figure 3.20 contains two LSPs (Lsp/0 and Lsp/3). These LSPs can have different Paths as well as different QoS constraints – in particular, different bandwidth parameters, such as *peak data rate* (PDR), *peak burst size* (PBS), *committed data rate* (CDR), and *committed burst size* (CBS) – depending on the requested data flow characteristics. However, each of the LSPs must ultimately terminate at the egress LER corresponding to the destination IP address.

3.3.6 Packet Classification

Based on the architecture thus far discussed, we can expect that data traffic with destination 10.4.11.103 injected at ingress LER QOSNET1 will traverse one of the two LSPs specified. But how do we determine which LSP to take? Those packets with a destination match to the LSPG destination and marked with a *MPLS Service Category* (MSC) corresponding to the MSC of the LSP will traverse that particular LSP. The MSC is determined, in turn, by the IP *Class of Service* (CoS) ↔ MSC mapping. So the next step is to discuss how an IP CoS is assigned to a particular data packet.

CoS Policies

In the Passport, the IP CoS can be assigned based on various attributes, such as the incoming media link, the IP ToS field (or DSCP), the source or destination IP address, the transport protocol (TCP or UDP), or the port number. Since our SLAs specify the source and destination IP addresses as the endpoints for the requested path, we use IP addresses to assign the IP CoS. And since the destination IP address is already used in the LSPG, we only need to consider the source IP address in assigning an appropriate CoS. There is much detail in setting up packet classification schemes using Passport's CoS Policy Groups and specifying packet filtering mechanisms, however much of that detail is irrelevant for our discussion. Suffice it to say that we can setup a CoS policy such that a data packet originating at the source IP address is marked with a CoS that matches the level of QoS desired – 3-gold, 2-silver, 1-bronze, 0-default. This marked packet then will be mapped by the ingress router to the corresponding MSC, and thus will traverse the appropriate LSP.

With the combination of LSPGs and IP packet classification, we now have support for fixed-path routing through QoSNET. Furthermore, since the PP7Ks use MPLS over ATM we also benefit from traffic policing – the explicit path that we set up for our data traffic uses provisioned circuits that are monitored and policed by the ATM subplane.

The commands we used to set up LSPs and LSPGs (along with CoS policy groups) were molded into a parameterized provisioning script for SLACtl to build its desired end-to-end paths through QoSNET – this forms the internal interface between SLACtl and the network. The provisioning script is provided in Appendix C.

3.4 Summary

To summarize, we have analyzed SLAOpt (the basis for SLACtl) and have determined that with some non-trivial modifications we can integrate it into SLACtl. Further, we have commissioned the network for QoSNET and written scripts for setting up explicit end-to-end paths necessary for our work. At this point we have all the pieces; all we have to do is integrate them.

3.4.1 Lessons Learned

One thing became apparent at this juncture in our work: we were forcing QoSNET into an area that was not supported by the MPLS implementation or the routers themselves. MPLS on the PP7Ks was designed with the view of using it for traffic engineering – manually setting up a few large pipes at one particular time and every so often after that. We needed to transform this concept of relatively static large-pipe provisioning for carriage of many diverse traffic flows into highly dynamic small-pipes locked to a single data flow. As we were to discover during testing, this was something that the hardware itself was not ready to support. As we set up more and more paths through QoSNET, the routers became unstable during the provisioning process – rebooting and restoring previous configurations.

To be fair, these backbone routers were designed for manual provisioning; with manual provisioning there is an upper bound on the number of paths provisioned at one time and a lower bound on the time interval for and between provisioning requests. That is, the

routers were not designed for rapid automated provisioning of so many paths; our automation kept the routers in their provisioning mode and hardly ever allowed them to enter their operating mode, which is contrary to how they were designed to operate.

3.4.2 Next Step

Our ultimate view of a customer requesting an end-to-end path through QoSNET for a particular data flow was reverting back into a dream. We could build a tool that would *help* automate the SLA admission process, but it would be more of a traffic-engineering tool. Not a bad idea, but not what we had envisioned at the start of our research. To realize the full potential of our work, we would need support from the routers at a very low level. We would need to influence how and when resources are allocated. We would need to change the design assumption that these routers use manual provisioning. And we would need to influence how QoS is handled from the hardware level on up.

It was during this re-thinking period that I became involved with Syscor R&D as a QoS consultant to their Hardware team. The technology they were building was in the early hardware / protocol design stage and they needed some direction on how to support QoS. Over the course of the last three years, I have been a part of the research and development effort at both the hardware and software level. This has led to our second solution, IPFSNET, which is discussed in the next chapter.

Chapter 4
IPFSNET

In this chapter we describe IPFSNET, a research and development environment built to investigate IPFS technology and its related protocols. Unlike QoSNET, which had a specific physical configuration, IPFSNET takes on many distinct configurations, depending on what part of the technology we are investigating. The hardware and software modules of IPFSNET were used to prove that the concept of IPFS is viable and will form the building blocks for a commercial routing / switching product targeted for MANs. Our work has now passed the proof-of-concept stage and we have implemented an operational development node called the G50. The hardware and software implementation including the FPGA logic, node configuration options, and datagram segmentation, encapsulation, and transmission is described in the latter part of this chapter, followed by a discussion on how QoS is supported in this technology. But first we explain the IPFS technology and its related protocols.

4.1 IPFS*

IP Frame Switching (IPFS) is a new communication technology based on the *Encapsulated Routing Protocol* (ERP) to transport IP packets over SONET [9] using

* **IPFS, ERP,** and **HRN** are presently under development at **Syscor R&D** and are protected under patents and Non-Disclosure Agreements (NDA). The following information is **strictly confidential.**

proprietary *Frame-Forwarding Logic* (FFL). The main design criterion for IPFS is to provide low-cost, high-bandwidth, guaranteed-QoS IP connectivity to a carrier's business customers. This technology interfaces with the existing SONET infrastructure, adding *layer-2* (L2) switching for IP datagrams. Delays that are usually incurred at network routers from buffering, waiting on large datagrams, and *layer-3* (L3) routing are minimized with IPFS. Using an Ethernet-like 6-byte *Medium Access Control* (MAC) address, the IPFS nodes are configured into a *Hierarchical Ring Network* (HRN) by dynamically assigned location-dependent addresses. The HRN structure allows for the implementation of a simple decision-making mechanism for the FFL. Using a HRN structure combined with the FFL, IPFS can rapidly switch IP (or other encapsulated payloads) through a HRN using only simple hardware circuitry, minimizing latency in the process. Also, IPFS uses a fixed-size frame of a reasonable-length so that QoS constraints such as bandwidth and latency can be absolutely guaranteed throughout the HRN without complicated packet interruption and preemption schemes.

4.1.1 HRN

An example of the *Hierarchical Ring Network* (HRN) structure is given in Figure 4.1. A node connecting a sub-ring to a higher-level ring (super-ring) is called a master and all the other nodes on the sub-ring are called slaves.

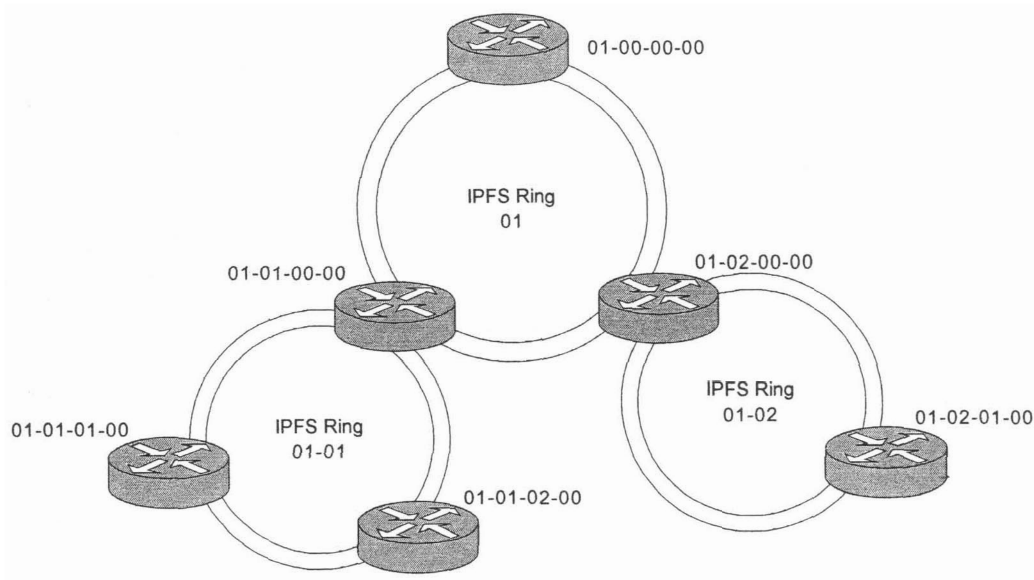

Figure 4.1 – HRN Topology

At the top of the hierarchy is the root node, which is the master of the highest level ring. A node's HRN address uniquely determines where the node resides in the HRN. For example, the root node (master) at the top of the HRN has address 01-00-00-00. HRN addresses are written using a hex format with bytes separated by a '-'. The structure of the IPFS HRN address is shown in Figure 4.2 – the first two bytes are the Area Code and the last four bytes are the Ring | Node address.

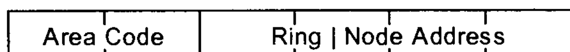

Figure 4.2 – HRN Address

For our discussions, we only consider the Ring | Node address with each byte denoting a new level of the hierarchy, although other schemes are also possible. Using this definition, our example HRNs can have up to four levels of rings. The Ring | Node address is comprised of the ringID concatenated with the nodeID then 00-filled.

The root node in our example has a ringID of 01 and a nodeID of 00 – a master node always has nodeID 00. Thus, the HRN address is 01-00-00-00 (zero-filled). The other nodes on ring 01 are node 01-01-00-00 which is a slave node on ring 01, but master to the 01-01 sub-ring, and node 01-02-00-00 which is master to the 01-02 sub-ring. Each ring has one master node and many slaves – each slave node on a ring may be a master node on a sub-ring. The other nodes in our example HRN are nodes 01-01-01-00 and 01-01-02-00 on sub-ring 01-01 and node 01-02-01-00 on sub-ring 01-02.

By arranging the nodes in this manner, IPFS can use simple hardware functions to determine what to do with a particular frame – pass it to the next node on the ring, drop it to other logic in the node, or copy it to a different ring. These functions use a small *Content Addressable Memory* (CAM) table to parse each address byte and set simple binary (0 / 1) signals that control the FFL and direct the frame to the appropriate *First-In First-Out* (FIFO) buffer. The speed of this FFL enables IPFS to use cut-through forwarding – a technique where a frame's transmission is started well before it has been entirely received. The FFL is entirely implemented in hardware using a *Field Programmable Gate Array* (FPGA).

4.1.2 Node Configurations

The G50 can operate in various configurations combining attributes such as Master/Slave, clock Source/Sink, Simple/Switch mode, and Protection/Non-Protection. The configurations for a typical HRN are illustrated in Figure 4.3.

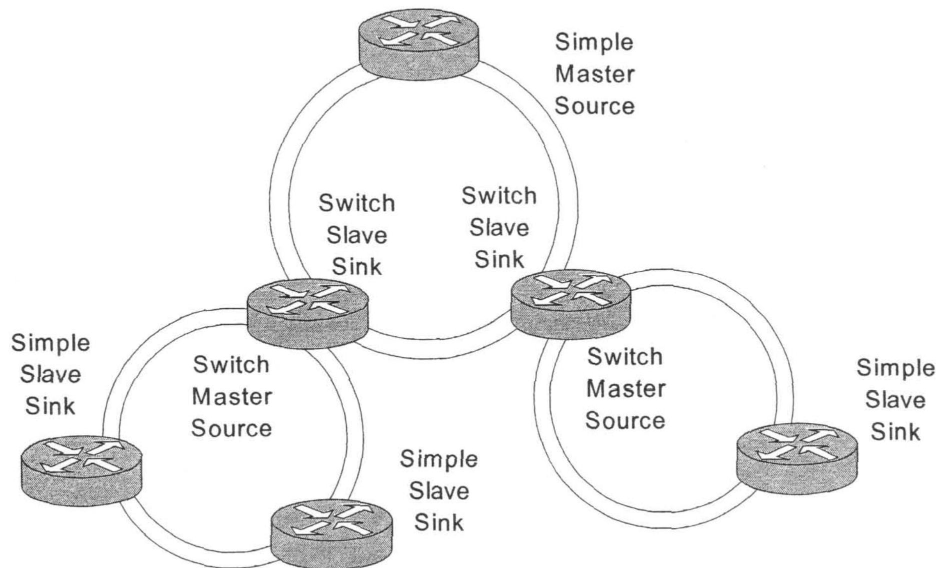

Figure 4.3 – G50 Node Configurations

Some nodes operate in one configuration on one ring and a different configuration on a second ring – depending on the role that they play in the HRN. For example, there is only one master node per ring and only one clock source; a switch node will be a master node on its sub-ring and a slave on its super-ring. Generally, a master node supplies the clocking for the ring, however, in some instances a non-master node or even an external device could supply the clocking.

4.1.3 ERP

The container used for transceiving data over the optical interface is 258 bytes long and contains the IPFS Header (2-bytes) and the *Encapsulated Routing Protocol* (ERP) frame (256-bytes). This container is called an IPFS frame and is illustrated in Figure 4.4. The ERP frame includes the ERP header and payload areas.

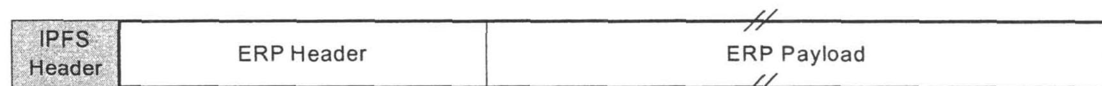

Figure 4.4 – IPFS Frame Layout

Several IPFS frames are aggregated together to form one optical frame to be transmitted. The number of IPFS frames per optical frame is dependent on the optical line speed; for STS-3, nine IPFS frames are carried in the *Synchronous Payload Envelope* (SPE) of the SONET OC-3c frame. This aggregation is shown in Figure 4.5 where each IPFS frame takes up a single row of the SONET SPE. The first column of the SPE is used by the *Path Overhead* (POH); the next 258 columns contain the aggregated IPFS frames, leaving two columns for future expansion. Only the SPE has been shown in Figure 4.5; nine columns of SONET overhead precede the SPE in an OC-3c optical frame.

POH	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	
	IPFS	ERP header	ERP Payload	

Figure 4.5 – IPFS Frame Aggregation in an STS-3 SPE

This row-aggregation of IPFS frames is orthogonal to the usual byte interleaving (or column-format) used for transceiving payloads in SONET. However, by using rows instead of columns, IPFS benefits from the efficiency of the physical layer of SONET that serializes optical frames on a row-by-row basis.

4.1.4 Frame-Switching

IPFS uses the HRN address in the ERP frame header to switch frames at layer 2. Pseudo code showing how the FFL switches frames through a node is given in Figure 4.6.

Broadcast frames are switched to all FIFOs to be transmitted on the super-ring and the sub-ring (Copy and Pass FIFOs) and to be processed by other logic in the node (Drop FIFO). Unicast frames are switched only to the appropriate FIFO depending on whether there is a match on the address or ring. Local broadcast frames are switched only to the Pass and Drop FIFOs, not to the Copy FIFO.

```
frameAddress <= ERP frame[2:7];          // address in bytes 2->7
frameRing <= ERP frame[ring mask];       // ring in bytes designated by mask
mode <= ERP frame[mode mask];            // mode in bits 25->26

if ((frameAddress == thisAddress) || (mode == broadcast) || (mode == localbroadcast))
    DropFIFO <= ERP frame;
    if (mode == unicast)
        return;

if ((frameRing == thisRing) || (mode == broadcast) || (mode == localbroadcast))
    PassFIFO <= ERP frame;
    if (mode == unicast)
        return;

if (((frameRing != thisRing) && (mode != localbroadcast)) || (mode == broadcast))
    CopyFIFO <= ERP frame;
    if (mode == unicast)
        return;
```

Figure 4.6 – Frame-Switching Pseudo Code

Example

Suppose we have an HRN as presented in Figure 4.7, and we have an ERP frame with some payload that we wish to send from node A (01-01-02-00) to node D (01-02-01-00). This frame would be a unicast frame (mode = C), with destination address 01-02-01-00. Using an arbitrary Area Code of 00-FA and integrating the mode bits results in HRN address 00-FA-C1-02-01-00. This is the full HRN address appearing in the ERP header – note that the mode bits occupy the first two bits of the Ring | Node Address and the Area Code is prepended to the Ring | Node Address.

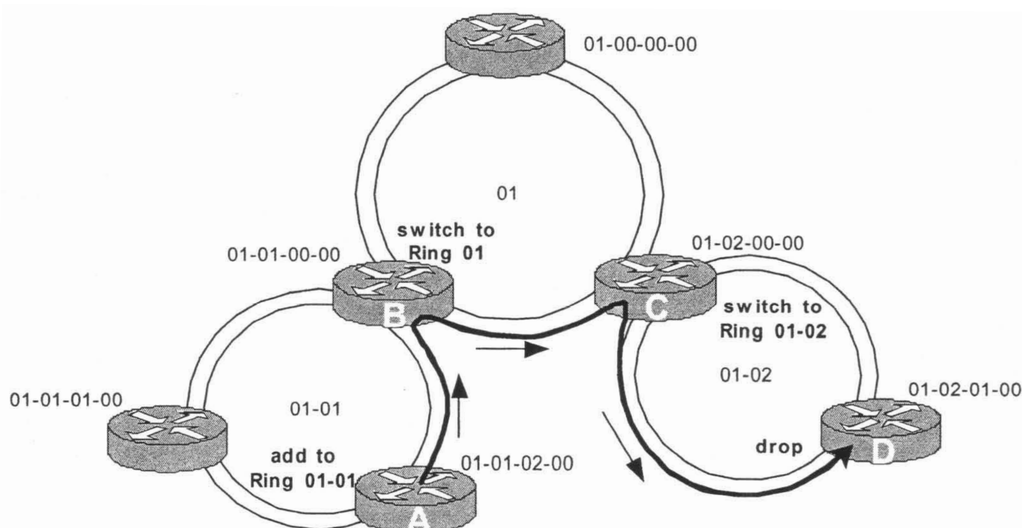

Figure 4.7 – Example IPFS HRN for Unicast Frame-Switching

The ERP frame would be similar to Figure 4.8; for simplicity, only the relevant header fields have been specified.

Figure 4.8 – Example ERP Frame with Prepended IPFS Header

Our example ERP frame will be aggregated with eight other frames at node A (01-01-02-00). The resulting optical frame will then be transmitted to node B (01-01-00-00) – assuming the optical path is in the direction of the arrows). As the optical frame is received, node B will decode the ERP headers in the SPE, placing this ERP frame in the Copy FIFO to be sent on its upper ring (01). When the frame is at the front of the Copy FIFO, it will be aggregated with other frames and transmitted on the upper ring. In turn, node C (01-02-00-00) will receive the optical frame, decode the ERP headers, and place this ERP frame into its Copy FIFO to be sent on the lower ring (01-02). Our frame will then travel on the lower ring to node D (01-02-01-00), its destination. When the destination node decodes the ERP header, it will place this frame in its Drop FIFO to be processed by other logic. This is how L2-switching is done in IPFS within the SONET infrastructure.

Besides unicast, other possible modes are multicast, global broadcast, and local broadcast. The latter is used for control frames within a single sub-ring, whereas the other modes are used to carry encapsulated user data.

By transporting the IPFS frames inside the SPE of the SONET optical frame, the IPFS protocol is transparent to other SONET equipment, such as *Add-Drop Multiplexers* (ADM), and can be easily added to an existing SONET network. The existing backbone can then be used to extend the IPFS HRN so that a customer with many locations distributed around the globe can interconnect their branch offices with a common IPFS network.

An important point to note here is that an IPFS HRN is easy to setup – there is no provisioning and configuration is automated; this is crucial to our work as we need a technology that can quickly and dynamically set up end-to-end routing through the network. Referring to our previous discussions concerning the setup, configuration and

provisioning of QoSNET, a traditional SONET network requires highly skilled network technicians to perform these mandatory time-consuming and manual tasks. With IPFS, it's really 'plug-and-play' – just connect up the optical lines (Tx → Rx) and it's ready to go – all IPFS nodes can immediately communicate with each other.

If the HRN needs to be extended through an existing SONET network, the only additional configuration required is to provision a bi-directional STS-3 circuit between the gateway ADMs in the SONET backbone. Then all of the IPFS nodes are able to communicate with each other no matter which partition of the IPFS network they are located in. An example of extending an IPFS HRN through a legacy SONET network is shown in Figure 4.9.

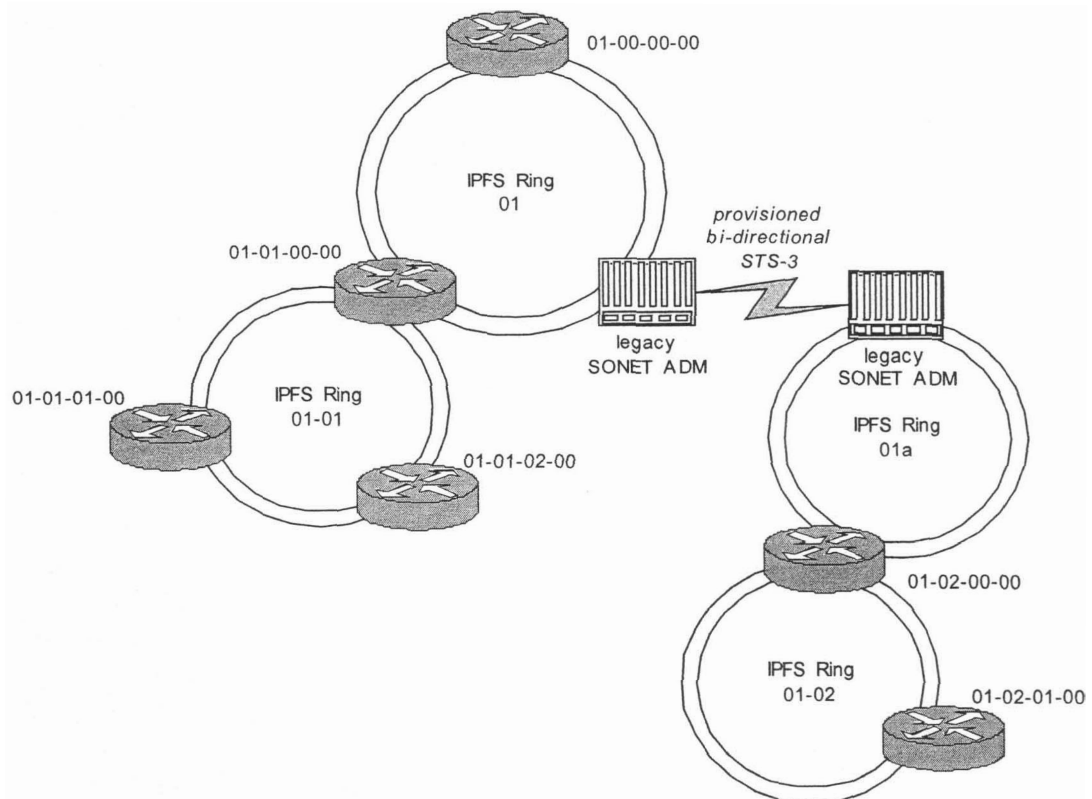

Figure 4.9 – Extending an IPFS HRN Using a Legacy SONET Network

4.2 Node Implementation

IPFS technology has been implemented in hardware and software. A block diagram of the development node, called the G50, is given in Figure 4.10.

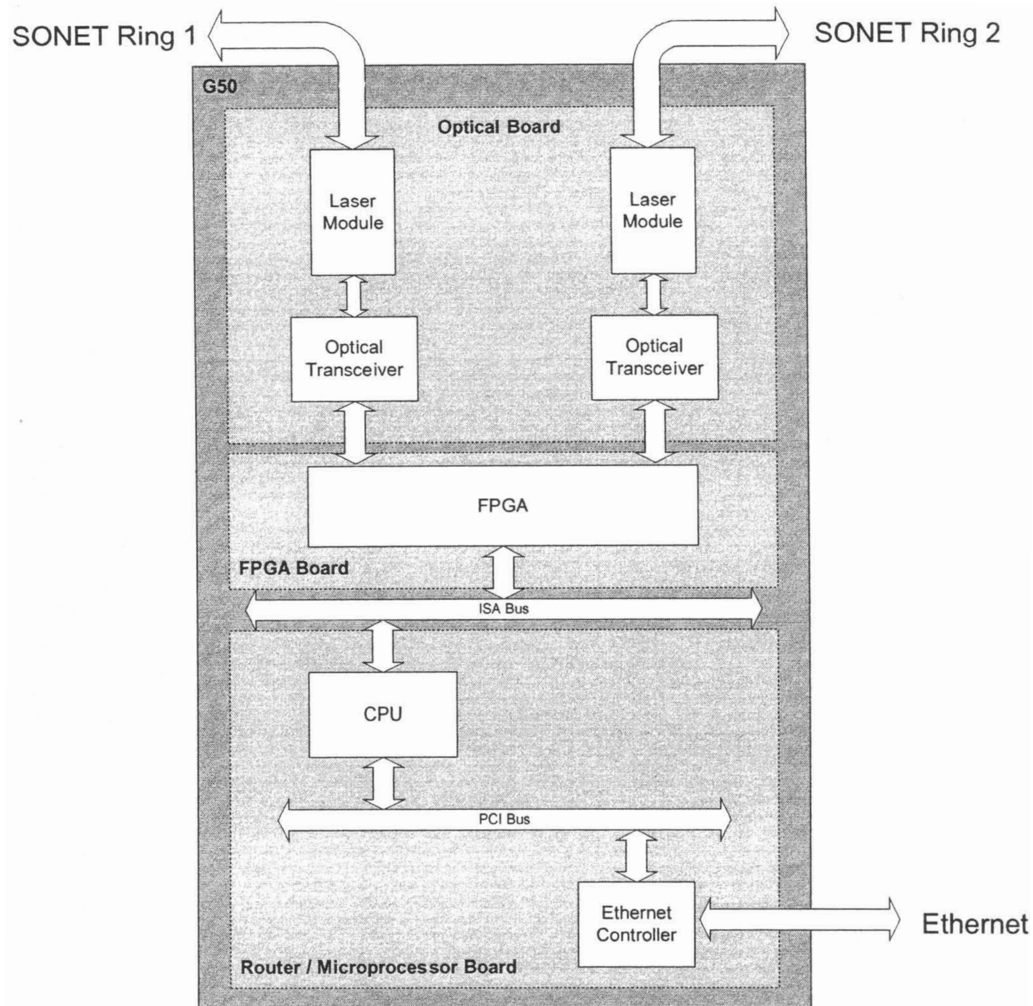

Figure 4.10 – G50 Block Diagram

The G50 consists of three separate but interconnected boards: Router / Microprocessor board, FPGA board, and the Optical board. The FPGA board is connected to the Router / Microprocessor board via an ISA Bus, and to the Optical board via a proprietary Telecom Bus. The optical transceivers interface to SONET rings via laser modules and transcode between parallel data to/from the FPGA and the electrical signals for driving the lasers. An Ethernet Craft interface is provided for controlling the node and a serial RS-232 interface can be used for terminal access. The Ethernet Craft interface is connected to the local PCI Bus via an Ethernet Controller. Likewise, the CPU (MachZ x86) is connected to both the ISA and PCI busses. In the current version of this development node, the Ethernet Craft interface is also used as the Data interface.

The next version (Rev. 2) will separate the Ethernet Craft and Data interfaces. This offloads some of the processor intensive duties of data transmission from the CPU and the slow ISA Bus. The reason for using the ISA Bus in the initial design was to simplify the early stages of development and debugging; our familiarity with the ISA Bus means that we can quickly localize bus errors versus logic or other errors. Rev. 2 connects four Ethernet 10/100 interfaces through a proprietary PCI Bus (a subset of the PCI standard constructed entirely in the FPGA) and one SONET OC-3c interface through a proprietary Telecom Bus to the FPGA. Rev. 2 provides greater bandwidth per interface (~50Mbps).

4.2.1 Data/Frame Flow

Data flow in the G50 is illustrated in Figure 4.11.

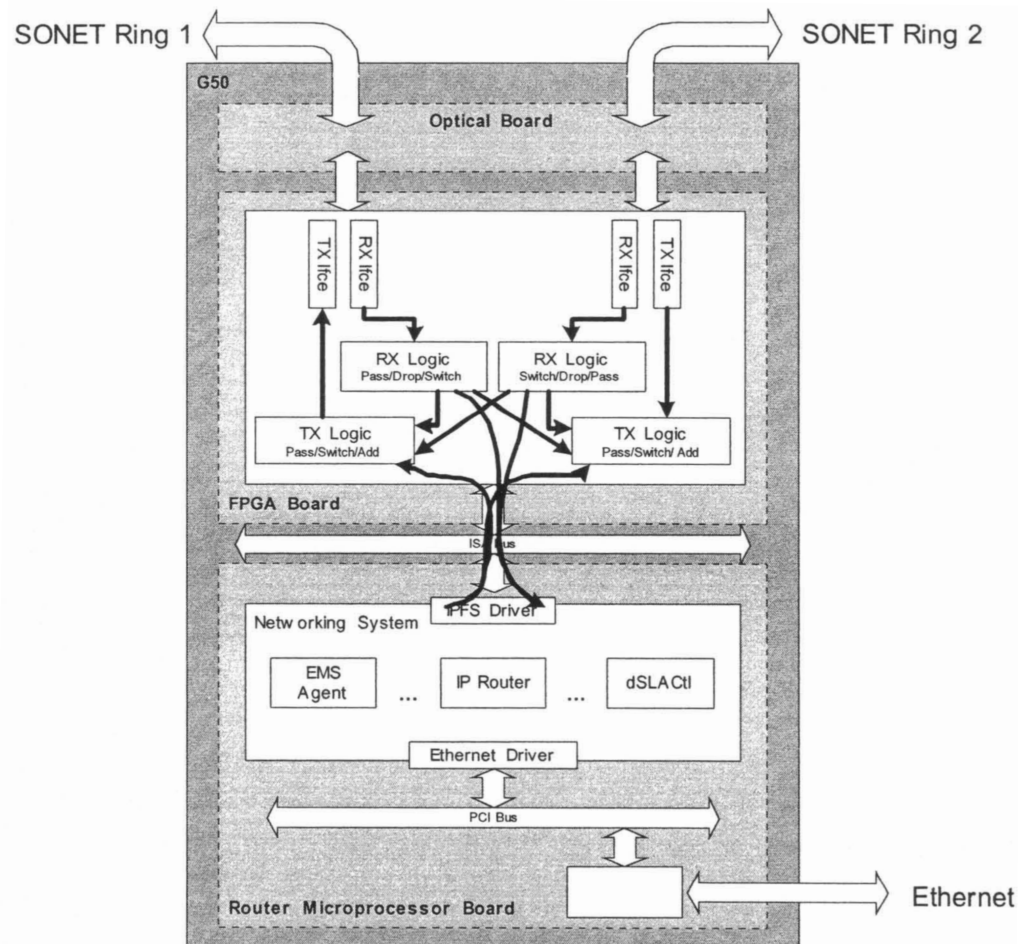

Figure 4.11 – G50 Data Flow

SONET frames are received via the optical transceivers that convert the OC-3c optical signal into a parallel electrical data signal fed into the FPGA Rx module. The RxLogic decodes the IPFS and ERP headers and determines what to do with the associated ERP frame – drop to this node, pass on to the next node on the same ring, or switch to the other ring. This decoding takes the form of an enable signal fed to the appropriate Drop, Pass, or Copy FIFO, in which the ERP frame is then temporarily buffered. Pass frames are immediately ready for transmission via the Tx module, as are Copy frames. Drop frames wait to be transferred over the ISA Bus to the networking system running on the node's CPU.

As for the transmit side, the TxLogic determines which ERP frames to send out in the SONET SPE – either Pass, Copy, or Add frames. Pass frames are those coming from one ring needing to be sent out along the same ring. Copy frames come from the other ring. Add frames are those that this node wishes to send; these come from the networking system running on the node's CPU and are transferred over the ISA Bus to the Add FIFO. Once the appropriate FIFO has been selected via an enable signal, the ERP frame is appended to an IPFS header, which is then aggregated into the SPE of the SONET frame and encoded into a parallel electrical signal to be sent out to the appropriate optical transceiver. The transceiver then converts this SONET frame into an optical signal and transmits over the fiber to the next node on the ring.

Typical IPFS frame flow through a G50 node is shown in Figure 4.12. IPFS frames not destined for the node flow only through the FPGA. Only those frames sourced or sinked by the node (either as host or gateway) are processed by the CPU and transferred over the ISA Bus. This ensures low latency through the node because the in-transit frames are processed solely by the FPGA.

There are four types of flows through a node (applies to either ring):

1. A ⇒ B – Pass frames – transferred through the FPGA, destined for the same ring
2. A ⇒ C – Copy frames – transferred through the FPGA, destined for the next ring
3. A ⇒ D – Drop frames – dropped to the CPU, destined for this node
4. D ⇒ B – Add frames – added from the CPU, originating from this node

In type 3, the G50 operates as a destination host or gateway. The dropped frame may be destined for the node itself or for some other host connected to the node through the

Ethernet interface. In type 4, the G50 operates as a source host or gateway. The frame to be sent over the IPFS HRN may have originated from the node itself or from some host connected to the node through the Ethernet interface.

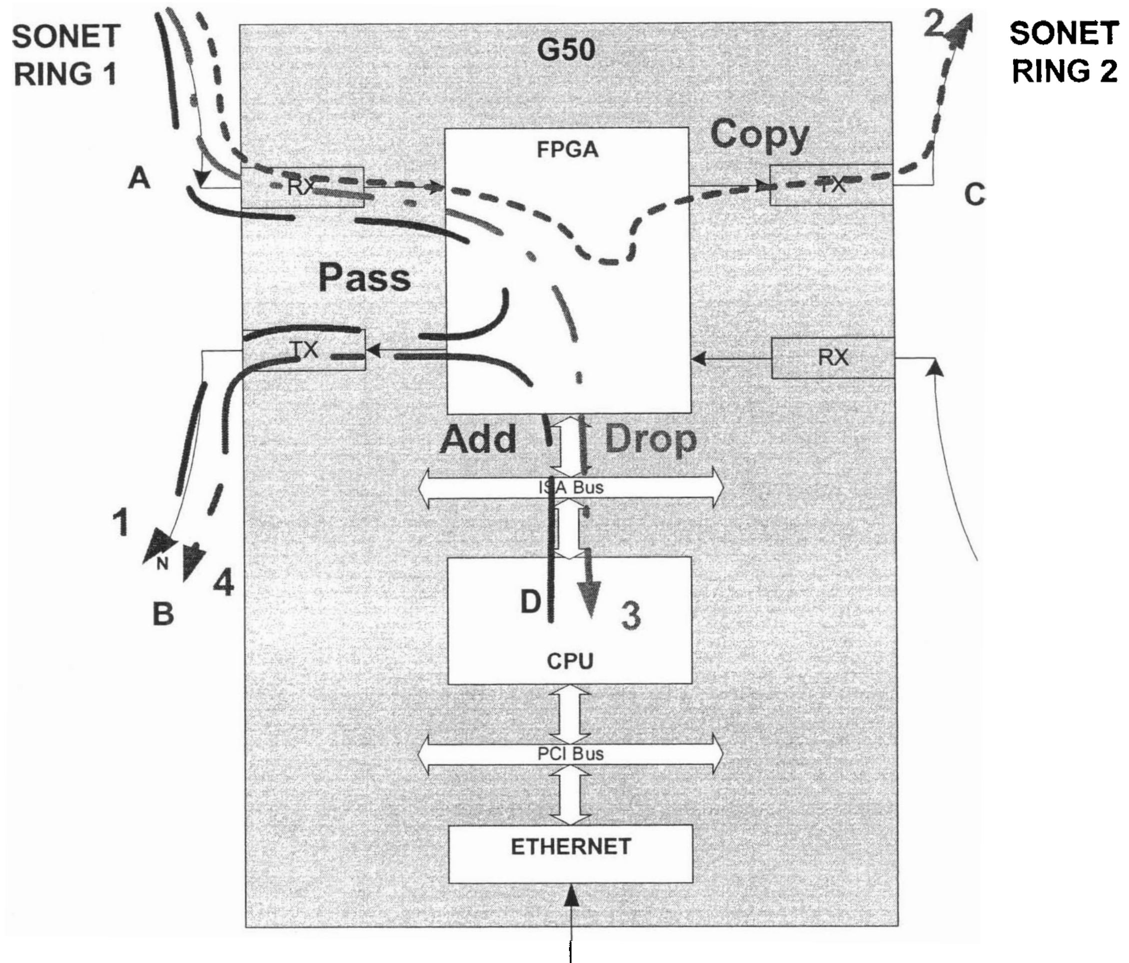

Figure 4.12 – G50 IPFS Frame Flow

4.2.2 FPGA Design

The basic modules in the FPGA are illustrated in Figure 4.13. There are two Rx modules – RxGet and RxPath. RxGet decodes the scrambled SONET frame and locates the IPFS frames in the SONET SPE. RxPath contains the RxLogic and CAMs to decide what to do with the incoming ERP frame. There are two Tx modules – TxPath and TxPut. TxPath contains the four FIFOs, a Look Up Table, and the TxLogic to decide which ERP frames should be sent. TxPut encodes the IPFS frames in the SONET SPE, and scrambles the SONET frame.

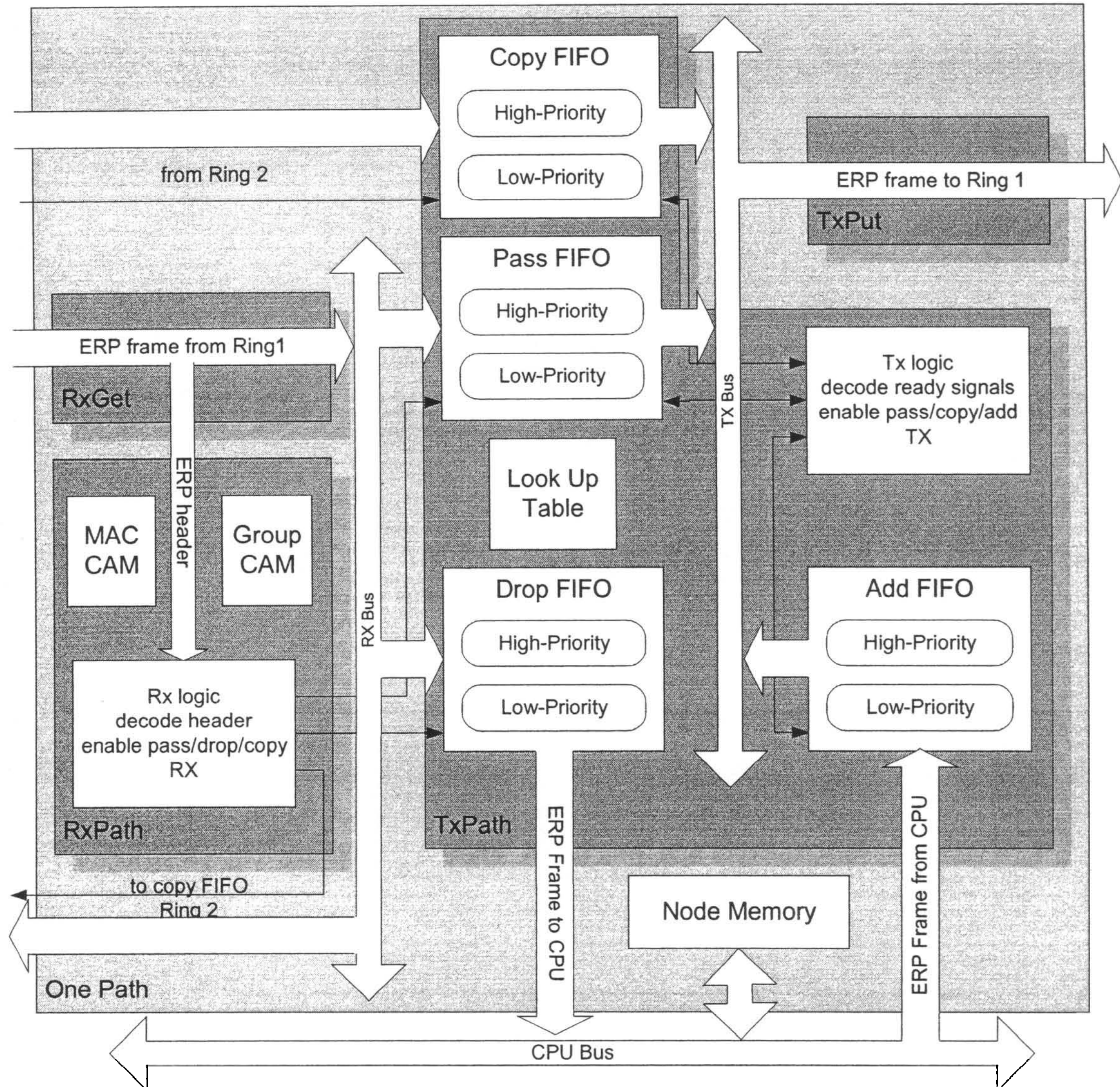

Figure 4.13 – G50 FPGA Design

Memory for storing node data, such as address and command registers, is also included. There are many internal busses and signals for communication between modules – only the most pertinent signals have been shown.

4.2.3 Router/Microprocessor

The CPU on the Router/Microprocessor board is a MachZ x86 processor with an on-chip PCI Bus interface. The main interconnections of this CPU are to an 8Mbyte Disk on Chip, 16Mbyte RAM module, RS-232 serial port, and ISA and PCI Busses.

The G50 runs either Linux OS or QNX Neutrino RTOS. The Linux OS is used when rapid and simple debugging of FPGA firmware is required – strictly for development; under Linux, a directory on the development PC can be mounted as a Samba directory, easily accessible from the G50. For operation as a gateway node, the G50 runs the QNX Neutrino *Real-time Operating System* (RTOS) [41] and routing utilities that support *Routing Information Protocol* (RIP) [27], *Border Gateway Protocol* (BGP) [42], and *Open Shortest Path First* (OSPF) [28]. A *Simple Network Management Protocol* (SNMP) [26] agent provides interfaces to an *Element Management System* (EMS) and a *Network Management System* (NMS). Our *distributed SLA Admission Controller* (dSLACtl) also runs sub-processes on the node to handle processing of SLA requests and provide path setup/removal and bandwidth allocation/de-allocation. Different sub-processes run on different nodes depending on the role the node plays in the HRN. For example, all ingress nodes need to run the SLA request sub-process, while non-ingress nodes do not. To interface the IPFS network with QNX's networking system, an IPFS-specific network device driver was constructed

4.2.4 Device Driver

The device driver abstracts the IPFS network, providing IP datagram communication for integration with other software components. The device driver is multi-threaded and has gone through many revisions to make it as robust and efficient as possible. The first implementation of the driver was not very efficient and resulted in a data rate of only 800Kbps. Many revisions later, the driver now transfers data at a rate of > 2Mbps using the ISA Bus (Rev. 2 removes this limitation by off-loading the data transfer from the CPU and ISA Bus to the FPGA and an internal PCI Bus). The device driver performs several important functions including interfacing between the networking system and the FPGA logic, and IP packet ↔ ERP frame segmentation and reassembly.

Data Transfer (CPU → FPGA)

The transfer path from the Microprocessor / Router Board to the FPGA Board uses ISA Bus 16-Bit Memory Writes.

Upon receiving an IP Packet from the networking system, the driver will:

construct the ERP frame header:

- set Hop Limit
- set Priority
- set Area Code
- set Mode

determine the HRN address

- if unicast frame → set to destination HRN address
- if broadcast frame → set to source HRN address
- if local broadcast frame → set to source HRN address
- if multicast frame → set to multicast group number

partition the IP packet into segments ≤ 244 Bytes

determine the flow value and the length of the overall IP packet

for each segment:

- set flow field to flow value
- set length field to overall length of packet
- prepend the ERP Frame Header onto the front of the segment
- transfer via ISA Bus to the FPGA

Successful transfer of the ERP frame to the FPGA generally means that the frame has been sent on the optical fiber. Frames are queued temporarily in the Add FIFO waiting for transmission in an appropriate IPFS frame, so there is a slight chance that frames can be contained in FPGA memory space having been transferred over the ISA Bus successfully, and not having made it onto the actual fiber itself. If an error occurs due to a power glitch or node shutdown resulting in only partial datagram delivery, it will be caught by higher-level protocols – similar to the case of a data packet being lost or dropped while in transit through a network.

Data Transfer (FPGA → CPU)

The transfer path from the FPGA Board to the Microprocessor / Router Board uses ISA Bus 16-Bit Memory Reads.

Upon receiving an ERP frame from the optical interface, the driver will:

transfer the ERP frame via the ISA Bus to system memory

unpack the ERP frame to extract the encapsulated payload

examine the overall length of the packet and reassemble using subsequent frames from the same flow until the full IP packet is received

hand over fully reassembled IP packet to the networking system

Successful transfer of the ERP frame from the FPGA Board generally means that the frame has been received properly. However, the IP packet will not be handed over to the networking system until the full packet has been received, so there is a possibility that an IP datagram can be contained in system memory without yet being transferred to the networking system. If this results in an error, it will be caught by higher-level protocols – similar to the case of a data packet being lost or dropped while in transit through a network.

4.2.5 Segmentation & Reassembly

Detailed examples of how the network device driver segments and reassembles IP packets are given in Appendix D. We included segmentation and reassembly in the driver to enhance performance. For this we added two fields to the original ERP header – the flow identifier and the length/type field. Since the IP packet is segmented into chunks $\leq$ 244 Bytes for inclusion in the ERP payload, some form of identifier was needed to reassemble all the chunks on reception. And we needed to know when all of the chunks for a particular IP packet had been received. For simplicity, the current driver uses a simple 2 Byte random number as a flow identifier. This flow identifier is unique to a single IP packet and identifies all of the chunks the same. Also the overall length of the IP packet is copied into the ERP header so that the driver on the receiving side can determine when the full IP packet has been received and reassembled. This idea assumes that ERP frames cannot be received out of order for a given IP packet. Given the design and operation of the IPFS HRN, this assumption is reasonable.

4.3 QoS Support

Quality of Service support is built into IPFS from the ground up using a combination of hardware and software components. The IPFS frame is fixed-size, translating into deterministic delays – no waiting for a frame of arbitrarily large and unknown length to be transmitted and no complex preemption or buffering schemes. Bi-level priority FIFO structures in each node supply two distinct paths through each node – high priority and low priority; each FIFO structure – Add, Pass, Drop, and Copy FIFOs – has two queues so that priority handling is uniform from ingress to egress. This effectively provides two

paths, of high and low priority, respectively, throughout the IPFS network. Also, the ERP header has a field for specifying the priority of the ERP frame and the IPFS header includes a field for determining the scheduling of the ring – more about the latter in a following section, but first a discussion of priorities.

4.3.1 Priority Scheme

The priority scheme is comprised of bi-level priority hardware FIFOs and a priority field carried in the ERP frame header, as well as associated hardware logic. Each of the hardware FIFOs has two priority levels – low-priority and high-priority. The ERP header priority field is 3-bits wide to handle eight levels of priority (values between 0 and 7 in decimal – 000 and 111 in binary); the ERP header priority field values are mapped as high priority: $4 \leftrightarrow 7$, and low priority: $0 \leftrightarrow 3$. A frame with low priority ($0 \rightarrow 3$) will travel between nodes using the low-priority FIFO structures and the high priority ($4 \rightarrow 7$) frames will be transported via the high-priority FIFO structures. At each node, high priority frames have absolute transmit precedence over low priority ones – all high-priority frames are sent before any low-priority frame is sent. They also take precedence when transferring between hardware and software modules within the node. In the G50 One Path module there are four FIFO structures – Copy, Pass, Drop, and Add. In each of these FIFOs there are a low-priority queue and a high-priority queue. Once the appropriate FIFO structure has been enabled by the RX logic, the FIFO itself determines which queue to place the ERP frame in by examining the priority field in the ERP header.

If the IPFS node receives a low priority ERP frame, which it should pass to the next node, this ERP frame will be transferred to the Pass FIFO and subsequently be placed in the low-priority queue. If a high priority ERP frame then comes in likewise, this frame will be transferred to the high-priority queue. On the transmit side, the Pass FIFO will first transfer all ERP frames which are in its high-priority queue before transmitting any frames in its low-priority queue. This same logic applies to each of the other FIFO structures – all high priority frames will take precedence over any low-priority frames.

This priority scheme gives IPFS the ability to partition user data into two classes: lowest-latency and standard best effort. Giving guaranteed traffic a high priority and assigning a low priority to standard traffic assures that guaranteed traffic will get through the network

as quickly as possible, with no interference from the low-priority traffic. This priority scheme is designed for future support of DiffServ-like *Class of Service* (CoS). The bi-level FIFO structures can be replaced with FIFOs with more levels so that traffic can be groomed based on service classes. This priority scheme provides support for relative precedence – Soft QoS – not for absolute QoS guarantees; the scheduling scheme provides the needed support for guaranteed delivery – Hard QoS.

4.3.2 Scheduling Scheme

The scheduling scheme allocates ring bandwidth, in units of frame slots, to nodes; thereby, allocating guaranteed bandwidth to customer sessions.

The IPFS header contains a Schedule field that carries an 8-bit Schedule ID; an IPFS node uses the Schedule ID to decide which ERP frame it can send in the associated IPFS frame. Only the master node of a ring determines which Schedule ID is carried in the Schedule field. The master node implements a Schedule table using a 2Kbyte dual-port RAM, where one side is connected through the Bus to the node's CPU and the other side is connected inside the FPGA to the TxPath module. A scheduling utility sets the values in the Schedule table and the TxPath module steps through this table placing the values in the Schedule field of a transmitting IPFS frame. Slave nodes are not permitted to modify the contents of the Schedule field in the IPFS frame header; instead they simply copy the Schedule field from the receiving frame through to the transmitting frame.

Each node (master or slave) has its own static 8-bit Schedule ID stored in a register. When the node is sending an IPFS frame it compares the Schedule field of the transmitting IPFS frame with its own stored Schedule ID and sends the appropriate frame from either its Copy, Pass, or Add FIFOs. This is based on the scheduling algorithm:

Hard QoS – a node is permitted to add frames to the ring (from either the Add or Copy FIFOs) only when its Schedule ID appears in the transmitting frame Schedule field, and can only pass frames when its Schedule ID does not appear in the transmitting frame Schedule field.

Best Effort – Pass frames take precedence over Copy frames, which take precedence over Add frames. In other words, frames that should be passed on to the next node are transmitted first, followed by frames that are added to this ring from a different ring, and finally frames the node wants to add from its external interface are transmitted.

Without a scheduling scheme such as this, only variations of best effort delivery can be supported on the HRN. However with this scheduling scheme, strict-guaranteed delivery can also be supported. The pseudo code for the method by which the TxPath module fills in the TxSched field and chooses which ERP frame to send is given in Figure 4.14.

```
TxSched <= RxSched     when !master else
           nextSchedID from SchedTable;

if (TxSched == mySchedID)
    mySched = TRUE;
else
    mySched = FALSE;

if (bestEffort) {
    TxFrame <= PassFrame  when PassFIFO !empty else
               CopyFrame when CopyFIFO !empty else
               AddFrame   when AddFIFO  !empty else
               NullFrame;
} else {                                                      // HardQoS
    TxFrame <= PassFrame  when PassFIFO !empty and !mySched   else
               CopyFrame when CopyFIFO !empty and  mySched    else
               AddFrame   when AddFIFO  !empty and  mySched   else
               NullFrame;
}
```

Figure 4.14 – TxSched/TxFrame Pseudo Code

A master node is responsible for scheduling its direct sub-ring only; sub-rings of that sub-ring are scheduled by their corresponding master nodes, and so on. In this way, the Scheduling system for the overall IPFS network is distributed amongst the master nodes – each master node is in charge of its own sub-ring. From this distributed Scheduling system we develop our *distributed SLA Admission Controller*, so it is important to understand how the Scheduling system works. We step through a simple example of scheduling one ring and provide more complex scenarios in Appendix E.

Example

Suppose we have a HRN as illustrated in Figure 4.15, and we wish to guarantee that one out of every 2048 frames sent on the 01-01 sub-ring is allocated to node 01-01-01-00 and two out of every 2048 frames are allocated to node 01-01-02-00. The Schedule IDs for these two nodes are their node IDs: 01 and 02, respectively.

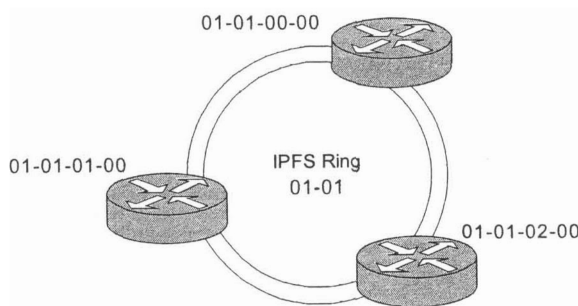

Figure 4.15 – Example HRN Sub-ring for Ring Scheduling

The master node will have a Schedule table with one entry being 01 and two entries being 02, similar to Table 4.1. Then, as the Schedule table pointer comes around to these entries, the master node will insert the values 02, 01, 02 into the Schedule field of three subsequent frames as in Figure 4.16.

Table 4.1 – Example Schedule Table (partial)

...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	02	...	...	...	...	...	...
...	...	01	...	...	...	...	...	...
...	...	02	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

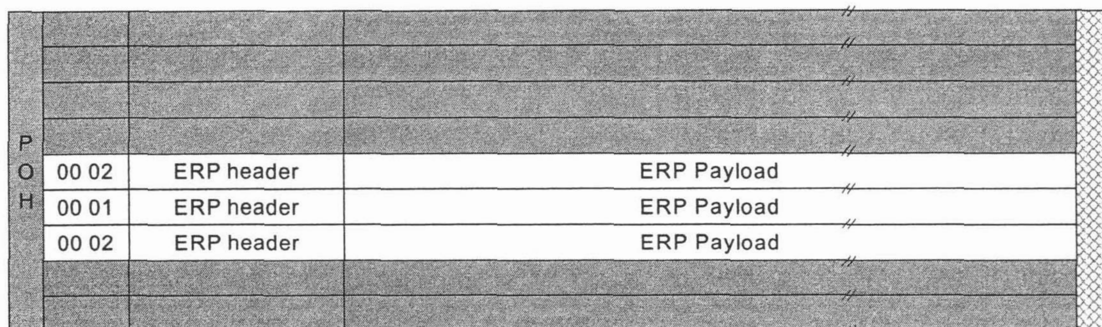

Figure 4.16 – Example STS-3 SPE with Scheduled IPFS Frames

When the STS-3 frame comes to node 01-01-01-00, the node will drop those ERP frames with destination address matching its own address and will put all others into the Pass or Copy FIFOs. When it transmits the STS-3 frame it will copy the IPFS header straight-through and will insert ERP frames from the Pass FIFO into all those IPFS frames having Schedule field ≠ 01 (its own Schedule ID). In the IPFS frames with Schedule field = 01, the node will insert ERP frames from its Add or Copy FIFOs.

When this STS-3 frame comes to node 01-01-02-00, the node will drop those ERP frames with matching destination address and will put all others into the Pass or Copy FIFOs. When it transmits the STS-3 frame it will copy the IPFS header straight-through and will insert ERP frames from the Pass FIFO into all those IPFS frames having Schedule field ≠ 02 (its own Schedule ID). In the IPFS frames with Schedule field = 02, the node will insert ERP frames from its Add or Copy FIFOs. When the master node receives the STS-3 frame, the node will drop, pass, or copy the ERP frames as required. The master node will replace the contents of the Schedule field of the transmitting IPFS frames with the next applicable entries in its Schedule table, instead of copying the field straight-through as in the case of all the slave nodes.

Since there are 2048 entries in the Schedule table, we have effectively scheduled node 01-01-01-00 with one frame slot out of 2048 and node 01-01-02-00 with two frame slots out of 2048. This discussion assumes that all of the nodes are configured to use Hard QoS scheduling.

4.3.3 Scheduling Granularity

Scheduling granularity refers to the minimum and incremental unit of bandwidth schedulable in the network. At the OC-3c line rate of 8000 STS-3 frames per second, IPFS has a frame rate of 9 x 8000 = 72000 frames per second.

Table 4.2 – IPFS Line / Data rates

IPFS frame rate: 72000fps	bps	Kbps	Mbps
IPFS line rate: (72000 * 258)	148608000	148608	148.608
IPFS data rate: (72000 * 244)	140544000	140544	140.544
IPFS schedule rate: (72000/2048)	35.15625 *fps*		
Line: (258)	**2064** ***bits per frame***		
⇒ effective line rate (1 frame scheduled):	72562.5	72.5625	0.0725625
Payload: (256 - 12 = 244)	**1952** ***bits per frame***		
⇒ effective data rate (IPFS granularity):	**68625**	**68.625**	**0.068625**
frame service interval: (2048/72000)	**28.4** ***msec***		
DS0 rate:	64000	64	0.064

As shown in Table 4.2, using a scheduling table with 2048 entries provides IPFS with the granularity of an effective data rate of ≅ 69Kbps, which closely matches that of a typical voice line (DS0) at 64Kbps.

The effective data rate is calculated as:

$$\begin{aligned}\text{effective data rate} &= p * s \\ &= (244 \times 8) * (72000/2048) = 68.625 \text{ Kbps} \\ \text{where } p &= \text{payload size in bits per frame} \\ s &= \text{schedule rate in frames per second}\end{aligned}$$

The frame service interval is the amount of time it takes for the Schedule table pointer to traverse the whole table. This gives an upper bound to the amount of delay between the time when a frame slot is scheduled in the Schedule table, and when an IPFS frame is marked with that Schedule ID. The frame service interval is calculated as:

$$\begin{aligned}\text{frame service interval} &= 1/s \\ &= 1/(72000/2048) \cong 0.0284 = 28.4 \text{ msec. per frame} \\ \text{where } s &= \text{schedule rate in frames per second}\end{aligned}$$

With a frame service interval of 28.4 msecs, IPFS can rapidly respond to dynamic requests for service. With a scheduling granularity of 68.625 Kbps (as small as one DS0 ≅ 64 Kbps), IPFS can be scheduled very efficiently to support any required data rate. IPFS scheduling for common payloads is given in Table 4.3.

Table 4.3 – IPFS Scheduling for Common Payloads

Client Payload	bitrate (Mbps)	IPFS Schedule	IPFS Bandwidth	efficiency
DS0	0.064	1	0.068625	93.26%
T1	2	30	2.058750	97.15%
10 Mbps Ethernet	10	146	10.019250	99.81%
T3	45	656	45.018000	99.96%
STS-1	51	744	51.057000	99.89%
Fast Ethernet	100	1458	100.055250	99.94%
ESCON	160	2332	160.033500	99.98%
Fibre Channel	850	12387	850.057875	99.99%
FICON	850	12387	850.057875	99.99%
1 Gigabit Ethernet	1000	14572	1000.003500	99.9997%

For example to carry a T3 payload we would schedule 656 frame slots in the Schedule table; a 10Mbps Ethernet connection requires scheduling 146 frame slots. The columns in the table are given as the requested bitrate (in Mbps), the IPFS Schedule – how many frame slots to schedule in the Schedule table, the IPFS Bandwidth – how much

bandwidth this schedule actually provides, and what the efficiency of the scheduling is – how much of the scheduled bandwidth is not wasted by over-allocating. The scheduling efficiency is calculated as:

$$\text{scheduling efficiency} = \frac{r}{\left(\left\lceil \frac{r}{f} \right\rceil \times f\right)}$$

where r = requested data rate

f = data rate for one frame scheduled (68.625 Kbps)

As an example, the scheduling efficiency of a 10Mbps Ethernet payload is:

$$10\text{ Mbps scheduling efficiency} = \frac{10000}{\left(\left\lceil \frac{10000}{68.625} \right\rceil \times 68.625\right)} = \frac{10000}{\left(\lceil 145.719 \rceil \times 68.625\right)}$$

$$= \frac{10000}{(146 \times 68.625)} = \frac{10000}{10019.25} \cong 99.81\%$$

The scheduling efficiencies are high in Table 4.3 because the scheduling granularity is relatively small compared to the bandwidth requested for the client payload.

4.4 Summary

IPFS technology uses the HRN and ERP protocols to add L2-switching to a SONET network. A development node, called the G50, has been implemented and can be interconnected to form various HRN configurations. Software utilities and an IPFS specific device driver integrate the IPFS technology into an IP networking system. QoS is supported at the hardware and software levels – using a priority scheme for soft QoS and a scheduling scheme for hard QoS. The small scheduling granularity (≅ 69Kbps) means that IPFS can be efficiently mapped to common client payloads. There is no special provisioning mode for IPFS nodes; scheduling of frame slots occurs during normal operation of the network. The small frame service interval (< 30 msecs) means that the path building time – the delay between scheduling a frame slot and being able to use the corresponding frame – are minimal. The scheduling scheme is the foundation on which we develop our *distributed SLA Admission Controller* (dSLACtl), the subject of the next chapter.

Chapter 5
The Controller (dSLACtl)

Our final solution to the guaranteed QoS problem as described in Section 1.2 was to integrate the lessons we learned from our initial attempt in building an *SLA Admission Controller* (SLACtl) into a new technology, one which has the required support built in from the initial design stage. While the design of SLACtl used a top-down methodology, the design for our *distributed SLA Admission Controller* (dSLACtl) follows a bottom-up approach. In this chapter we describe the architecture and design of dSLACtl, starting with an analysis of requirements. Note that for dSLACtl, the only *Quality of Service* (QoS) parameter we are concerned with is bandwidth. Latency is minimized by the design of the IPFS nodes and we have no control over either this or jitter at this point in the design process.

5.1 Analysis

Similar to QoSNET, the IPFSNET concept is to use an admission controller to allow/disallow traffic streams through the network. A customer requests admission by specifying the stream parameters (endpoints, BW). The admission controller then determines if it can admit the new stream while respecting current streams, and will schedule the appropriate resources, if possible. No attempt is made, at this point, to select a revenue-optimal subset of the requestors.

Our *distributed SLA Admission Controller* is designed around the interfaces to various other components, such as the packet classifier, the scheduler, and the customer. Like SLACtl, dSLACtl uses SLAs for customer communication of path requirements to dSLACtl. An SLA consists of the specification of endpoints, such as source and destination IP addresses, and the specific desired characteristics of the traffic stream, in this case strictly bandwidth. The interface to the packet classifier, which resides on the ingress gateway, consists of a specification of which packets should be classified as belonging to a particular stream, as well as a specification of the stream characteristics so that packets can be marked as compliant or non-compliant to the stream. The packet classifier is beyond the scope of this work; so we assume that there is such an entity and it performs its job as specified.

However, we have developed the scheduler in detail in this thesis because it is vital to the operation of the admission controller. Schedulers are distributed across the IPFS network and the admission controller is likewise distributed. The distributed nature of this controller lies in the fact that the master node of a ring is responsible for admitting traffic only from the other nodes on its immediate sub-ring. The master node of one ring can be a slave node of a second ring, in which case that ring's master is responsible for admitting traffic from the nodes on that ring. This second master can be a slave of a different ring, and so on. Thus, each master node is responsible only for its sub-ring of the HRN.

5.2 Architecture

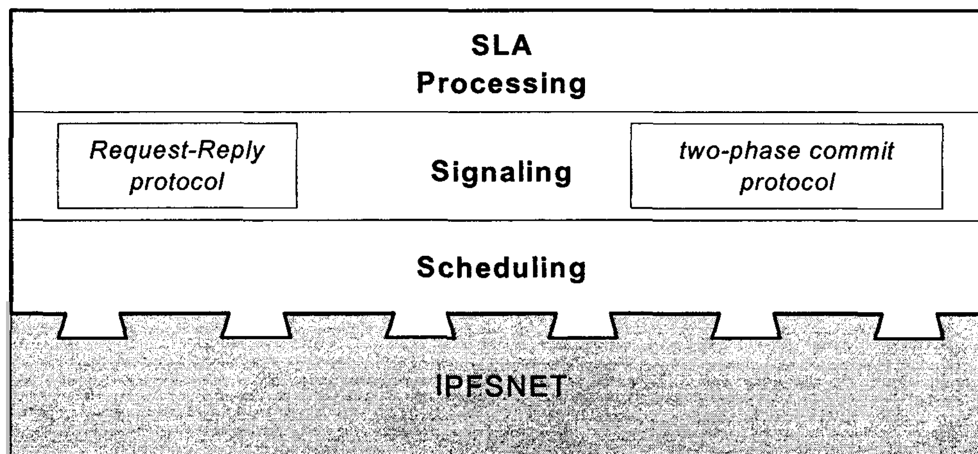

Figure 5.1 – dSLACtl Architecture

The architecture of dSLACtl is illustrated in Figure 5.1. There are three parts: the *Scheduling system*, the *Signaling system*, and *SLA Processing*. The Signaling system uses two protocols: the *Request-Reply protocol* and the *two-phase commit protocol*. Our admission controller relies on built-in support from the IPFS network, specifically for the Scheduling system.

5.2.1 Scheduling System

The Scheduling system has two parts: a *Look Up Table* (LUT) in hardware, called the Schedule table, and a software module that sets values, called Schedule IDs, in the LUT. A Scheduler runs on each master node in an HRN and schedules frame slots for nodes on its immediate sub-ring. In the last chapter we described how a node uses the Schedule field in the IPFS header to determine what ERP frame it can send in a given IPFS frame. We extend this here to show how a software module sets the Schedule ID in the Schedule table of the master node. The pseudo code for adding a schedule or removing a schedule is given in Figure 5.2.

In the addSchedule procedure, we check for enough free frame slots for the request and calculate an offset into the Schedule table; this ensures the scheduled frame slots will be uniformly distributed. While we have not scheduled all of the requested frame slots, we check to see if the current Schedule table entry has not already been scheduled. If it is free, we set the value in the Schedule table to our requested Schedule ID, update the number of scheduled and free frame slots, and then add the offset to the current Schedule table pointer. If the entry is not free, then we move to the next entry in the Schedule table.

The removeSchedule procedure is similar, but instead of looking for free entries, we look for entries that are scheduled with the requested Schedule ID. When we find one, we reset the Schedule ID in the Schedule table and update the scheduled and free frame slots counters. This continues until we have found and freed the requested number of frame slots.

```
tableSize = 2048;
freeFrames = tableSize;

addSchedule(reqFrames, schedID) {
    if (reqFrames > freeFrames)
        return;                     // not enough free slots for request

    offset = tableSize / reqFrames;
    i = 0;
    schedFrames = 0;
    while(schedFrames < reqFrames) {
        if (schedTable[i] == notSched) {
            schedTable[i] = schedID;
            schedFrames++;
            freeFrames--;
            i = (i + offset) mod tableSize;
        } else
            i++;
    }
}
removeSchedule(reqFrames, schedID) {
    offset = tableSize / reqFrames;
    i = 0;
    schedFrames = reqFrames;
    while(schedFrames > 0) {
        if (schedTable[i] == schedID) {
            schedTable[i] = notSched;
            schedFrames--;
            freeFrames++;
            i = (i + offset) mod tableSize;
        } else
            i++;
    }
}
```

Figure 5.2 – Scheduler Pseudo Code

To set up an end-to-end path through the HRN requires setting the appropriate Schedule ID in each of the Schedule tables in each of the master nodes on each of the rings that the path must traverse. This scheduling process is the job of the Scheduling system.

There are three distinct scheduling scenarios:

1. **Same Ring Scheduling**
   the source node is on the same ring as the destination node.

2. **Sub-ring to Super-ring Scheduling**
   the source node is on a sub-ring of the ring with the destination node.

3. **Super-ring to Sub-ring Scheduling**
   the source node is on a super-ring of the ring with the destination node.

Detailed examples of each of these scenarios are given in Appendix E. Combining these scenarios gives a generalization for any multi-ring scheduling needed in an HRN.

Example -- Generalized Multi-Ring Scheduling

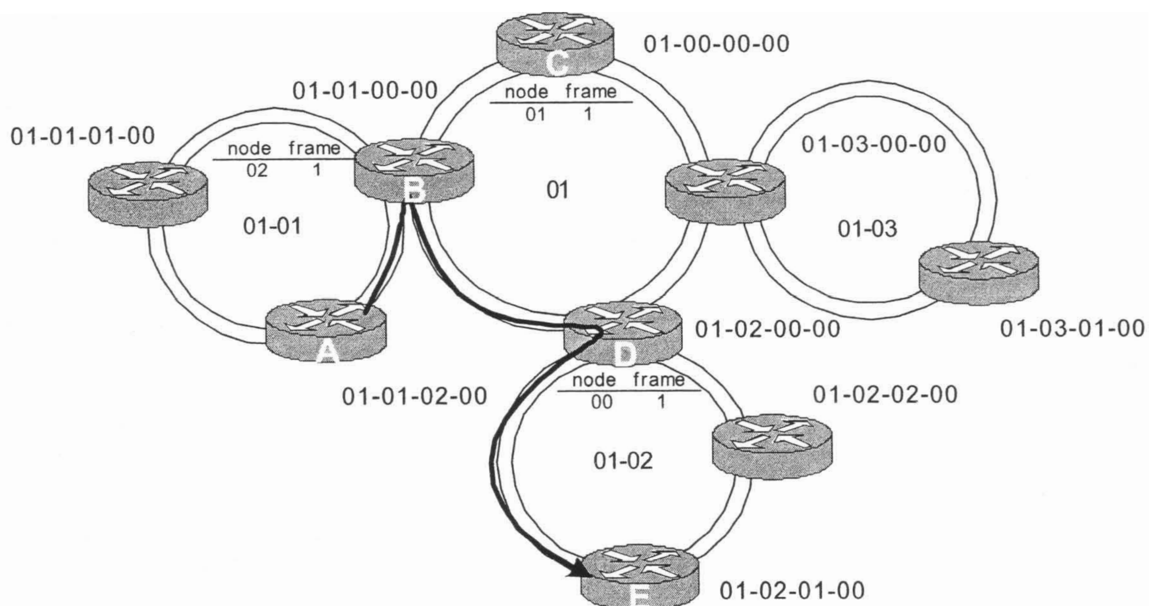

Figure 5.3 – Network Layout: Generalized Multi-Ring Scheduling

Suppose we wish to setup a DS0 between node A (01-01-02-00) and node E (01-02-01-00) as shown in Figure 5.3. Traffic for this DS0 will need to traverse three rings – ring 01-01, ring 01 and ring 01-02. We schedule one frame slot on ring 01-01 for node A (01-01-02-00) using master node B (01-01-00-00). We also schedule one frame slot on ring 01 for node B (01-01-00-00) using master node C (01-00-00-00) and one frame slot on ring 01-02 for node D (01-02-00-00) using master node D (01-02-00-00). This scheduling gives us a DS0 on each of the rings 01-01, 01, and 01-02.

This is how multi-ring scheduling in an HRN is used to schedule the necessary frame slots for an end-to-end path and leads into a discussion of signaling. That is, a discussion of how to communicate the necessary scheduling requests to each of the master nodes controlling each of the sub-rings in which we need to set up our schedule. We develop the necessary signaling in a following section, but first a look at schedule aggregation.

Schedule Aggregation

Schedule aggregation refers to the process of combining the scheduling from many requests into one. If a request requires a master node to request scheduling in a super-ring for some node on its own sub-ring, it will make the request using its own Schedule

ID. That is, all of the requests from a node to its super-ring are *aggregated* together and all have the same Schedule ID – the Schedule ID of the master; this is illustrated by working through an example.

Example

Suppose we have the IPFS HRN as given in Figure 5.4 and further suppose we have a DS0 (solid arrow) between node A (01-01-02-00) and node E (01-02-01-00). Then we wish to setup a DS0 (dashed arrow) between node F (01-01-01-00) and node G (01-02-02-00). For the existing DS0 we have a schedule of one frame slot on rings 01-01, 01, and 01-02 using master nodes B (01-01-00-00), C (01-00-00-00), and D (01-02-00-00), respectively. The schedule on ring 01-01 would have Schedule ID 02 (for node 01-01-02-00). The schedule on ring 01 would have Schedule ID 01 (for node 01-01-00-00). And the schedule on ring 01-02 would have Schedule ID 00 (for node 01-02-00-00).

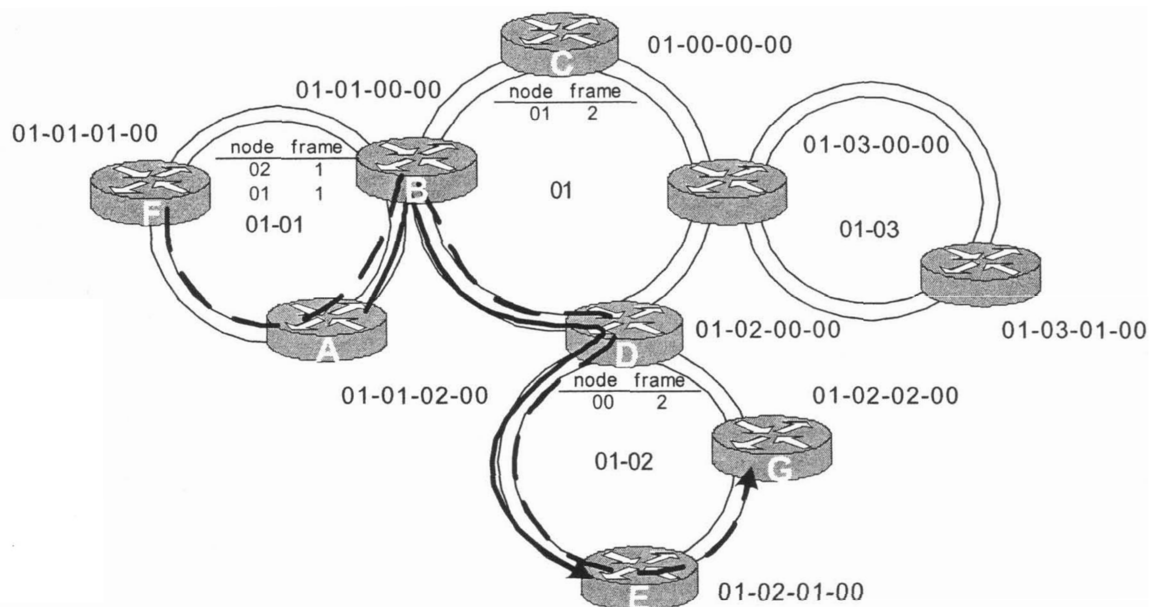

Figure 5.4 – Network Layout: Schedule Aggregation

Adding a DS0 from node F (01-01-01-00) to node G (01-02-02-00) would require scheduling one frame slot on each of the rings 01-01, 01, and 01-02 using master nodes B (01-01-00-00), C (01-00-00-00), and D (01-02-00-00). The Schedule ID on ring

01-01 would be 01 (for node 01-01-01-00), on ring 01 would be 01 (for node 01-01-00-00), and on ring 01-02 would be 00 (for node 01-02-00-00).

Combining these two schedules gives an overall schedule of one frame slot with Schedule ID 02 and one frame slot with Schedule ID 01 on ring 01-01; a schedule of two frame slots with Schedule ID 01 on ring 01; and a schedule of two frame slots with Schedule ID 00 on ring 01-02. The schedule on the super-ring 01 is the aggregate of the schedule on the sub-ring 01-01.

Aggregated schedules do not have to remain aggregated; they can split apart as the streams they represent diverge through the HRN. Suppose we add a DS0 from node F (01-01-01-00) to node I (01-03-01-00) as shown in Figure 5.5 (dotted arrow). This DS0 would have a schedule of one frame slot in each of the rings 01-01, 01, and 01-03. The Schedule IDs for this schedule would be 01 on ring 01-01, 01 on ring 01, and 00 on ring 01-03.

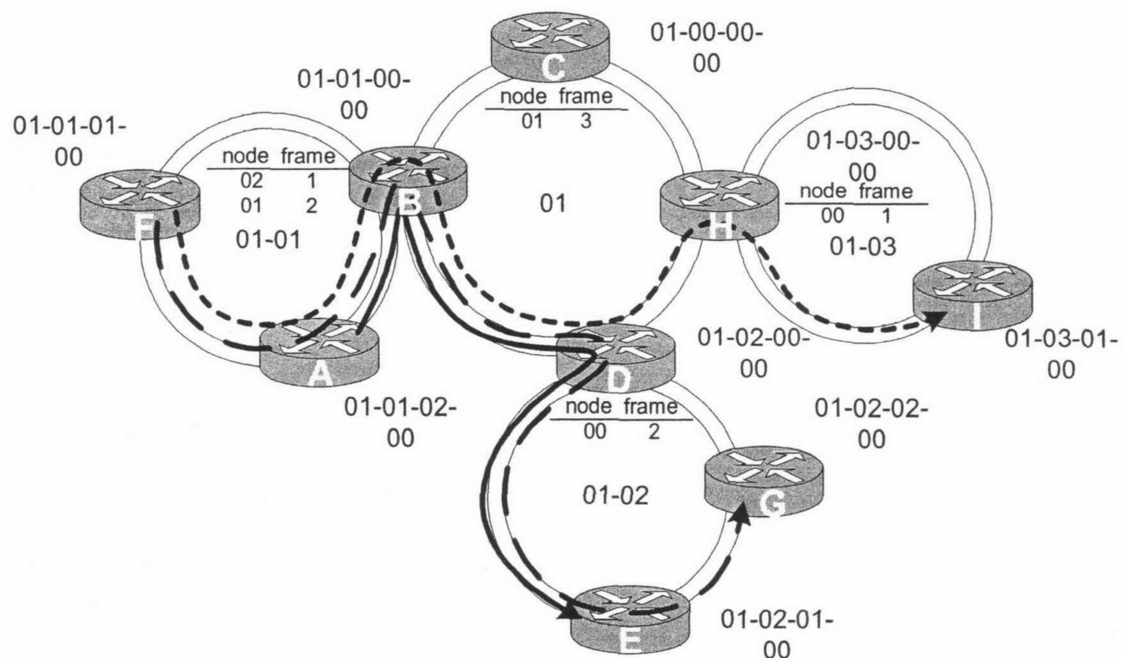

Figure 5.5 – Network Layout: Schedule Aggregation – adding a DS0

Therefore, the overall schedule will be one frame slot with Schedule ID 02 and two frame slots with Schedule ID 01 on ring 01-01; three frame slots with Schedule ID 01 on ring 01; two frame slots with Schedule ID 00 on ring 01-02; and one frame slot with Schedule ID 00 on ring 01-03.

The schedule (3 frame slots) on the super-ring 01 is the aggregate of the schedule (1 + 2 frame slots) on the sub-ring 01-01, but the schedule is split between the next two sub-rings; 01-02 has two frame slots scheduled and 01-03 has one frame slot scheduled. This aggregation mechanism is very important for the scalability of the Scheduling system. As the IPFS HRN grows, the Scheduling system does not increase in complexity.

With this background on the Scheduling system, we now develop the Signaling system for our admission controller.

5.2.2 Signaling System

The admission controller's internal processing can be thought of as an abstraction of the interaction or signaling between master nodes as required by an SLA request. The basic ideas behind the signaling process have been introduced throughout the discussion of the Scheduling system; now, we continue the development of this signaling by moving up a level and looking at how nodes communicate with each other to set up a path, schedule frame slots, and allocate bandwidth. These are performed using the Request-Reply protocol, and a variant of the two-phase commit protocol.

Request-Reply Protocol

The *Request-Reply protocol* (RR) [15] is used for inter-nodal communication, where one node plays the role of client and another node plays the role of server. In the RR protocol that we use, there are three message communication primitives: *DoOperation*, *GetRequest*, and *SendReply* as illustrated in Figure 5.6.

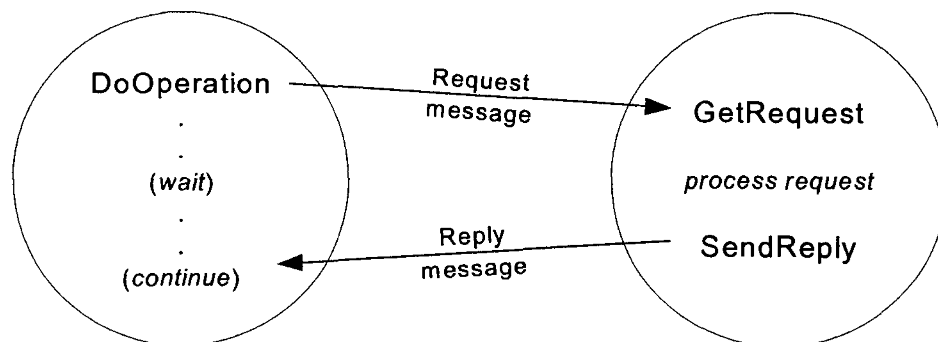

Figure 5.6 – Request-Reply Protocol

DoOperation is used by clients to invoke remote server operations. The message specifies the request ID, the operation ID, and any required arguments. *GetRequest* is used by the server to receive requests from clients. When the request has been processed, the server uses *SendReply* to send a reply message back to the client. The reply message contains the request ID, and the result of the operation. The *DoOperation* blocks the client, waiting on the reply message. When the client receives the reply message it unblocks and continues with processing.

The *DoOperation* primitive in our Signaling system is called a Path request, either a Path-add request or a Path-remove request. Both of these requests cause the requestor (client) to block waiting for a reply. The requestee (server) gets the request, processes the request – which may include sending its own Path request to another node – and replies to the requestor. The reply will either be a valid PathID (>0) or an errorID (<0). A timeout is used to prevent deadlock in the case of a lost request or reply message.

Two-Phase Commit Protocol

A variant of the *two-phase commit* protocol [16] for nested transactions is used to reserve and commit scheduling transactions; this is to provide atomicity in our scheduling system – either a path is completely setup and the frame slots scheduled on all necessary rings or not at all. In the usual two-phase commit, a coordinator is used to communicate to all the workers in a distributed transaction. In the first phase the coordinator asks all the workers if they are ready to commit and in the second it tells them to commit (or abort) the transaction. There is a voting phase and a completion phase as shown in Figure 5.7.

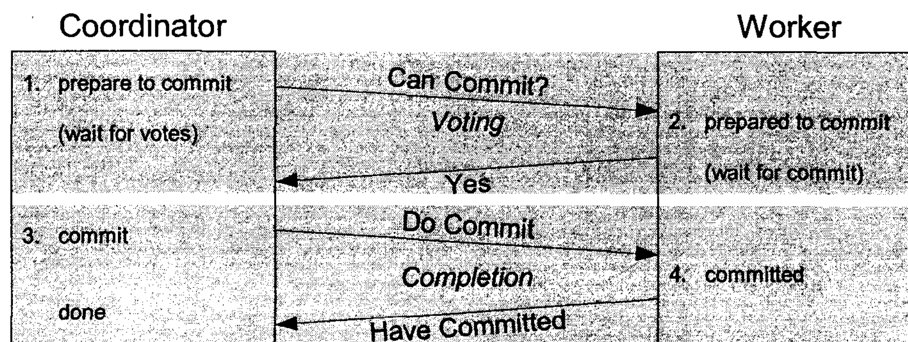

Figure 5.7 – Two-Phase Commit Protocol

In our variant of the two-phase commit protocol, there is no coordinator. Instead, as a Path request is processed by a node, the node completes the first phase (voting) by reserving a schedule and passing the request on to the next node to denote a *yes* vote. If the node cannot reserve the schedule then it immediately replies to the requestor; this denotes a *no* vote. The completion phase is started by either the final node in the Path request successfully (unsuccessfully) scheduling the frame slots and sending a success (failure) in its reply message, or by any node in the Path request that cannot reserve the schedule. If a node receives a reply message containing an errorID, then the node aborts the schedule transaction, by freeing the reserved frame slots, and replies to its requestor with a failure message. If a node receives a reply message containing a valid PathID then the node commits the schedule and replies to its requestor with a success message containing a valid PathID.

This variant sends fewer messages than the standard two-phase commit protocol, but can take more time as the process is serialized – concurrency is removed.

The reserve, commit, and free procedures added to the Scheduler pseudo code for the two-phase commit protocol are given in Figure 5.8.

```
reserveSchedule(reqFrames, schedID) {
    if (reqFrames > freeFrames)
        return;                         // not enough free slots for request
    freeFrames -= reqFrames;
}
commitSchedule(reqFrames, schedID) {
    offset = tableSize / reqFrames;
    i = 0;
    schedFrames = 0;
    while(schedFrames < reqFrames) {
        if (schedTable[i] == notSched) {
            schedTable[i] = schedID;
            schedFrames++;
            i = (i + offset) mod tableSize;
        } else
            i++;
    }
}
freeSchedule(reqFrames, schedID) {
    freeFrames += reqFrames;
}
```

Figure 5.8 – Reserve, Commit, Free Scheduler Pseudo Code

The addSchedule procedure has been separated into two procedures: reserveSchedule and commitSchedule. The reserveSchedule procedure checks to see that there are

enough free frame slots for the request and then subtracts the requested number from the free frame slot counter. The Schedule table is not updated so that the schedule is only reserved and not committed – this happens in the commitSchedule procedure. The freeSchedule procedure frees the reserved frame slots by adding the requested number back to the free frame slot counter. This is a very simple way of providing schedule reservation and depends on a reliable signaling protocol.

These Scheduling system procedures are called from the addPath and removePath procedures, which are invoked in response to the Path request from another node. The pseudo code for addPath is given in Figure 5.9.

```
pathID = 0;
addPath(source, destination, reqFrames) {
    if !(isSlave(source) || isMaster(source)) {
        reply(FAIL);
        return;                     // don't control schedule for source
    }
    if (isSlave(destination)) {
        addSchedule(reqFrames, schedID(source));
        if (success) {
            reply(++pathID);
            return;
        }
    } else {
        reserveSchedule(reqFrames, schedID(source));
        if (success) {
            source = myAddress;
            masterNode = nextMasterInPath(destGW);
            sendPathRequest(masterNode,
                                addPath(source, destination, reqFrames));
            if (success) {
                commitSchedule(reqFrames, schedID(source));
                reply(++pathID);
                return;
            }
            freeSchedule(reqFrames, schedID(source));
        }
    }
    reply(FAIL);
    return;
}
```

Figure 5.9 – addPath Pseudo Code

When a node receives a Path-add request, it checks to see if it controls the scheduling for the source node; if it doesn't control the scheduling then the request is in error and a failure reply is returned. If it does control the scheduling for the source then the node checks to see if the destination node is on its sub-ring; this is so that the node can decide

whether it needs to send a Path request to the next master node in the requested path. If this node is the last master in the path, then it schedules the source node and sends a reply message containing a PathID to the requestor.

If this node is not the last master in the path, then it reserves the schedule and sends a Path-add request to the next master, substituting its own address as the source address. When this node receives a reply from its Path request, it commits the previously reserved schedule and replies back to its requestor with a valid PathID (in the case of a successful request). If the request was not successful, then the node frees the schedule and replies back with a failure message to its requestor.

For removing a path and its associated schedule and allocated bandwidth, a Path-remove request is used. In response to a Path-remove request, a node invokes the removePath procedure as given in Figure 5.10.

```
removePath(source, destination, reqFrames) {
    if !(isSlave(source) || isMaster(source)) {
        reply(FAIL);
        return;                    // don't control schedule for source
    }
    if (isSlave(destination)) {
        removeSchedule(reqFrames, schedID(source));
        if (success) {
            reply(OK);
            return;
        }
    } else {
        removeSchedule(reqFrames, schedID(source));
        if (success)
            reply(OK);
        else
            reply(FAIL);
        source = myAddress;
        masterNode = nextMasterInPath(destGW);
        sendPathRequest(masterNode,
                        removePath(source, destination, reqFrames));
        return;
    }
    reply(FAIL);
    return;
}
```

Figure 5.10 – removePath Pseudo Code

The removePath procedure is similar to the addPath procedure except that calls to the Scheduling system are to removeSchedule. First, the node checks to ensure it controls the scheduling for the source node. Then if the destination is on its sub-ring, the node

removes the schedule and replies back to its requestor with a successful reply message. If the destination is not on the sub-ring of this node, then a Path-remove request is sent to the next master in the path.

There are three distinct signaling scenarios:

1. **PATH Request on Single Ring**
   the source node is on the same ring as the destination node.
2. **PATH Request from Sub-ring → Super-ring**
   the source node is on a sub-ring of the ring with the destination node.
3. **PATH Request from Super-ring → Sub-ring**
   the source node is on a super-ring of the ring with the destination node.

Combining these scenarios gives a generalization for any multi-ring signaling needed in an HRN.

Example – Generalized Multi-Ring Signaling

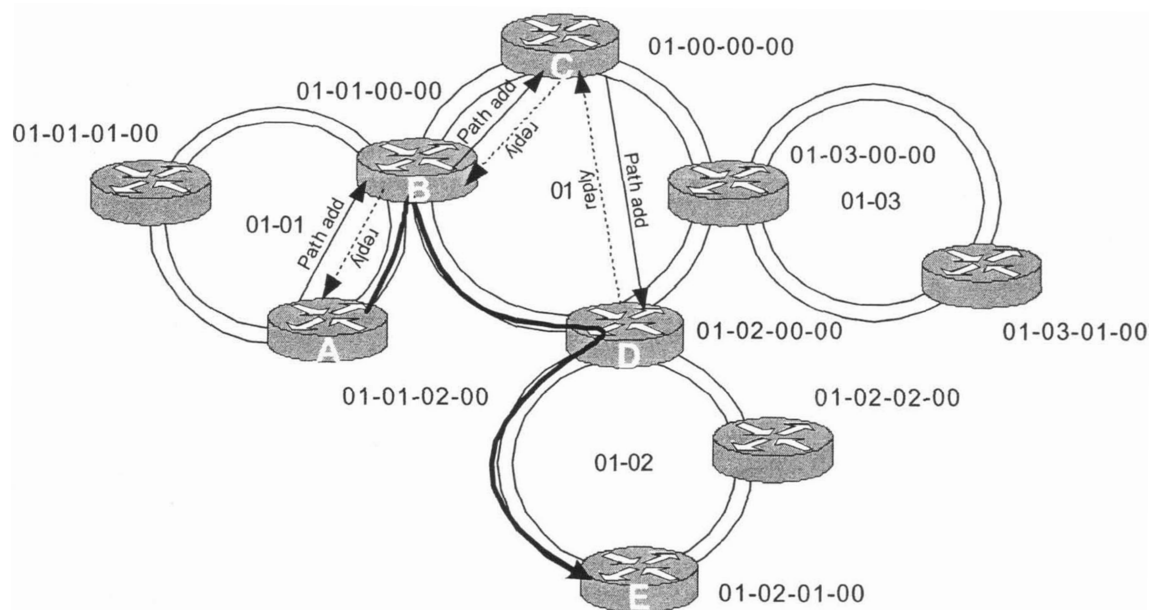

Figure 5.11 – Network Layout: Generalized Multi-Ring Signaling

Suppose we had an HRN as shown in Figure 5.11 and we wish to setup a path (to carry a DS0) between node A and node E as shown. The signaling would be as follows: node A (01-01-02-00) would send a Path-add request to node B (01-01-00-00) for one frame slot. This node would see that the source is on its sub-ring and the destination is not, so it would reserve one frame slot with Schedule ID 02 and send a Path-add request to node

C (01-00-00-00). This Path-add request would have 01-01-00-00 as the source. Node C would see that the source is on its sub-ring but the destination is not. It would in turn reserve one frame slot with Schedule ID 01 and send a Path-add request to node D (01-02-00-00). Node D notices that it is the last master in the path so it schedules the one frame slot with Schedule ID 00 and replies back with a positive PathID. Node C receives the successful reply, commits its schedule of one frame slot and replies to node B. In turn, node B also commits its schedule and replies back to node A. In this way, an end-to-end path is setup in the HRN.

5.2.3 SLA Processing

A mapping tool between external requests and internal requests performs *SLA Processing*. On each ingress node a SLA → Path process converts admission requests in the form of SLA requests into the corresponding Path requests. The pseudo code for the addSLA procedure is given in Figure 5.12.

```
slaID = 0;
addSLA(source, destination, reqBW) {
    srcGW = resolveGateway(source);
    destGW = resolveGateway(destination);
    reqFrames = ceiling(reqBW / schedGranularity);

    if (srcGW == destGW) {
        reply(FAIL);
        return;                    // don't control resources
    }
    if (srcGateway != myAddress) {
        reply(FAIL);
        return;                    // only make requests for self as gateway
    }
    masterNode = nextMasterInPath(destGW);
    sendPathRequest(masterNode, addPath(srcGW, destGW, reqFrames));
    if (success) {
        reply(++slaID);
        saveSLA(slaID, source, destination, reqBW, srcGW, destGW, reqFrames);
        return;
    }
    reply(FAIL);
    return;
}
```

Figure 5.12 – addSLA Pseudo Code

IP addresses are changed to HRN addresses using *Address Resolution Protocol* (ARP) [38] and HRN addresses are converted back into IP addresses using *Reverse Address Resolution Protocol* (RARP) [18]. Bandwidth is converted into the appropriate number

of frame slots by dividing the requested bandwidth by the IPFS Scheduling granularity (68.625Kbps).

The source and destination gateways are checked to make sure that the SLA request originates from a host connected to the ingress gateway and that the requested path is through the HRN. A Path-add request is created and sent to the next master in the path. Assuming the path was setup successfully, a reply containing the slaID is sent to the host and the SLA is saved in the local database. If the SLA request fails for any reason, a failure reply is sent back to the host.

The removeSLA procedure given in Figure 5.13 is similar, with the exception that the parameters for the removal request are retrieved from the local database using a provided slaID. A Path-remove request is created and the path is subsequently torn down and resources are freed.

```
removeSLA(slaID) {
    retrieveSLA(slaID, source, destination, reqBW, srcGW, destGW, reqFrames);
    if !(success) {
        reply(FAIL);
        return;                     // don't know about SLA
    }
    masterNode = nextMasterInPath(destGW);
    sendPathRequest(masterNode, removePath(srcGW, destGW, reqFrames));
    if (success) {
        reply(OK);
        return;
    }
    reply(FAIL);
    return;
}
```

Figure 5.13 – removeSLA Pseudo Code

Detailed examples showing how SLAs are converted, paths are setup, and bandwidth is allocated are provided in Appendix F.

5.3 Design

The Scheduling system, the Signaling system, and SLA Processing are the design elements used in our distributed SLA Admission Controller. First, our admission controller has a process running on the ingress gateways to accept SLA requests from customers. Second, our admission controller has a process running on the master nodes

to accept Path requests from other nodes and perform the necessary scheduling. Both of these processes listen on a well-known port and accept TCP/IP connections. A high-level state chart for both of these processes is given in Figure 5.14. These processes respond to both add and remove requests. The add requests are used for setting up an end-to-end path through the HRN, and the remove requests are used to free up the allocated resources once the customer has finished with the stream. SLA requests originate with the customer and are mapped to Path requests at the ingress gateway.

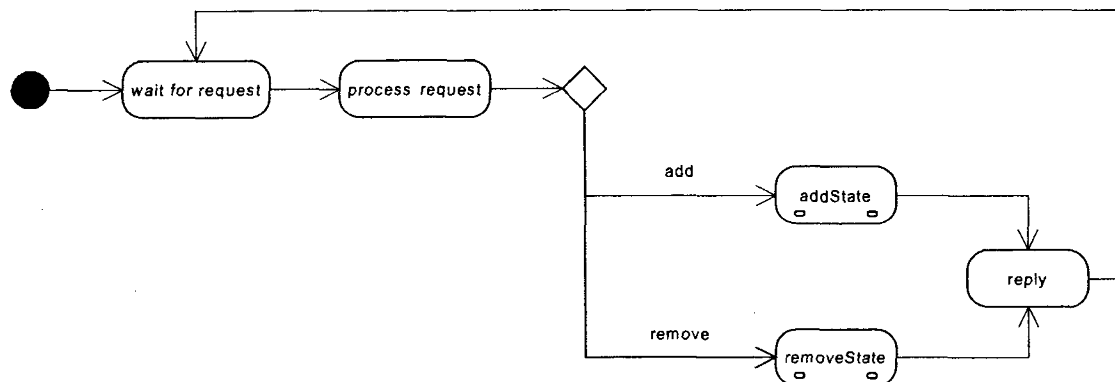

Figure 5.14 – SLA/Path Request Statechart

5.3.1 slad Daemon

A daemon called slad, runs on the ingress gateway and processes admission requests. A host connects to its gateway slad daemon and sends an SLA request for admission. The slad daemon parses the SLA request and resolves the source and destination IP addresses into ingress and egress gateways, which it then resolves into associated HRN addresses. The slad daemon also converts the bandwidth requirements for this request into the number of frames to be scheduled. A Path request is then constructed using the ingress and egress gateway HRN addresses and the number of frames needed; this is sent to the next master node in the path. Once the master nodes in the path have processed the Path request, slad receives either an ACK (success) or a NACK (failure) reply. If the reply is an ACK, then slad stores the SLA in its database and sends an ACK reply to the host. This ACK contains the slaID. If the reply is a NACK, then slad sends a NACK reply to the host. The state chart for the addSLA process is given in Figure 5.15 and the state chart for removeSLA is given in Figure 5.16.

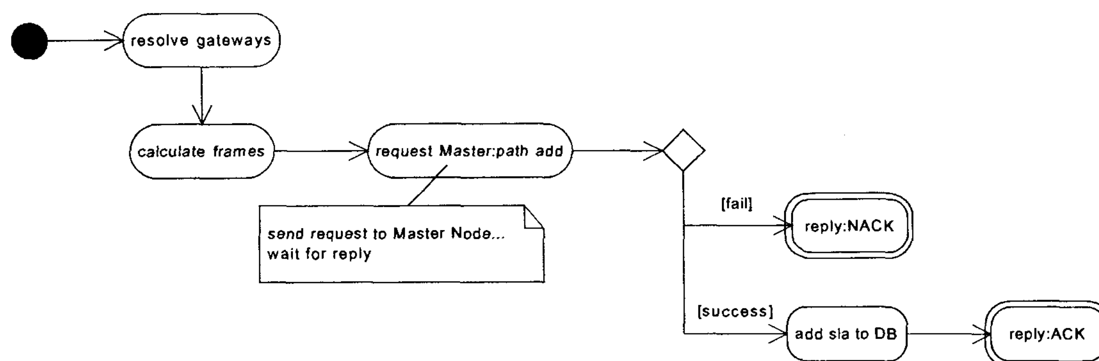

Figure 5.15 – addSLA Statechart

The SLA remove request procedure is similar to the add request procedure, however the SLA is retrieved from the database instead of being supplied by the requestor.

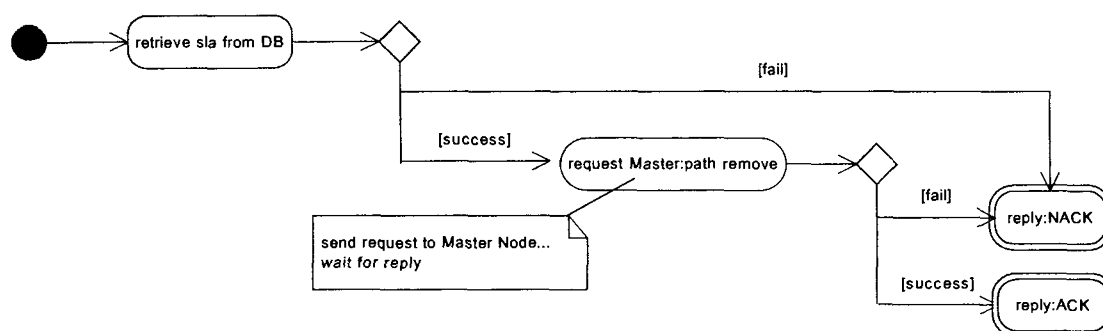

Figure 5.16 – removeSLA Statechart

5.3.2 pathd Daemon

The process running on the master nodes is a daemon called pathd; either a slad daemon or another pathd daemon connects to the pathd daemon and sends a Path request. The pathd daemon parses the Path request and determines if it controls all of the scheduling needed for the request. If so, then the pathd daemon can go ahead and schedule the frames using the Schedule ID it calculates for the source gateway. If the scheduling is successful then pathd can send an ACK to the requestor, else it sends a NACK. If the pathd daemon does not control all of the scheduling necessary for the Path request, then it first tries to reserve a schedule using the number of frames and the Schedule ID it calculates for the source gateway. If this reservation is successful then pathd can, in turn, make a modified Path request to the next master in the path. Once all of the pathd daemons running on the master nodes in the path have processed their modified Path requests, then this pathd daemon will receive either an ACK or a NACK reply. If the

reply is an ACK, then pathd commits the schedule it had reserved and sends an ACK reply to the requestor. If the reply is a NACK, then pathd sends a NACK reply to the requestor. The state chart for the addPATH process is given in Figure 5.17.

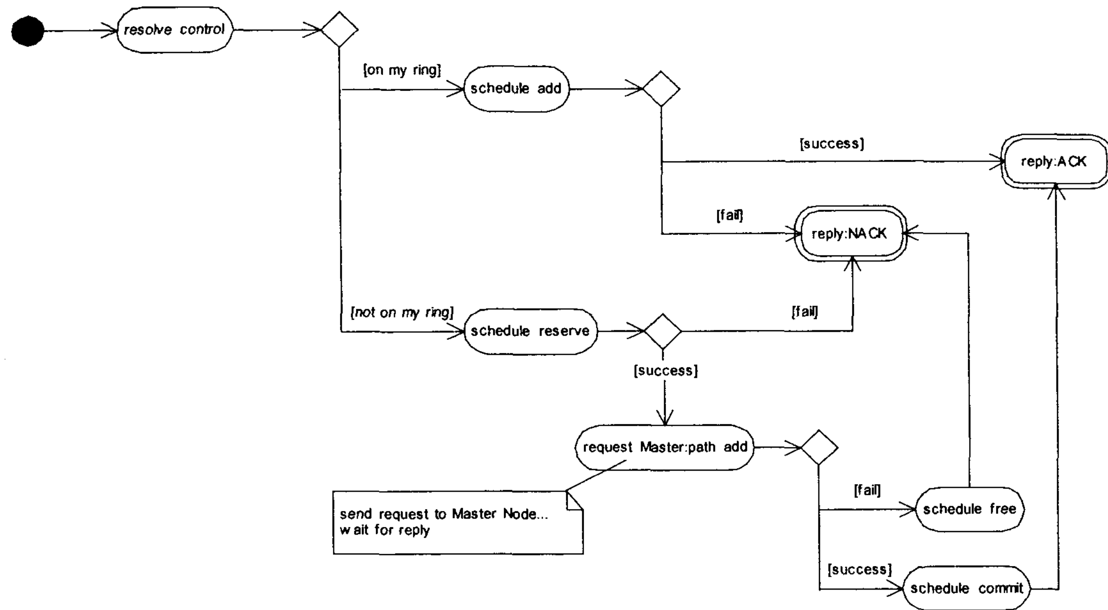

Figure 5.17 – addPATH Statechart

For a Path-remove request, pathd follows a similar procedure – freeing up any scheduling it has performed for the original Path-add request and requesting removal from subsequent pathds. The state chart for removePATH is given in Figure 5.18.

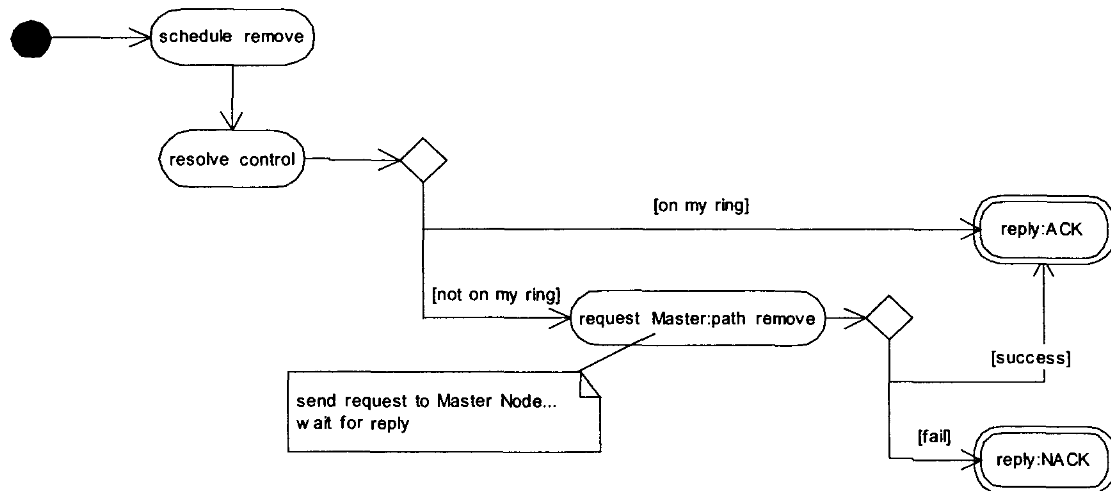

Figure 5.18 – removePATH Statechart

5.3.3 Interfaces & Considerations

There are two interfaces that are important for these two daemons. The slad daemon interfaces to the routing daemon for gateway and IP ↔ HRN resolution, and the pathd daemon interfaces to the local Scheduler. This latter interface directly manipulates the Schedule table in the FPGA.

The combination of all of the slad daemons running on ingress nodes and all of the pathd daemons running on master nodes and all of their inter-process signaling makes up our admission controller – dSLACtl. This design of multiple daemons interacting through TCP/IP implies an application layer implementation, which is what we have developed. Each individual slad and pathd daemon has an IP address (that of the node itself) and a port that it listens to, waiting for requests. If a particular request – either SLA or Path request – requires network resources that are not controlled by the daemon, then it makes a request to the next master in the path. The UML Sequence chart given in Figure 5.19 illustrates how these daemons interact to process an SLA admission request.

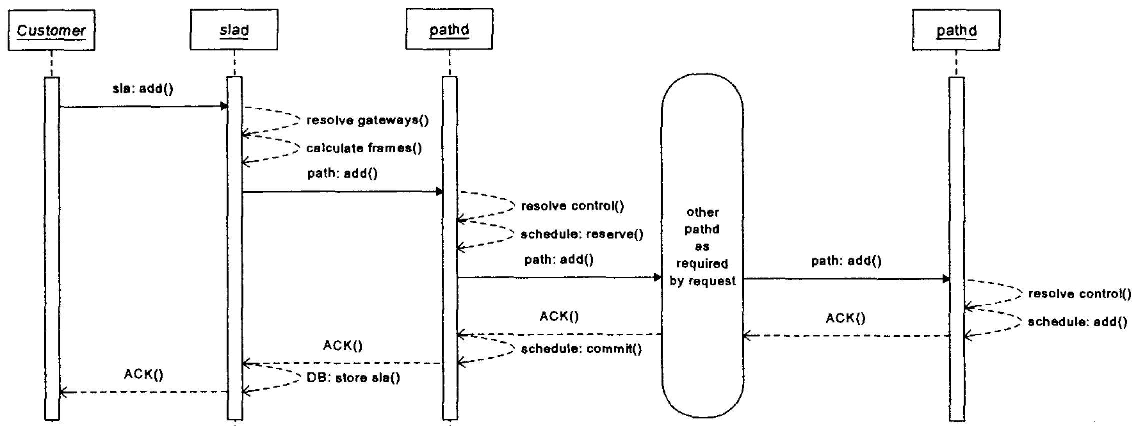

Figure 5.19 – dSLACtl Sequence Chart

There are two important considerations in our design to account for streams that traverse multiple rings. The first is that requests are modified as they traverse the hierarchy to represent the entity that is making the request – for example, a master node changes the source gateway from 00-FA-01-01-01-04 to its own HRN address 00-FA-01-01-01-00; this provides for Schedule aggregation. The second is the addition of reserve, commit, and free to the Scheduler for the two-phase commit protocol.

5.4 Admission Process

Given this design for dSLACtl, we now abstract this into the Admission Process as illustrated in Figure 5.20. A customer wishing to be granted guaranteed access to IPFSNET first determines its QoS needs – step 1. From these needs, the customer creates an SLA and sends it to the local ingress gateway – step 2. dSLACtl and its subsequent daemons, slad and pathd process this SLA request. During this process, dSLACtl tries to schedule the resources required by the request – step 3. If the request fails at any point then a NACK is returned and the process ends. If all of the scheduling is successful, then dSLACtl sends an ACK to inform the customer that its SLA request has been granted – step 4. The customer can then start sending data on this path – step 5.

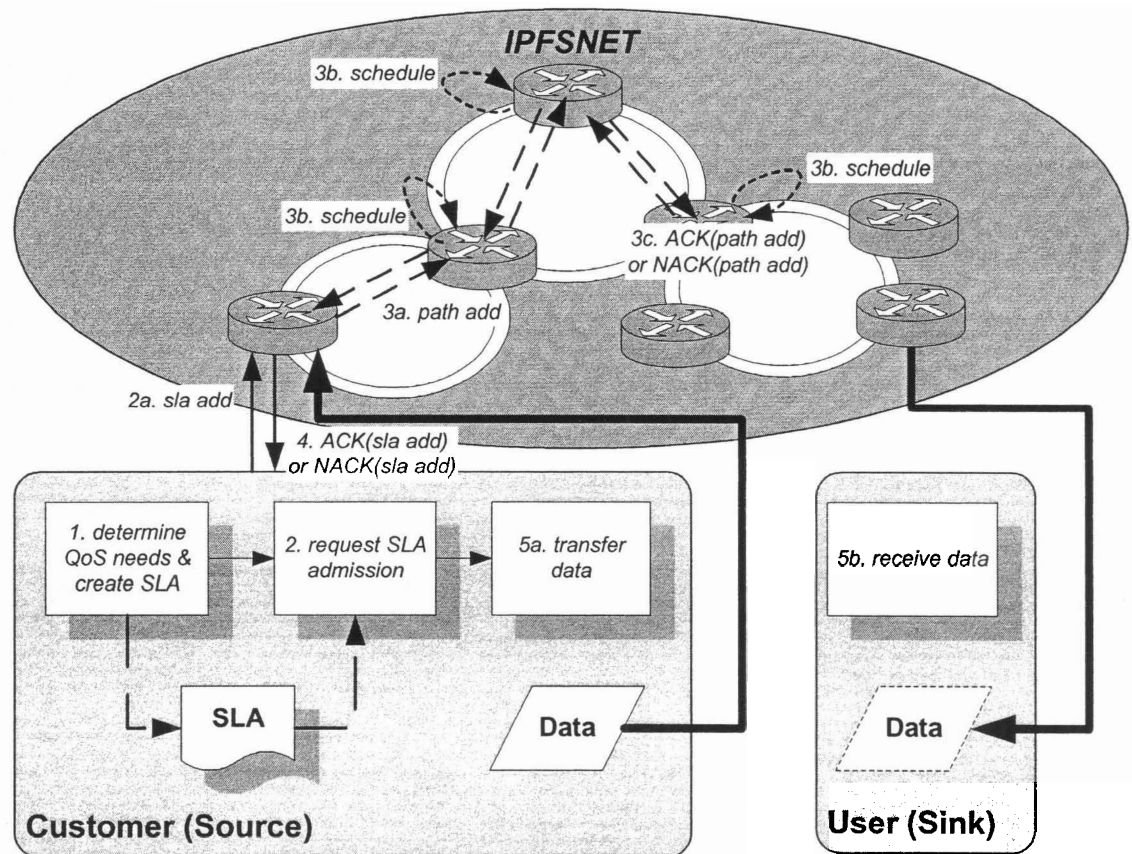

Figure 5.20 – IPFSNET Admission Process

The customer interface to dSLACtl is identical to our original design for SLACtl. However, with dSLACtl the actual processing of the SLA request is distributed throughout the HRN, instead of offline.

This design for dSLACtl decides if an SLA request can be granted (or not) based solely on available resources – if the resources in the path are free enough to grant the request, then it is granted. SLA optimization is not included at this point. However, dSLACtl has been designed with an internal interface so that 'plug-in' modules, such as a Bandwidth Broker or a Policy Manager, can be added. An SLA request can be re-directed to one or more of these plug-in modules prior to mapping into a Path request. If the plug-in module decides to approve the SLA request, then it can either create its own Path request or use the SLA processing system to setup the path. These 'plug-in' modules have been added to the architecture diagram in Figure 5.21

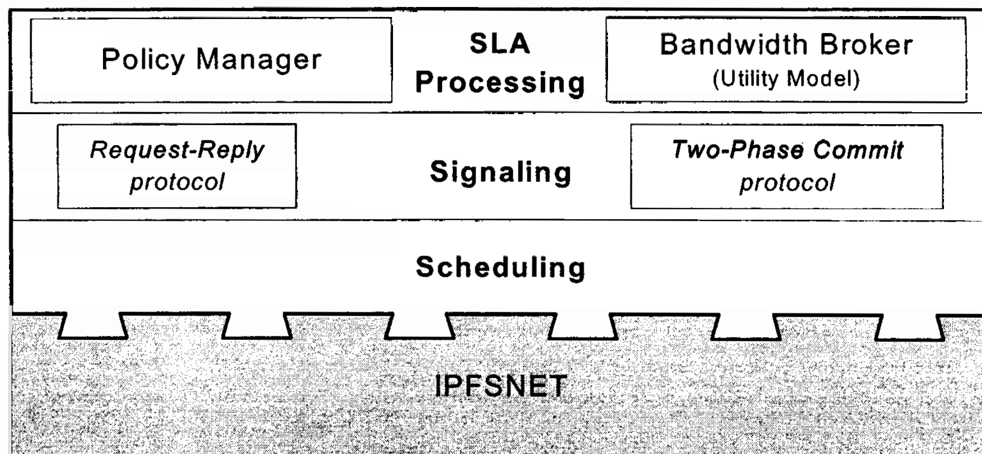

Figure 5.21 – dSLACtl Architecture with 'plug-in' Modules

5.5 Summary

Our *distributed SLA Admission Controller* is comprised of three systems – the Scheduling system, the Signaling system, and SLA Processing. A Scheduler runs on each node and performs the scheduling of frame slots for a given Path request. The Request-Reply protocol is used in signaling between master nodes and ingress nodes, and a variant of the two-phase commit protocol is used to reserve, commit, or free frame slots in each sub-ring. SLA Processing maps between customer SLA requests and Path requests. Two daemons, slad and pathd, run on ingress and master nodes and perform all required work for an SLA/Path request, including determining where subsequent Path requests should be sent. These components are distributed throughout the HRN. With the use of schedule aggregation, dSLACtl scales well as the HRN grows.

IPFSNET solves the problem of guaranteeing QoS in an IP network. Support for our admission controller is integral to the IPFS technology at the hardware and software levels. We can allocate bandwidth in a highly dynamic fashion through the use of automated SLA processing, as opposed to current ad hoc methods and those used in off-line provisioning, such as MPLS and ATM technologies, which are only suitable for static large-grained provisioning. With our solution, there is no special provisioning mode for IPFS nodes; instead, all admission requests are handled by dSLACtl concurrent with network operation. Frame slot scheduling is done by simply setting a value in a hardware look up table; as soon as a frame slot has been scheduled, the node can immediately (< 30 msec) start sending ERP frames in it.

The algorithms used in the IPFS nodes guarantee hard QoS and provide for soft QoS (or relative precedence), similar to DiffServ's assured and expedited forwarding modes. But in our solution, traffic streams do not need to be mapped to a limited number of service classes; a stream can have its own dedicated path and network resources. The small scheduling granularity (≅ 69Kbps) of our solution means that customer requests for bandwidth can be efficiently mapped to corresponding IPFS schedules with minimal waste. Our admission controller scales well due to schedule aggregation and distribution of the Scheduling system, as opposed to IntServ, which suffers from a lack of scalability. With schedule aggregation, all requests that need to traverse a gateway into another ring are aggregated together and treated as a single request. Since a master node is only responsible for scheduling the nodes on its direct sub-ring, our admission controller is distributed across the HRN. Plug-in interfaces allow dSLACtl to include other admission decision-making processes, such as a Bandwidth Broker or a Policy Manager, making dSLACtl extendable.

With its low-level hardware and software support, our *distributed SLA Admission Controller* provides for automated highly dynamic fine-grained SLA requests for service – a customer can request bandwidth for only the amount they need, the duration they need, and at the time they need.

Chapter 6
Conclusions & Further Work

In this chapter we provide a synopsis of our research, summarize our contributions, and briefly discuss ongoing work.

6.1 Synopsis

We started our research by looking for ways to automate the current manual SLA process and also include support for highly dynamic fine-grained SLAs. The success in applying the Utility Model to a simulated network prompted us to extend this work by building an automated SLA Admission Controller and integrating it into a real IP network based on Nortel Passport 7440 routers. It became apparent that we would need fixed-path routing, so we chose ER-MPLS. Our analysis of the SLAOpt simulator showed that we could modify it and integrate it into our *SLA Admission Controller* – SLACtl. We then commissioned and provisioned a prototype MPLS network – QoSNET. When QoSNET became functional, we realized that the current implementation of MPLS was intended for traffic engineering only – it is not as agile or robust as we need for our work.

We then looked at a new SONET-based technology, IPFS, which allowed us to build in the support for QoS that we needed. Our discussion of the IPFS technology and its associated HRN and ERP protocols led to a discussion of QoS support through the use of priorities and, more importantly, a scheduling system. We developed this scheduling

system in this thesis and extended the discussion to explain the necessary signaling between nodes to set up a path through an HRN. This became the basis that we needed for our *distributed SLA Admission Controller* – dSLACtl. We expanded this signaling into the design of dSLACtl and showed that dSLACtl uses the same customer-interface as SLACtl did. However, instead of being external to the network, as was SLACtl, dSLACtl is tightly integrated into the network so that it can take advantage of the IPFS technology and the specific QoS support that we built in to support our work. This second solution solves the problem of guaranteeing Quality of Service in an IP network.

6.2 Main Contributions

- Developed an extendable, scaleable, and distributed SLA Admission Controller that can guarantee quality of service using automated SLA processing.
- Designed a rapid frame slot Scheduling system that provides fine-grained allocation of IPFS network resources.
- Implemented the infrastructure needed to support dynamic SLA admission requests.
- Integrated Soft QoS and Hard QoS services into new IPFS technology, which adds layer-2 switching to a SONET backbone.

6.3 Further Work

In this work we provide the entire infrastructure for UM-style optimization of IPFS network resources – SLA processing, end-to-end path building, bandwidth allocation, and an interface to these processes – but we have not yet connected a MMKP optimizer to the interface; this is ongoing work. A part of what the Utility Model offers – specified path routing through the network – is presently not necessary in the IPFS technology; there is only one path for each end-to-end data stream; this is inherent in the HRN structure. When multiple-path routing is provided in IPFS networks we will need a fixed-path routing algorithm and can then take advantage of this feature of the Utility Model. The integration of the Utility Model into the IPFS technology should be in the form of a Bandwidth Broker that auctions bandwidth to customers; this can be either a central utility or a distributed system.

Interoperability is very important for commercial products – any new product will be accepted better by industry if it can work with other products seamlessly. For this reason, it is beneficial to use existing industry-accepted interfaces, rather than build proprietary interfaces. The signaling we developed in this work is very similar to CR-LDP and RSVP – setting up paths and reserving resources. Since the IEEE endorsed RSVP signaling for MPLS, we are integrating RSVP into our technology – porting RSVP to the QNX Neutrino RTOS and modifying it to interface with our scheduling system and *distributed SLA Admission Controller*.

REFERENCES

[1] Akbar, M. M. *The Distributed Utility Model Applied to Optimal Admission Control & QoS Adaptation in Multimedia Systems & Enterprise Networks*. PhD Dissertation, Department of Computer Science, University of Victoria. 2002.

[2] Almquist, P. "Type of Service in the Internet Protocol Suite." RFC1349 July 1992.

[3] Andersson, L., G. Swallow. "The Multiprotocol Label Switching (MPLS) Working Group decision on MPLS signaling protocols." RFC3468 February 2003.

[4] Andersson, L., P. Doolan, N. Feldman, A. Fredette, B. Thomas. "LDP Specification." RFC3036 January 2001.

[5] ATM Forum, www.atmforum.com

[6] ATM Forum. *ATM User-Network Interface Specification V3.0*. September 1993.

[7] Awduche, D., L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow. "RSVP-TE: Extensions to RSVP for LSP Tunnels." RFC3209 December 2001.

[8] Baker, F., C. Iturralde, F. Le Faucheur, B. Davie. "Aggregation of RSVP for IPv4 and IPv6 Reservations." RFC3175 September 2001.

[9] Bellcore GR-253-CORE. *Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria*. Issue 2, Revision 1. December 1997.

[10] Bernet, Y., P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, E. Felstaine. "A Framework for Integrated Services Operation over Diffserv Networks." RFC2998 November 2000.

[11] Blake, S., D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. "An Architecture for Differentiated Services." RFC2475 December 1998.

[12] Boyle, J., V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, W.S. Lai. "Applicability Statement for Traffic Engineering with MPLS." RFC3346. August 2002.

[13] Braden, R., D. Clark, S. Shenker. "Integrated Services in the Internet Architecture: an Overview." RFC1633 June 1994.

[14] Braden, R., L. Zhang, S. Berson, S. Herzog, and S. Jamin. "Resource reSerVation Protocol (RSVP) version 1, Functional Specification." RFC2205 September 1997.

[15] Coulouris, G., J. Dollimore, T. Kindberg. *Distributed Systems – Concepts and Design.* 2nd ed., Addison-Wesley, 1996. 108-112.

[16] Coulouris, G., J. Dollimore, T. Kindberg. *Distributed Systems – Concepts and Design.* 2nd ed., Addison-Wesley, 1996. 414-421

[17] Davie, B., Y. Rekter. *MPLS – Technology and Applications.* Morgan Kaufmann Publishers, 2000.

[18] Finlayson, R., T. Mann, J.C. Mogul, M. Theimer. "Reverse Address Resolution Protocol." STD0038 June 1984.

[19] Goralski, W. *SONET.* 2nd ed., McGraw-Hill, 2000.

[20] Heinanen, J., F. Baker, W. Weiss, J. Wroclawski. "Assured Forwarding PHB Group." RFC2597 June 1999.

[21] IEEE 802.3-2002. *Carrier Sense Multiple Access with Collision Detection CSMA/CD.* 2002.

[22] International Engineering Consortium, www.iec.org

[23] Internet Engineering Task Force, www.ietf.org

[24] Jamoussi, B., Ed., L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, N. Feldman, A. Fredette, M. Girish, E. Gray, J. Heinanen, T. Kilty, A. Malis. "Constraint-Based LSP Setup using LDP." RFC3212 January 2002.

[25] Khan, S. *Quality Adaptation in a Multi-Session Adaptive Multimedia System: Model and Architecture.* PhD Dissertation, Department of Electrical and Computer Engineering, University of Victoria. 1998.

[26] MacFaden, M., D. Partain, J. Saperia, W. Tackabury. "Configuring Networks and Devices with Simple Network Management Protocol (SNMP)." RFC3512 April 2003.

[27] Malkin, G. "RIP Version 2." STD0056 November 1998.

[28] Moy, J. "OSPF Version 2." STD0054 April 1998.

[29] Nichols, K., S. Blake, F. Baker and D. Black. "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers." RFC2474 December 1998.

[30] Nortel Networks. *Passport 7400, 15000 Commisioning Guide*, 241-5701-275 ver. 2.3S1. June 2001.

[31] Nortel Networks. *Passport 7400, 15000 Configuration Guide*, 241-5701-600 ver. 2.3S1. June 2001.

[32] Nortel Networks. *Passport 7400, 15000 Configuring IP*, 241-5701-810 ver. 2.3S1. June 2001.

[33] Nortel Networks. *Passport 7400, 15000 Documentation Guide*, 241-5701-001 ver. 2.3S1. June 2001.

[34] Nortel Networks. *Passport 7400, 15000 FP Configuration Reference*, 241-5701-615 ver. 2.3S1. June 2001.

[35] Nortel Networks. *Passport 7400, 15000 Multiprotocol Label Switching Guide*, 241-5701-445 ver. 3.1S1. November 2001.

[36] Nortel Networks. *Passport 7400, 15000 Trunking Guide*, 241-5701-420 ver. 2.3S1. June 2001.

[37] Nortel Networks. *Preside Multiservice Data Manager Documentation Guide*, 241-6001-000 ver. 13.2RSUP. September 2001.

[38] Plummer, D.C. "Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware." STD0037 November 1982.

[39] Postel, J. "Internet Protocol." STD0005 September 1981.

[40] Protocols.com, www.protocols.com

[41] QNX Software Systems, www.qnx.com

[42] Rekhter, Y., T. Li. "A Border Gateway Protocol 4 (BGP-4)." RFC1771 March 1995.

[43] Rosen, E., A. Viswanathan, R. Callon. "Multiprotocol Label Switching Architecture." RFC3031 January 2001.

[44] Shenker, S., C. Partridge, and R. Guerin. "Specification of Guaranteed Quality of Service." RFC2212 September 1997.

[45] Watson, R. K. *Applying the Utility Model to IP Networks: Optimal Admission & Upgrade of Service Level Agreements*. MASc Thesis, Department of Electrical and Computer Engineering, University of Victoria. April 2001.

[46] Wroclawski, J. "Specification of the Controlled-Load Network Element Service." RFC2211 September 1997.

[47] Wu, Y., J. Wang, L. Kung, W. Hwang, "Design of Reservation Mechanism Based on the Slotted Ring Network." International Conference on Parallel and Distributed Systems (ICPADS). 1998.

[48] Xiao, X., L. Ni. "Internet QoS: A Big Picture." IEEE Network 13.2 (March 1999): 8-18.

APPENDIX A

This is a basic configuration script for QoSNET0.

```
start prov
add Sw
set Sw avList !base_CC01S1F ip_CC01S1F mpls_CC01S1F atmNetworking_CC01S1F
wanDte_CC01S1F networking_CC01S1F,patchList !
add Sw Lpt/ATM
set Sw Lpt/ATM commentText "",featureList !,systemConfig default
add Sw Lpt/CP
set Sw Lpt/CP commentText "",featureList !oamenet ip bgp mvr,systemConfig default
add Sw Lpt/ENET_IP
set Sw Lpt/ENET_IP commentText "",featureList !ip atmmpe,systemConfig default
add Sw Lpt/MPLS_IP
set Sw Lpt/MPLS_IP commentText "",featureList !ip atmmpe mplscrldp atmiisp,systemConfig
default
add Shelf
set Shelf commentText ""
add Shelf Card/0
set Shelf Card/0 commentText "",cardType CP,sparingConnection notApplicable
add Shelf Card/0 Diag
add Shelf Card/0 Diag TrapData
add Shelf Card/0 Diag RecErr
add Shelf Card/1
set Shelf Card/1 commentText "",cardType 2pEth100BaseT,sparingConnection notApplicable
add Shelf Card/1 Diag
add Shelf Card/1 Diag TrapData
add Shelf Card/1 Diag RecErr
add Shelf Card/2
set Shelf Card/2 commentText "",cardType none,sparingConnection notApplicable
add Shelf Card/2 Diag
add Shelf Card/2 Diag TrapData
add Shelf Card/2 Diag RecErr
add Shelf Card/3
set Shelf Card/3 commentText "",cardType 2pOC3MmAtm2,sparingConnection notApplicable
add Shelf Card/3 Diag
add Shelf Card/3 Diag TrapData
add Shelf Card/3 Diag RecErr
add Shelf Card/4
set Shelf Card/4 commentText "",cardType none,sparingConnection notApplicable
add Shelf Card/4 Diag
add Shelf Card/4 Diag TrapData
add Shelf Card/4 Diag RecErr
add Shelf Test
set Shelf Test automaticBusClockTest disabled
add Lp/0
set Lp/0 mainCard Shelf Card/0,spareCard !,logicalProcessorType Sw Lpt/CP,customerIdentifier
0
add Lp/1
set Lp/1 mainCard Shelf Card/1,spareCard !,logicalProcessorType Sw
Lpt/ENET_IP,customerIdentifier 0
add Lp/3
set Lp/3 mainCard Shelf Card/3,spareCard !,logicalProcessorType Sw
Lpt/MPLS_IP,customerIdentifier 0
add Ac
```

```
set Ac publicKeyAuth ! ~telnet
#CAS SMS_DELIVERY
add Rtg
set Rtg splittingRegionIds !,clusterNode no,clusterFieldTrial off,tandemTraffic
allowed,defaultTraceLevel none
add Rtg Top
add Mod
set Mod nodeId 100,nodeName QOSNET0,namsId 5100,regionId 0,nodePrefix
47000001000000000000000000,alternatePorsPrefixes 0 ""  1 ""  2 ""  3 ""
#CAS GLOBAL_DELIVERY
add Lp/0 OamEnet/0
set Lp/0 OamEnet/0 customerIdentifier 0,ifAdminStatus up,ifIndex 1,applicationFramerName
La/0 Framer,switchoverOnFailure enabled,switchoverHoldoff 30,extendedStatistics
enabled,lineSpeed autoConfig,duplexMode autoConfig,vendor "",commentText "",defaultTraceLevel
none
add Lp/0 Eng
add Lp/1 Eng
add Lp/1 Eth100/0
set Lp/1 Eth100/0 customerIdentifier 0,ifAdminStatus up,ifIndex 3,applicationFramerName
La/10 Framer,duplexMode half,lineSpeed 100,autoNegotiation enabled,vendor "",commentText
"",defaultTraceLevel none
add Lp/1 Eth100/0 Lt
add Lp/1 Eth100/0 Test
add Lp/1 Eth100/1
set Lp/1 Eth100/1 customerIdentifier 0,ifAdminStatus up,ifIndex 4,applicationFramerName
La/11 Framer,duplexMode half,lineSpeed 100,autoNegotiation enabled,vendor "",commentText
"",defaultTraceLevel none
add Lp/1 Eth100/1 Lt
add Lp/1 Eth100/1 Test
add Lp/3 Sonet/0
set Lp/3 Sonet/0 lineAutomaticProtectionSwitch !,clockingSource local,txSectionTrace
"",expectedRxSectionTrace "",customerIdentifier 0,vendor "",commentText "",ifAdminStatus up
add Lp/3 Sonet/0 Path/0
set Lp/3 Sonet/0 Path/0 customerIdentifier 0,ifAdminStatus up,ifIndex 5
add Lp/3 Sonet/0 Path/0 Cell
set Lp/3 Sonet/0 Path/0 Cell alarmActDelay 500,scrambleCellPayload
on,correctSingleBitHeaderErrors off,lcdStateDelay 3,clearDelay 0,hecSampleInterval
0,hecErrorLimit 250000,hecErrorIntervalLimit 1
add Lp/3 Sonet/1
set Lp/3 Sonet/1 lineAutomaticProtectionSwitch !,clockingSource local,txSectionTrace
"",expectedRxSectionTrace "",customerIdentifier 0,vendor "",commentText "",ifAdminStatus up
add Lp/3 Sonet/1 Path/0
set Lp/3 Sonet/1 Path/0 customerIdentifier 0,ifAdminStatus up,ifIndex 7
add Lp/3 Sonet/1 Path/0 Cell
set Lp/3 Sonet/1 Path/0 Cell alarmActDelay 500,scrambleCellPayload
on,correctSingleBitHeaderErrors off,lcdStateDelay 3,clearDelay 0,hecSampleInterval
0,hecErrorLimit 250000,hecErrorIntervalLimit 1
add Lp/3 Eng
add Col/accounting
set Col/accounting collectionTimes !,peakWaterMarkInterval !
add Col/accounting Sp
set Col/accounting Sp spooling on,maximumNumberOfFiles 200
add Col/alarm
set Col/alarm collectionTimes !,peakWaterMarkInterval !
add Col/alarm Sp
set Col/alarm Sp spooling on,maximumNumberOfFiles 30
add Col/log
set Col/log collectionTimes !,peakWaterMarkInterval !
add Col/log Sp
```

```
set Col/log Sp spooling on,maximumNumberOfFiles 10
add Col/debug
set Col/debug collectionTimes !,peakWaterMarkInterval !
add Col/debug Sp
set Col/debug Sp spooling off,maximumNumberOfFiles 2
add Col/scn
set Col/scn collectionTimes !,peakWaterMarkInterval !
add Col/scn Sp
set Col/scn Sp spooling on,maximumNumberOfFiles 10
add Col/trap
set Col/trap collectionTimes !,peakWaterMarkInterval !
add Col/trap Sp
set Col/trap Sp spooling off,maximumNumberOfFiles 2
add Col/stats
set Col/stats collectionTimes !,peakWaterMarkInterval !
add Col/stats Sp
set Col/stats Sp spooling on,maximumNumberOfFiles 200
add Col/rtStats
set Col/rtStats collectionTimes !,peakWaterMarkInterval !
add Col/rtStats Sp
set Col/rtStats Sp spooling off,maximumNumberOfFiles 2
add Time
#CAS ACTIVATE_DELIVERY
#CAS VIRTUAL_PROCESS_DELIVERY
add Vr/0
set Vr/0 snmpAdminStatus up,managementAccess enabled,virtualPrivateIntranetIdentifier
1,vpnMode customer,virtualPrivateNetworkIdentifier 00-00-00-00-00-00-
00,virtualRouterIdentifier 1,customerIdentifier 0,defaultTraceLevel none
add Vr/0 QosP/0
add Vr/0 Mm
set Vr/0 Mm vrMaxHeapSpace 100
add Vr/0 Pp/OAMENET0
set Vr/0 Pp/OAMENET0 snmpAdminStatus up,linkToMedia La/0,protocolPortId
1,accountingControl sameAsProtocol
add Vr/0 Pp/OAMENET0 IpPort
set Vr/0 Pp/OAMENET0 IpPort cosPolicyAssignment !,filterAssignment !,adminStatus
enabled,maxTxUnit 0,arpStatus auto,proxyArpStatus disabled,arpNoLearn disabled,sendRedirect
enabled,multicastStatus auto,relayAddress 0.0.0.0,relayBroadcast disabled,directedBroadcast
enabled,linkModel localAreaNetwork,lanModel localAreaNetwork,encap auto,icmpMaskReply
enabled,assignedQos 0,allowMcastMacDest disabled,sourceRouteEndStationSupport
disable,snmpAdminStatus up
add Vr/0 Pp/OAMENET0 IpPort LogicalIf/10.4.0.10
set Vr/0 Pp/OAMENET0 IpPort LogicalIf/10.4.0.10 netMask 255.255.255.0,broadcastAddress
10.4.0.255,linkDestinationAddress 0.0.0.0
add Vr/0 Pp/OAMENET0 IpPort LogicalIf/10.4.0.10 RipIf
set Vr/0 Pp/OAMENET0 IpPort LogicalIf/10.4.0.10 RipIf snmpAdminStatus up,metric
1,poisonReverseFlag false,flashUpdateFlag true,networkRouteStatus
naturalNetWithOutDefaultRoute,defaultRouteMetric 1,acceptDefault true,ifConfSend
rip1Compatible,ifConfReceive rip1OrRip2
add Vr/0 Ip
set Vr/0 Ip cosPolicyAssignment !,filterAssignment !,adminStatus enabled,forwarding
gateway,sourceRoute disabled,defaultTtl 255,cacheTableSize 0 100  1 3000  2 3000  3 3000  4
3000  5 3000  6 3000  7 3000  8 3000  9 3000  10 3000  11 3000  12 3000  13 3000  14 3000
15 3000,accountCollection ! ~bill ~test ~study ~audit ~force,snmpAdminStatus
up,defaultTraceLevel none
add Vr/0 Ip Icmp
set Vr/0 Ip Icmp defaultTraceLevel none
add Vr/0 Ip Udp
add Vr/0 Ip Arp
```

```
set Vr/0 Ip Arp autoRefresh disabled,autoRefreshTimeout 5,defaultTraceLevel none
add Vr/0 Ip Tcp
add Vr/0 lp RelayBC
set Vr/0 lp RelayBC relayStatus disabled,relayNdStatus disabled
add Vr/0 lp Rip
set Vr/0 Ip Rip snmpAdminStatus up,adminStatus enabled,migrateRip
disabled,rfc1058MetricUsage enabled,generateDiscardRoute yes,redistributelbgp
false,defaultRipRtePreference 82,defaultTraceLevel none
add Vr/LER
set Vr/LER snmpAdminStatus up,managementAccess disabled,virtualPrivateIntranetIdentifier
2,vpnMode carrier,virtualPrivateNetworkIdentifier 00-00-00-00-00-00,virtualRouterIdentifier
2,customerIdentifier 0,defaultTraceLevel none
add Vr/LER QosP/0
add Vr/LER Mm
set Vr/LER Mm vrMaxHeapSpace 100
add Vr/LER Pp/LA10
set Vr/LER Pp/LA10 snmpAdminStatus up,linkToMedia La/10,protocolPortId
8,accountingControl sameAsProtocol
add Vr/LER Pp/LA10 lpPort
set Vr/LER Pp/LA10 lpPort cosPolicyAssignment !,filterAssignment !,adminStatus
enabled,maxTxUnit 0,arpStatus auto,proxyArpStatus disabled,arpNoLearn disabled,sendRedirect
enabled,multicastStatus auto,relayAddress 0.0.0.0,relayBroadcast disabled,directedBroadcast
enabled,linkModel localAreaNetwork,lanModel localAreaNetwork,encap auto,icmpMaskReply
enabled,assignedQos 0,allowMcastMacDest disabled,sourceRouteEndStationSupport
disable,snmpAdminStatus up
add Vr/LER Pp/LA10 lpPort LogicalIf/10.4.1.1
set Vr/LER Pp/LA10 IpPort LogicalIf/10.4.1.1 netMask 255.255.255.0,broadcastAddress
0.0.0.0,linkDestinationAddress 0.0.0.0
add Vr/LER Pp/LA11
set Vr/LER Pp/LA11 snmpAdminStatus up,linkToMedia La/11,protocolPortId
9,accountingControl sameAsProtocol
add Vr/LER Pp/LA11 IpPort
set Vr/LER Pp/LA11 IpPort cosPolicyAssignment !,filterAssignment !,adminStatus
enabled,maxTxUnit 0,arpStatus auto,proxyArpStatus disabled,arpNoLearn disabled,sendRedirect
enabled,multicastStatus auto,relayAddress 0.0.0.0,relayBroadcast disabled,directedBroadcast
enabled,linkModel localAreaNetwork,lanModel localAreaNetwork,encap auto,icmpMaskReply
enabled,assignedQos 0,allowMcastMacDest disabled,sourceRouteEndStationSupport
disable,snmpAdminStatus up
add Vr/LER Pp/LA11 IpPort LogicalIf/10.4.2.1
set Vr/LER Pp/LA11 IpPort LogicalIf/10.4.2.1 netMask 255.255.255.0,broadcastAddress
0.0.0.0,linkDestinationAddress 0.0.0.0
add Vr/LER Pp/LER1
set Vr/LER Pp/LER1 snmpAdminStatus up,linkToMedia AtmMpe/30,protocolPortId
2,accountingControl sameAsProtocol
add Vr/LER Pp/LER1 IpPort
set Vr/LER Pp/LER1 IpPort cosPolicyAssignment !,filterAssignment !,adminStatus
enabled,maxTxUnit 0,arpStatus auto,proxyArpStatus disabled,arpNoLearn disabled,sendRedirect
enabled,multicastStatus auto,relayAddress 0.0.0.0,relayBroadcast disabled,directedBroadcast
enabled,linkModel localAreaNetwork,lanModel localAreaNetwork,encap auto,icmpMaskReply
enabled,assignedQos 0,allowMcastMacDest disabled,sourceRouteEndStationSupport
disable,snmpAdminStatus up
add Vr/LER Pp/LER1 IpPort LogicalIf/10.5.1.1
set Vr/LER Pp/LER1 IpPort LogicalIf/10.5.1.1 netMask 255.255.255.252,broadcastAddress
10.5.1.255,linkDestinationAddress 0.0.0.0
add Vr/LER Pp/LER1 IpPort LogicalIf/10.5.1.1 OspfIf
set Vr/LER Pp/LER1 IpPort LogicalIf/10.5.1.1 OspfIf areaId 0.0.0.0,ifType
nbma,snmpAdminStatus up,rtrPriority 1,transitDelay 1,retransInterval 5,helloInterval
1,rtrDeadInterval 5,pollInterval 15,authKey 0000000000000000,status valid,multicastForwarding
blocked,pointToMultiPointMode nonBroadcast
```

```
add Vr/LER Pp/LER1 IpPort LogicalIf/10.5.1.1 OspfIf Nbr/10.5.1.2
set Vr/LER Pp/LER1 IpPort LogicalIf/10.5.1.1 OspfIf Nbr/10.5.1.2 priority 1,status valid
add Vr/LER Pp/LER1 MplsPort
set Vr/LER Pp/LER1 MplsPort elspCacScalingFactor 100
add Vr/LER Pp/LER1 MplsPort LdpIf
set Vr/LER Pp/LER1 MplsPort LdpIf mediaName AtmIf/30 Vcc/0.32 Nep,ldpInterworking
false,labelType atmLabel,atmLabelMinIn 0.100,atmLabelMaxIn 0.200,atmLabelMinOut
0.100,atmLabelMaxOut 0.200
add Vr/LER Pp/LER2
set Vr/LER Pp/LER2 snmpAdminStatus up,linkToMedia AtmMpe/31,protocolPortId
3,accountingControl sameAsProtocol
add Vr/LER Pp/LER2 IpPort
set Vr/LER Pp/LER2 IpPort cosPolicyAssignment !,filterAssignment !,adminStatus
enabled,maxTxUnit 0,arpStatus auto,proxyArpStatus disabled,arpNoLearn disabled,sendRedirect
enabled,multicastStatus auto,relayAddress 0.0.0.0,relayBroadcast disabled,directedBroadcast
enabled,linkModel localAreaNetwork,lanModel localAreaNetwork,encap auto,icmpMaskReply
enabled,assignedQos 0,allowMcastMacDest disabled,sourceRouteEndStationSupport
disable,snmpAdminStatus up
add Vr/LER Pp/LER2 IpPort LogicalIf/10.5.2.2
set Vr/LER Pp/LER2 IpPort LogicalIf/10.5.2.2 netMask 255.255.255.252,broadcastAddress
10.5.1.255,linkDestinationAddress 0.0.0.0
add Vr/LER Pp/LER2 IpPort LogicalIf/10.5.2.2 OspfIf
set Vr/LER Pp/LER2 IpPort LogicalIf/10.5.2.2 OspfIf areaId 0.0.0.0,ifType
nbma,snmpAdminStatus up,rtrPriority 1,transitDelay 1,retransInterval 5,helloInterval
1,rtrDeadInterval 5,pollInterval 15,authKey 0000000000000000,status valid,multicastForwarding
blocked,pointToMultiPointMode nonBroadcast
add Vr/LER Pp/LER2 IpPort LogicalIf/10.5.2.2 OspfIf Nbr/10.5.2.1
set Vr/LER Pp/LER2 IpPort LogicalIf/10.5.2.2 OspfIf Nbr/10.5.2.1 priority 1,status valid
add Vr/LER Pp/LER2 MplsPort
set Vr/LER Pp/LER2 MplsPort elspCacScalingFactor 100
add Vr/LER Pp/LER2 MplsPort LdpIf
set Vr/LER Pp/LER2 MplsPort LdpIf mediaName AtmIf/31 Vcc/0.32 Nep,ldpInterworking
false,labelType atmLabel,atmLabelMinIn 0.100,atmLabelMaxIn 0.200,atmLabelMinOut
0.100,atmLabelMaxOut 0.200
add Vr/LER Pp/VM01IF0
set Vr/LER Pp/VM01IF0 snmpAdminStatus up,linkToMedia Vm/1 If/0,protocolPortId
6,accountingControl sameAsProtocol
add Vr/LER Pp/VM01IF0 IpPort
set Vr/LER Pp/VM01IF0 IpPort cosPolicyAssignment !,filterAssignment !,adminStatus
enabled,maxTxUnit 0,arpStatus auto,proxyArpStatus disabled,arpNoLearn disabled,sendRedirect
enabled,multicastStatus auto,relayAddress 0.0.0.0,relayBroadcast disabled,directedBroadcast
enabled,linkModel pointToPoint,lanModel localAreaNetwork,encap auto,icmpMaskReply
enabled,assignedQos 0,allowMcastMacDest disabled,sourceRouteEndStationSupport
disable,snmpAdminStatus up
add Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.1
set Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.1 netMask 255.255.255.255,broadcastAddress
0.0.0.0,linkDestinationAddress 0.0.0.0
add Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.1 OspfIf
set Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.1 OspfIf areaId 0.0.0.0,ifType
passive,snmpAdminStatus up,rtrPriority 1,transitDelay 1,retransInterval 5,helloInterval
10,rtrDeadInterval 40,pollInterval 120,authKey 0000000000000000,status
valid,multicastForwarding blocked,pointToMultiPointMode nonBroadcast
add Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.1 OspfIf TOS/0
set Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.1 OspfIf TOS/0 tosMetric 0,metricStatus valid
add Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.2
set Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.2 netMask 255.255.255.255,broadcastAddress
0.0.0.0,linkDestinationAddress 0.0.0.0
add Vr/LER Pp/VM01IF0 IpPort LogicalIf/10.5.10.2 OspfIf
```

APPENDIX E

Scheduling Frame Slot in IPFS

The three scheduling scenarios for setting up an end-to-end path through an HRN are:

- **Same Ring Scheduling**
  the source node is on the same ring as the destination node.
- **Sub-ring to Super-ring Scheduling**
  the source node is on a sub-ring of the ring with the destination node.
- **Super-ring to Sub-ring Scheduling**
  the source node is on a super-ring of the ring with the destination node.

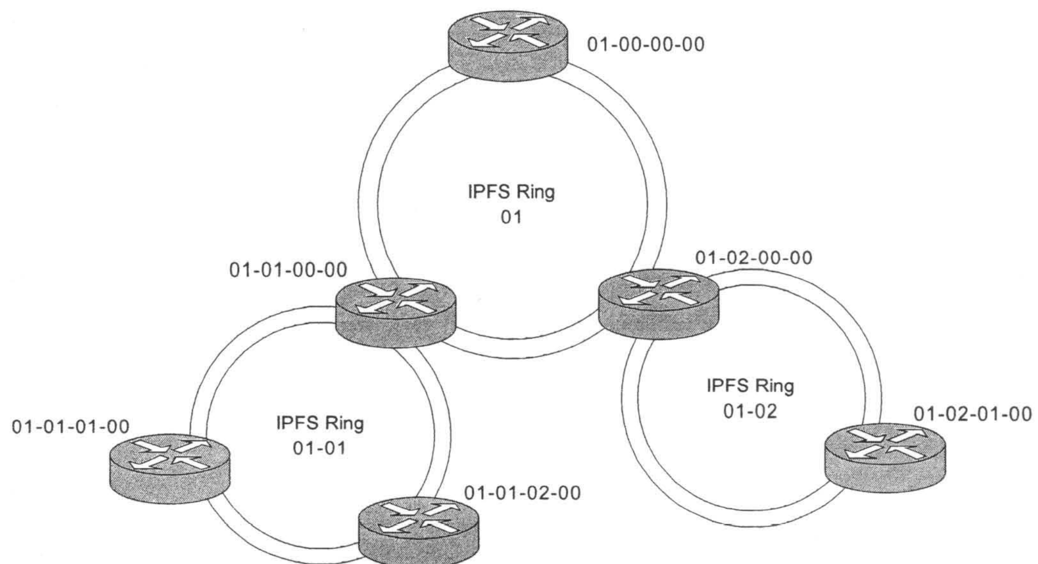

Figure A.2 – Network Layout: Multi-Ring Scheduling

To illustrate these scenarios, we use the example HRN as given in Figure A.2.

```
add Vr/LER Mpls LspG/1 Lsp/2 Msc
set Vr/LER Mpls LspG/1 Lsp/2 Msc standbyLsp !,serviceCategory 1
add Vr/LER Mpls LspG/1 Lsp/3
add Vr/LER Mpls LspG/1 Lsp/3 CrQos
set Vr/LER Mpls LspG/1 Lsp/3 CrQos peakDataRate 500000,peakBurstSize
500000,committedDataRate 500000,committedBurstSize 500000,frequency veryFrequent
add Vr/LER Mpls LspG/1 Lsp/3 Te
set Vr/LER Mpls LspG/1 Lsp/3 Te pathName Vr/LER Mpls Path/1,pathOnDemand
disabled,alarmControl none
add Vr/LER Mpls LspG/1 Lsp/3 Msc
set Vr/LER Mpls LspG/1 Lsp/3 Msc standbyLsp !,serviceCategory 2
add Vr/LER Mpls LspG/1 Lsp/4
add Vr/LER Mpls LspG/1 Lsp/4 CrQos
set Vr/LER Mpls LspG/1 Lsp/4 CrQos peakDataRate 1000000,peakBurstSize
1000000,committedDataRate 1000000,committedBurstSize 1000000,frequency veryFrequent
add Vr/LER Mpls LspG/1 Lsp/4 Te
set Vr/LER Mpls LspG/1 Lsp/4 Te pathName Vr/LER Mpls Path/1,pathOnDemand
disabled,alarmControl none
add Vr/LER Mpls LspG/1 Lsp/4 Msc
set Vr/LER Mpls LspG/1 Lsp/4 Msc standbyLsp !,serviceCategory 3
add Vr/LER Mpls LspG/1 IpFwd
add Vr/LER Mpls LspG/1 IpFwd StIpv4/10.4.11.0,24
set Vr/LER Mpls LspG/1 IpFwd StIpv4/10.4.11.0,24 bestMatchOverride disabled
add Vr/LER Mpls LspG/1 IpFwd MscMap
set Vr/LER Mpls LspG/1 IpFwd MscMap cosToMsc 0 0  1 1  2 2  3 3
add Vr/LER Mpls Ldp
set Vr/LER Mpls Ldp maxLspPerPeer 30,keepAliveHoldTimer 10,helloTimer 15,failedInitThreshold
10,loopDetection on,hopCountLimit 10
add La/0
set La/0 customerIdentifier 0,linkToProtocolPort Vr/0 Pp/OAMENET0,ifAdminStatus up,ifIndex 2
add La/0 Framer
set La/0 Framer interfaceName !Lp/0 OamEnet/0
add La/10
set La/10 customerIdentifier 0,linkToProtocolPort Vr/LER Pp/LA10,ifAdminStatus up,ifIndex 9
add La/10 Framer
set La/10 Framer interfaceName !Lp/1 Eth100/0
add La/11
set La/11 customerIdentifier 0,linkToProtocolPort Vr/LER Pp/LA11,ifAdminStatus up,ifIndex 10
add La/11 Framer
set La/11 Framer interfaceName !Lp/1 Eth100/1
add AtmIf/30
set AtmIf/30 customerIdentifier 0,interfaceName Lp/3 Sonet/0 Path/0,oamSegmentBoundary
yes,tempMaxVpcs 128,tempMaxVpts 128,tempMaxVccs 255,maxVpiBits
8,remoteAtmInterfaceLabel "",txCellMemory 100,faultHoldOffTime infinity,ctdCalculation
off,segLinkSideLoopback off,segSwitchSideLoopback off,endToEndLoopback off
add AtmIf/30 Iisp
set AtmIf/30 Iisp version iisp10Sig31,side userSide,softPvpAndPvcRetryPeriod
60,softPvpAndPvcHoldOffTime 0,interDomainCrankback disabled,accountCollection ! ~bill ~test
~study ~audit ~force,accountConnectionType intermediate,accountClass 0,serviceExchange 0
add AtmIf/30 Iisp Addr/47000002,primary
add AtmIf/30 Iisp Sig
set AtmIf/30 Iisp Sig vci 5,addressConversion none,operatingMode normal
add AtmIf/30 CA
set AtmIf/30 CA maxVpcs autoConfigure,maxVpts autoConfigure,maxVccs
autoConfigure,minMulticastBranches 0,maxMulticastBranches
autoConfigure,minAutoSelectedVpi 1,maxAutoSelectedVpi 128,minAutoSelectedVciForVpiZero
50,maxAutoSelectedVciForVpiZero 767,minAutoSelectedVciForNonZeroVpi
32,maxAutoSelectedVciForNonZeroVpi 63,permittedAtmServiceCategories ! unspecifiedBitRate
constantBitRate rtVariableBitRate nrtVariableBitRate,bandwidthPool 1 100  2 0  3 0  4 0  5 0
```

add AtmIf/30 CA Cbr/0
set AtmIf/30 CA Cbr/0 pool pool1,cdvt 250,cdv 250,maxCtd 1000,cellLossRatio 10,txQueueLimit autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate autoConfigure,holdingPriority 3,svcMpHoldingPriority 3,emissionPriority 0,ccForClp0 1,ccForClp1 3,clp1CcLevel 3,trafficShaping disabled,shapeRecoupPolicy minimumCdv,unshapedTransmitQueueing common,weightPolicy ecr,forceTagging disabled,usageParameterControl disabled
add AtmIf/30 CA RtVbr/0
set AtmIf/30 CA RtVbr/0 pool pool1,cdvt 250,cdv 1268,maxCtd 2000,cellLossRatio 10,txQueueLimit autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate autoConfigure,holdingPriority 3,svcMpHoldingPriority 3,emissionPriority 1,ccForClp0 1,ccForClp1 3,clp1CcLevel 3,trafficShaping disabled,shapeRecoupPolicy minimumCdv,unshapedTransmitQueueing autoConfigure,weightPolicy ecr,forceTagging disabled,usageParameterControl disabled
add AtmIf/30 CA NrtVbr/0
set AtmIf/30 CA NrtVbr/0 pool pool1,cdvt 250,cellLossRatio 7,txQueueLimit autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate autoConfigure,holdingPriority 3,svcMpHoldingPriority 3,emissionPriority 4,ccForClp0 2,ccForClp1 3,clp1CcLevel 3,trafficShaping disabled,shapeRecoupPolicy maximumEfficiency,unshapedTransmitQueueing autoConfigure,weightPolicy ecr,forceTagging disabled,usageParameterControl disabled
add AtmIf/30 CA Abr/0
set AtmIf/30 CA Abr/0 pool pool1,cdvt 250,txQueueLimit autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate autoConfigure,holdingPriority 3,emissionPriority 6,ccForClp0 2,ccForClp1 3,clp1CcLevel 3,fairnessPolicy pcrMinusMcr,frttPortion 5,abrConnectionType abrSwitch,usageParameterControl disabled,initialCellRate usePcr,rateDecreaseFactor 15,rateIncreaseFactor 0,maxCellPerRmCell 256,maxTimeBetweenRmCell 100000,cutoffDecreaseFactor 16,acrDecreaseTimeFactor 50,dgcraMaximumDelay 10000,dgcraMinimumDelay 1
add AtmIf/30 CA Ubr/0
set AtmIf/30 CA Ubr/0 maxVpcs sameAsCa,maxVpts sameAsCa,maxVccs sameAsCa,pool pool1,minimumCellRate 0,cdvt 250,txQueueLimit autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate autoConfigure,holdingPriority 3,svcMpHoldingPriority 3,emissionPriority 7,ccForClp0 3,ccForClp1 3,clp1CcLevel 1,trafficShaping disabled,shapeRecoupPolicy maximumEfficiency,unshapedTransmitQueueing autoConfigure,weightPolicy pcr,forceTagging disabled,usageParameterControl disabled
add AtmIf/30 Vcc/0.32
add AtmIf/30 Vcc/0.32 Nep
add AtmIf/30 Vcc/0.32 Vcd
set AtmIf/30 Vcc/0.32 Vcd segLinkSideLoopback sameAsInterface,segSwitchSideLoopback sameAsInterface,endToEndLoopback sameAsInterface,mCastConnectionType pointToPoint,correlationTag ""
add AtmIf/30 Vcc/0.32 Vcd Tm
set AtmIf/30 Vcc/0.32 Vcd Tm txTrafficDescType 1,txTrafficDescParm 1 0 2 0 3 0 4 0 5 0,txQueueLimit sameAsCa,holdingPriority 2,rxTrafficDescType sameAsTx,rxTrafficDescParm 1 0 2 0 3 0 4 0 5 0,atmServiceCategory unspecifiedBitRate,trafficShaping sameAsCa,unshapedTransmitQueueing sameAsCa,weight sameAsCa,forceTagging sameAsCa,usageParameterControl sameAsCa,fwdQosClass 0,fwdQosParameters cdv 16777215 ctd 16777215 clr 255,bwdQosClass sameAsFwd,bwdQosParameters cdv 16777215 clr 255,bearerClassBbc derivedFromServiceCategory,transferCapabilityBbc derivedFromServiceCategory,clippingBbc no,bestEffort derivedFromServiceCategory,txPacketWiseDiscard disabled,txWredMode disabled,txWredThreshold 25,rxPacketWiseDiscard disabled,abrConnectionType sameAsCa
add AtmIf/30 Pm
set AtmIf/30 Pm segSwitchSideMeasurement ! ~availabilityRatio ~cellLossRatio,segLinkSideMeasurement ! ~availabilityRatio ~cellLossRatio,controlMode autoStart
add AtmIf/31
set AtmIf/31 customerIdentifier 0,interfaceName Lp/3 Sonet/1 Path/0,oamSegmentBoundary yes,tempMaxVpcs 128,tempMaxVpts 128,tempMaxVccs 255,maxVpiBits

```
8,remoteAtmInterfaceLabel "",txCellMemory 100,faultHoldOffTime infinity,ctdCalculation
off,segLinkSideLoopback off,segSwitchSideLoopback off,endToEndLoopback off
add AtmIf/31 Iisp
set AtmIf/31 Iisp version iisp10Sig31,side userSide,softPvpAndPvcRetryPeriod
60,softPvpAndPvcHoldOffTime 0,interDomainCrankback disabled,accountCollection ! ~bill ~test
~study ~audit ~force,accountConnectionType intermediate,accountClass 0,serviceExchange 0
add AtmIf/31 Iisp Addr/47000003,primary
add AtmIf/31 Iisp Sig
set AtmIf/31 Iisp Sig vci 5,addressConversion none,operatingMode normal
add AtmIf/31 CA
set AtmIf/31 CA maxVpcs autoConfigure,maxVpts autoConfigure,maxVccs
autoConfigure,minMulticastBranches 0,maxMulticastBranches
autoConfigure,minAutoSelectedVpi 1,maxAutoSelectedVpi 128,minAutoSelectedVciForVpiZero
50,maxAutoSelectedVciForVpiZero 767,minAutoSelectedVciForNonZeroVpi
32,maxAutoSelectedVciForNonZeroVpi 63,permittedAtmServiceCategories ! unspecifiedBitRate
constantBitRate rtVariableBitRate nrtVariableBitRate,bandwidthPool 1 100 2 0 3 0 4 0 5 0
add AtmIf/31 CA Cbr/0
set AtmIf/31 CA Cbr/0 pool pool1,cdvt 250,cdv 250,maxCtd 1000,cellLossRatio 10,txQueueLimit
autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate
autoConfigure,holdingPriority 3,svcMpHoldingPriority 3,emissionPriority 0,ccForClp0 1,ccForClp1
3,clp1CcLevel 3,trafficShaping disabled,shapeRecoupPolicy
minimumCdv,unshapedTransmitQueueing common,weightPolicy ecr,forceTagging
disabled,usageParameterControl disabled
add AtmIf/31 CA RtVbr/0
set AtmIf/31 CA RtVbr/0 pool pool1,cdvt 250,cdv 1268,maxCtd 2000,cellLossRatio
10,txQueueLimit autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate
autoConfigure,holdingPriority 3,svcMpHoldingPriority 3,emissionPriority 1,ccForClp0 1,ccForClp1
3,clp1CcLevel 3,trafficShaping disabled,shapeRecoupPolicy
minimumCdv,unshapedTransmitQueueing autoConfigure,weightPolicy ecr,forceTagging
disabled,usageParameterControl disabled
add AtmIf/31 CA NrtVbr/0
set AtmIf/31 CA NrtVbr/0 pool pool1,cdvt 250,cellLossRatio 7,txQueueLimit
autoConfigure,minPerVcQueueLimit 88,perVcQueueLimitReferenceRate
autoConfigure,holdingPriority 3,svcMpHoldingPriority 3,emissionPriority 4,ccForClp0 2,ccForClp1
3,clp1CcLevel 3,trafficShaping disabled,shapeRecoupPolicy
maximumEfficiency,unshapedTransmitQueueing autoConfigure,weightPolicy ecr,forceTagging
disabled,usageParameterControl disabled
add AtmIf/31 CA Abr/0
set AtmIf/31 CA Abr/0 pool pool1,cdvt 250,txQueueLimit autoConfigure,minPerVcQueueLimit
88,perVcQueueLimitReferenceRate autoConfigure,holdingPriority 3,emissionPriority 6,ccForClp0
2,ccForClp1 3,clp1CcLevel 3,fairnessPolicy pcrMinusMcr,frttPortion 5,abrConnectionType
abrSwitch,usageParameterControl disabled,initialCellRate usePcr,rateDecreaseFactor
15,rateIncreaseFactor 0,maxCellPerRmCell 256,maxTimeBetweenRmCell
100000,cutoffDecreaseFactor 16,acrDecreaseTimeFactor 50,dgcraMaximumDelay
10000,dgcraMinimumDelay 1
add AtmIf/31 CA Ubr/0
set AtmIf/31 CA Ubr/0 maxVpcs sameAsCa,maxVpts sameAsCa,maxVccs sameAsCa,pool
pool1,minimumCellRate 0,cdvt 250,txQueueLimit autoConfigure,minPerVcQueueLimit
88,perVcQueueLimitReferenceRate autoConfigure,holdingPriority 3,svcMpHoldingPriority
3,emissionPriority 7,ccForClp0 3,ccForClp1 3,clp1CcLevel 1,trafficShaping
disabled,shapeRecoupPolicy maximumEfficiency,unshapedTransmitQueueing
autoConfigure,weightPolicy pcr,forceTagging disabled,usageParameterControl disabled
add AtmIf/31 Vcc/0.32
add AtmIf/31 Vcc/0.32 Nep
add AtmIf/31 Vcc/0.32 Vcd
set AtmIf/31 Vcc/0.32 Vcd segLinkSideLoopback sameAsInterface,segSwitchSideLoopback
sameAsInterface,endToEndLoopback sameAsInterface,mCastConnectionType
pointToPoint,correlationTag ""
add AtmIf/31 Vcc/0.32 Vcd Tm
```

```
set AtmIf/31 Vcc/0.32 Vcd Tm txTrafficDescType 1,txTrafficDescParm 1 0 2 0 3 0 4 0 5
0,txQueueLimit sameAsCa,holdingPriority 2,rxTrafficDescType sameAsTx,rxTrafficDescParm 1 0
2 0 3 0 4 0 5 0,atmServiceCategory unspecifiedBitRate,trafficShaping
sameAsCa,unshapedTransmitQueueing sameAsCa,weight sameAsCa,forceTagging
sameAsCa,usageParameterControl sameAsCa,fwdQosClass 0,fwdQosParameters cdv 16777215
ctd 16777215 clr 255,bwdQosClass sameAsFwd,bwdQosParameters cdv 16777215 clr
255,bearerClassBbc derivedFromServiceCategory,transferCapabilityBbc
derivedFromServiceCategory,clippingBbc no,bestEffort
derivedFromServiceCategory,txPacketWiseDiscard disabled,txWredMode
disabled,txWredThreshold 25,rxPacketWiseDiscard disabled,abrConnectionType sameAsCa
add AtmIf/31 Pm
set AtmIf/31 Pm segSwitchSideMeasurement ! ~availabilityRatio
~cellLossRatio,segLinkSideMeasurement ! ~availabilityRatio ~cellLossRatio,controlMode autoStart
add AtmMpe/30
set AtmMpe/30 customerIdentifier 0,ifAdminStatus up,ifIndex 6,ilsForwarder
!,maxTransmissionUnit 9188,encapType llcEncap,linkToProtocolPort Vr/LER Pp/LER1
add AtmMpe/30 Ac/1
set AtmMpe/30 Ac/1 atmConnection AtmIf/30 Vcc/0.32 Nep,ipCoS 0
add AtmMpe/31
set AtmMpe/31 customerIdentifier 0,ifAdminStatus up,ifIndex 8,ilsForwarder
!,maxTransmissionUnit 9188,encapType llcEncap,linkToProtocolPort Vr/LER Pp/LER2
add AtmMpe/31 Ac/1
set AtmMpe/31 Ac/1 atmConnection AtmIf/31 Vcc/0.32 Nep,ipCoS 0
add ARtg
set ARtg useBestMatchAddrOnCrankback none
add Vm/1
add Vm/1 If/0
set Vm/1 If/0 customerIdentifier 0,ifAdminStatus up,ifIndex 11,linkToProtocolPort Vr/LER
Pp/VM01IF0,mode alwaysUpInterface
end prov
```

APPENDIX B

This is a basic MPLS Path provisioning script.

```
;prompts
Set Path attributes
============================

;commands
start prov

add Vr/LER Mpls Path/1
add Vr/LER Mpls Path/1 Hop/1
set Vr/LER Mpls Path/1 Hop/1 ipv4Address 10.5.20.1,ipv4PrefixLen 32,hopMode strict
add Vr/LER Mpls Path/1 Hop/2
set Vr/LER Mpls Path/1 Hop/2 ipv4Address 10.5.30.1,ipv4PrefixLen 32,hopMode strict
add Vr/LER Mpls Path/2
add Vr/LER Mpls Path/2 Hop/1
set Vr/LER Mpls Path/2 Hop/1 ipv4Address 10.5.30.1,ipv4PrefixLen 32,hopMode strict

chact prov
```

APPENDIX C

This is a basic MPLS LSPG provisioning script. At the prompt, enter the group ID, QoS level (1, 2 or 3), data rate, path ID, destination and source IP addresses.

```
;prompts
Set LSPG attributes
===========================
Enter Group ID: <grp_id>
Enter QoS (3-gold, 2-silver, 1-bronze): <qos>
Enter Data Rate (in octets): <rate>
Enter Path ID: <path_id>
Enter Dest Addr (10.4.11.103): <dest_addr>
Enter Source Addr (10.4.1.101): <src_addr>

;commands
start prov

add Vr/LER Mpls LspG/<grp_id>
add Vr/LER Mpls LspG/<grp_id> IpFwd
add Vr/LER Mpls LspG/<grp_id> IpFwd StIpv4/<dest_addr>,32
set Vr/LER Mpls LspG/<grp_id> IpFwd StIpv4/<dest_addr>,32 bestMatchOverride disabled
add Vr/LER Mpls LspG/<grp_id> IpFwd MscMap
set Vr/LER Mpls LspG/<grp_id> IpFwd MscMap cosToMsc 0 0  1 1  2 2  3 3

add Vr/LER Mpls LspG/<grp_id> Lsp/0
add Vr/LER Mpls LspG/<grp_id> Lsp/0 CrQos
set Vr/LER Mpls LspG/<grp_id> Lsp/0 CrQos peakDataRate 125,peakBurstSize
0,committedDataRate 0,committedBurstSize 0,frequency unspecified
add Vr/LER Mpls LspG/<grp_id> Lsp/0 Te
set Vr/LER Mpls LspG/<grp_id> Lsp/0 Te pathName Vr/LER Mpls
Path/<path_id>,pathOnDemand disabled,alarmControl none
add Vr/LER Mpls LspG/<grp_id> Lsp/0 Msc
set Vr/LER Mpls LspG/<grp_id> Lsp/0 Msc standbyLsp !,serviceCategory 0

add Vr/LER Mpls LspG/<grp_id> Lsp/<qos>
add Vr/LER Mpls LspG/<grp_id> Lsp/<qos> CrQos
set Vr/LER Mpls LspG/<grp_id> Lsp/<qos> CrQos peakDataRate <rate>,peakBurstSize
<rate>,committedDataRate <rate>,committedBurstSize <rate>,frequency veryFrequent
add Vr/LER Mpls LspG/<grp_id> Lsp/<qos> Te
set Vr/LER Mpls LspG/<grp_id> Lsp/<qos> Te pathName Vr/LER Mpls
Path/<path_id>,pathOnDemand disabled,alarmControl none
add Vr/LER Mpls LspG/<grp_id> Lsp/<qos> Msc
set Vr/LER Mpls LspG/<grp_id> Lsp/<qos> Msc standbyLsp !,serviceCategory <qos>

# QoS Filters for IP
add vr/ler ip pg/1 policy/<grp_id>
set vr/ler ip pg/1 policy/<grp_id> assignedcos <qos>
add vr/ler ip pg/1 policy/<grp_id> ipaddrlayer4flow/<qos>
set vr/ler ip pg/1 policy/<grp_id> ipaddrlayer4flow/<qos> prefix <src_addr>
set vr/ler ip pg/1 policy/<grp_id> ipaddrlayer4flow/<qos> prefixlen 32
set vr/ler ip pg/1 policy/<grp_id> ipaddrlayer4flow/<qos> protocol udp

chact prov
```

APPENDIX D

Transmitting IP Packets using IPFS

This is an example of how the IPFS driver works, using a simple case of transferring a single IP datagram between two hosts connected through G50s on the same ring. A network layout diagram for this example is given in Figure A.1. Suppose we have host A with IP address 192.168.3.33 connected to G50 C with HRN address 00-FA-01-01-01-04 and host B with IP address 192.168.1.144 connected to G50 E with HRN address 00-FA-01-01-01-03 – these two G50s are on the same ring (00-FA-01-01-01).

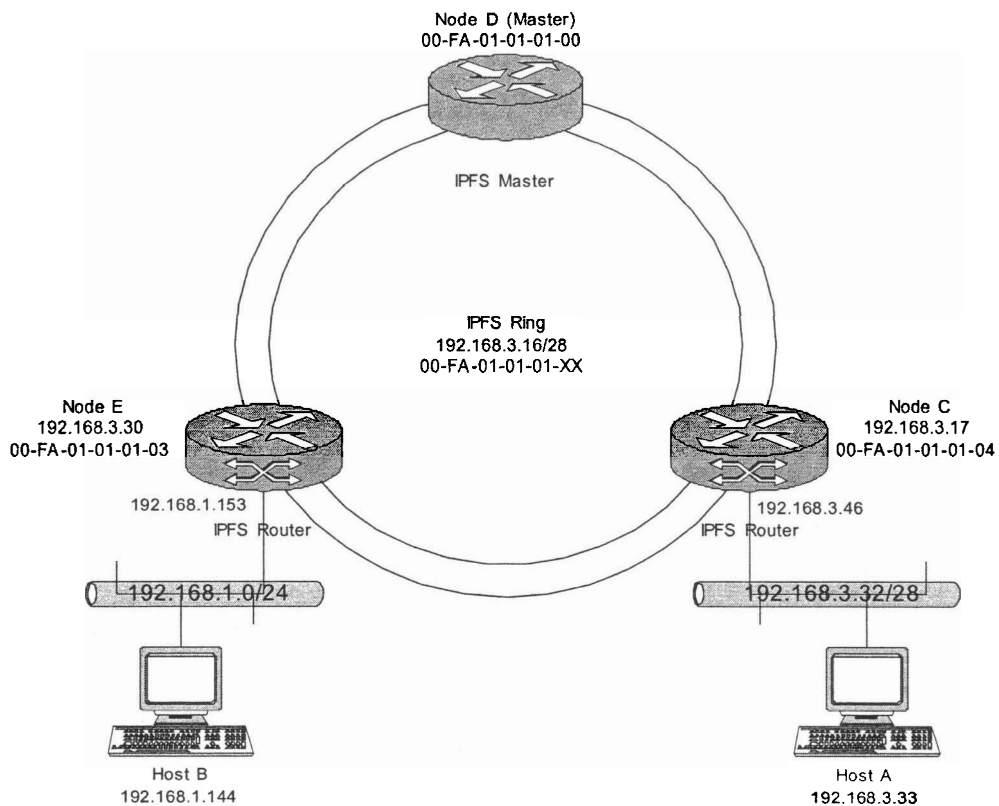

Figure A.1 – Network Layout: IP datagram Host A → Host B

Transfer of IP datagrams with data ≤ 216 bytes long

Host A wants to send an IP datagram (UDP) containing 100 bytes of data to Host B. The UDP packet will be 108 bytes long (8 byte UDP Header); the IP packet will be 128 bytes long (20 byte IP Header – assuming no options); the Ethernet frame will be 148 bytes long (16 byte Ethernet Header + 4 byte Ethernet Frame Check Sum).

Host A will make a routing call to resolve destination IP address 192.168.1.144 and be given the gateway IP address of node C (192.168.3.17). This gateway address will then be resolved to an HRN address of 00-FA-01-01-01-04. Host A will send the Ethernet frame onto the network segment and node C will grab it. Node C's networking system will resolve destination address 192.168.1.144 to the gateway address of node E (192.168.3.30), which is resolved to an HRN address of 00-FA-01-01-01-03.

The networking system will hand off the Ethernet packet to the IPFS driver, which will construct an ERP frame header and chunk the IP packet into chunks ≤ 244 bytes. The ERP frame header will have hop limit > 1, priority = 0 (low priority), Area code = 00-FA, mode ='11' (unicast), destination Ring | Node address = 01-01-01-03, CRC8 = 0 (this is calculated by the FPGA logic), flow tag set to a unique flow ID, and length field = 128. The driver will then transfer the ERP header and payload (IP packet) to the FPGA Add FIFO via the ISA Bus (16-bit word writes). Since there is only one chunk (IP packet = 128 bytes < 244), the driver will only transfer the one chunk. However, to signal the FPGA that there is an ERP frame ready in the ADD FIFO (and to reset the FIFO so it is ready for another frame to be transferred), the driver will write a NULL value to the end of the ERP frame at address offset 128 (word access). The ERP frame is 256 bytes long so NULLs are appended to the 160 byte (12 byte Header) payload by the FPGA to fill the ERP frame.

The G50 transmits the ERP frame (with 116 bytes of padding) in the appropriate IPFS frame slot to G50 with HRN address 00-FA-01-01-01-03 (node E). Intermediate G50s will examine the destination HRN address of the frame and determine to Pass the frame on. Master G50s will also decrement the Hop Limit field and throw away any ERP

frames with hop limit = 0 (thus, why hop limit should be set to > 1 for single ring transmission).

When node E sees the ERP frame, it will examine the destination HRN address and determine to Drop the frame to this node's Drop FIFO. The driver polls the Drop FIFO waiting for any ERP frames needing to be transferred to system memory. Once the ERP frame has been completely dropped to the Drop FIFO, the driver transfers the frame (16-bit word reads) via the ISA Bus. At this point the ERP header is examined and the payload extracted. From the header (length and flow fields), the driver determines that the payload contains the first and only chunk of an IP packet. The driver reconstructs the Ethernet header and hands off the Ethernet frame to the networking system.

The networking system resolves the destination IP address (192.168.1.144) and determines that this host is reachable via its local LAN. The Ethernet MAC address is resolved to host B and the Ethernet frame is transmitted on the Ethernet segment. Host B grabs the Ethernet frame, extracts the IP packet, extracts the UDP datagram, and delivers the data to the destination port.

Transfer of IP datagrams with data > 216 bytes long

Suppose we wish to transfer an IP datagram containing more than 216 bytes of data (total IP packet size > 244 bytes). We follow the same procedure as above except when we come to chunking the IP packet at the ingress G50 node. Since the IP packet is > 244 bytes, the IPFS driver will extract the first 244 bytes and prepend the header as before. Then it will continue with chunking the IP packet into 244 byte chunks until we have a final chunk ≤ 244 bytes. For each chunk, the driver will transfer the same ERP header to the Add FIFO followed by the chunk.

APPENDIX E

Scheduling Frame Slot in IPFS

The three scheduling scenarios for setting up an end-to-end path through an HRN are:

- **Same Ring Scheduling**
  the source node is on the same ring as the destination node.
- **Sub-ring to Super-ring Scheduling**
  the source node is on a sub-ring of the ring with the destination node.
- **Super-ring to Sub-ring Scheduling**
  the source node is on a super-ring of the ring with the destination node.

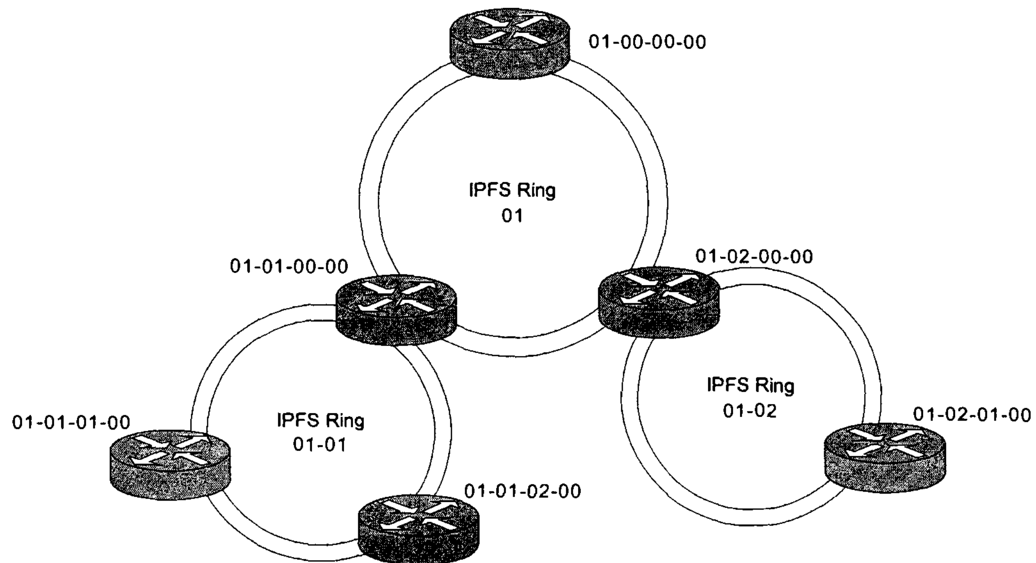

Figure A.2 – Network Layout: Multi-Ring Scheduling

To illustrate these scenarios, we use the example HRN as given in Figure A.2.

Same Ring Scheduling

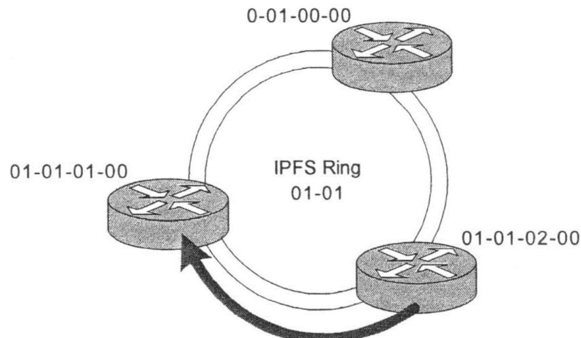

Figure A.3 – Network Layout: Same Ring Scheduling

We wish to setup a DS0 between node 01-01-02-00 and node 01-01-01-00 as shown in Figure A.3. The master node of this ring is node 01-01-00-00. We add an entry of one frame slot with Schedule ID 02 in the Schedule table of the master node 01-01-00-00 for node 01-01-02-00. Then when node 01-01-02-00 sees its Schedule ID (02), it will add its traffic into the associated ERP frame and send it on the ring.

Sub-ring → Super-ring Scheduling

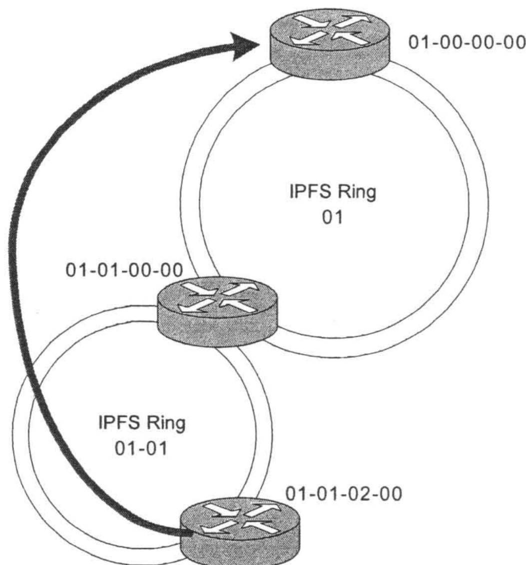

Figure A.4 – Network Layout: Sub-ring → Super-ring Scheduling

We wish to setup a DS0 between node 01-01-02-00 and node 01-00-00-00 as shown in Figure A.4. Traffic for this DS0 will need to traverse two rings – 01-01 and 01. As

in the previous scenario, we add an entry of one frame slot with Schedule ID 02 in the Schedule table of the master node 01-01-00-00 for node 01-01-02-00. This enables a DS0 on ring 01-01 for node 01-01-02-00. We also add an entry of one frame slot with Schedule ID 01 in the Schedule table of the master node 01-00-00-00 for node 01-01-00-00. This enables a DS0 on ring 01 for node 01-01-00-00. These two schedules will give us the required overall scheduling we need. Node 01-01-02-00 will add its traffic into IPFS frames with Schedule ID 02 on ring 01-01. This traffic will be copied by node 01-01-00-00 from ring 01-01 to ring 01 and inserted into IPFS frames with Schedule ID 01 on ring 01. The traffic will be dropped by node 01-00-00-00 as specified.

Super-ring → Sub-ring Scheduling

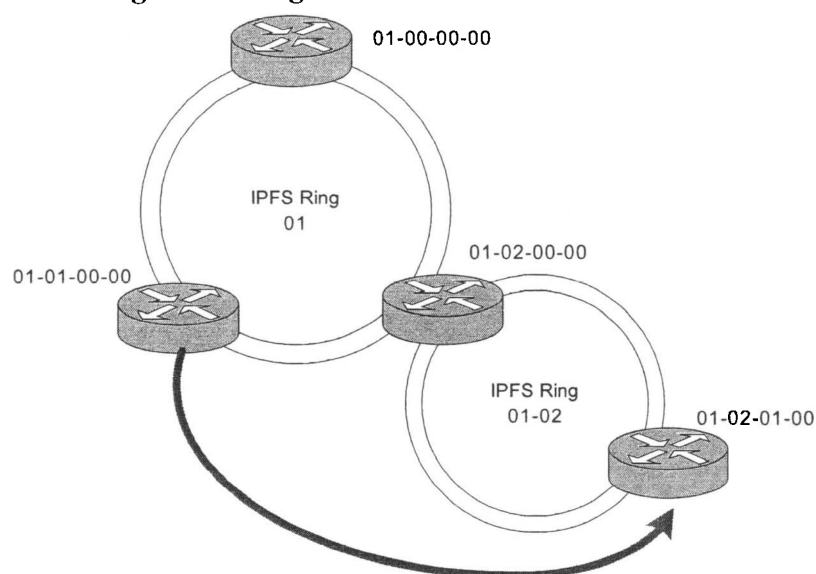

Figure A.5 – Network Layout: Super-ring → Sub-ring Scheduling

We wish to setup a DS0 between node 01-01-00-00 and node 01-02-01-00 as shown in Figure A.5. Traffic for this DS0 will also need to traverse two rings – 01 and 01-02. As in the second part of the previous scenario, we add an entry of one frame slot with Schedule ID 01 in the Schedule table of the master node 01-00-00-00 for node 01-01-00-00. This enables a DS0 on ring 01 for node 01-01-00-00. We also need to schedule one frame slot on ring 01-02 for node 01-02-00-00. However, this is the master node for ring 01-02. This has some implications on scheduling depending on the

default scheduling employed, since we are now talking about traversing the HRN downwards.

If the default scheduling uses a Schedule ID of 00 for all non-scheduled entries in the Schedule table then this implies that the master node (with Schedule ID 00) has all frame slots already preallocated. Thus, there is no problem scheduling one frame slot for itself as it already has the majority. In this case, the master would maintain a separate counter with the value of how many frame slots it has hard scheduled – scheduled as part of an end-to-end path. This counter is needed so that the master can determine when it has fully scheduled the ring to avoid over-scheduling. However, another possibility is to treat the master node just like any other requestor and use a default schedule with some Schedule ID other than 00, such as FF. There are other implications in choosing which should be the default schedule so we leave this as an implementation decision and assume that the master needs to be scheduled just like any other node on the ring.

Scheduling one frame slot with Schedule ID 00 on ring 01-02 for node 01-02-00-00 will then enable a DS0 on this ring for this node – the master. These two schedules give us the required overall scheduling we need. Node 01-01-00-00 will add its traffic into IPFS frames with Schedule ID 01 on ring 01. This traffic will be copied by node 01-02-00-00 from ring 01 to ring 01-02 and inserted into IPFS frames with Schedule ID 00 on ring 01-02. The traffic will be dropped by node 01-02-01-00, as specified.

APPENDIX F

SLA Processing & Path Signaling

We combine our discussion on SLA processing and Path signaling to show how end-to-end paths are set up in an HRN

The three signaling scenarios for setting up an end-to-end path through an HRN are:

- **SLA Request on Single Ring**
  the ingress gateway is on the same ring as the egress gateway.
- **SLA Request from Sub-ring → Super-ring**
  the ingress gateway is on a sub-ring of the ring with the egress gateway.
- **SLA Request from Super-ring → Sub-ring**
  the ingress gateway is on a super-ring of the ring with the egress gateway.

SLA Request on Single Ring

The network layout for this scenario is illustrated in Figure A.6.

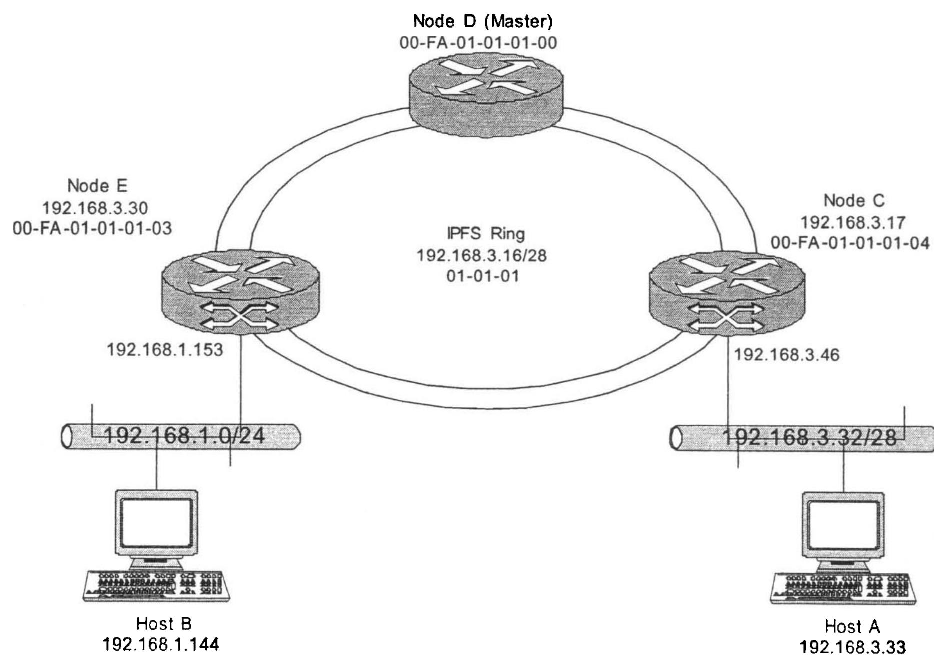

Figure A.6 – Network Layout: Single-level HRN

Suppose we have a request for 10Mbps (= 10000Kbps) traffic from host A to host B. The source host A would send an SLA request to ingress node C.

```
sla add -srcIP 192.168.3.33 -destIP 192.168.1.144
        -bw 10000
```

Node C would then parse the request and determine if the ingress and egress gateways have been resolved; this is done by examining the source and destination IP addresses and performing a routing call to determine the ingress and egress gateway IP addresses. These IP addresses then are resolved to HRN addresses using ARP. Next we determine how many frames are needed to schedule this request. The request is for 10Mbps, which equates to a schedule of 146 frames in the schedule table. This is calculated by dividing the requested bandwidth by the scheduling granularity (~69Kbps).

```
#frames = 10000Kbps / 68.625Kbps = 146
```

The SLA request is mapped into a Path request, with the source and destination IP addresses replaced by the ingress and egress gateways, and the bandwidth replaced by the number of frames needed. Node C sends this Path request to its master node D.

```
path add   –srcGW 00-FA-01-01-01-04 –destGW 00-FA-01-01-01-03
           -frames 146
```

Node D receives the request and determines that it controls the scheduling this request needs. Furthermore, an AND between the source node HRN address and the inverse of node D's mask gives the nodeID for scheduling.

```
-are gateways under my direct control? → Yes
HRN C  00-FA-01-01-01-04     HRN E    00-FA-01-01-01-03
MASK   FF-FF-FF-FF-FF-00     MASK     FF-FF-FF-FF-FF-00
       =================              =================
       00-FA-01-01-01-00              00-FA-01-01-01-00
?= HRN D? → YES              ?= HRN D? → YES
HRN C  00-FA-01-01-01-04
!MASK  00-00-00-00-00-FF
       =================
Src nodeID C: 04
```

Node D then tries to schedule the requested 146 frames with Schedule ID 04.

```
schedule add -node 04 -frames 146
```

If the scheduling is successful then node D can reply with an ACK to node C, otherwise it would send a NACK. If node C receives an ACK from node D then it would know that the request was successful and it could store the SLA in its database and in turn acknowledge host A that it can use the requested 10Mbps stream. The UML Sequence chart for this successful scenario is presented Figure A.7.

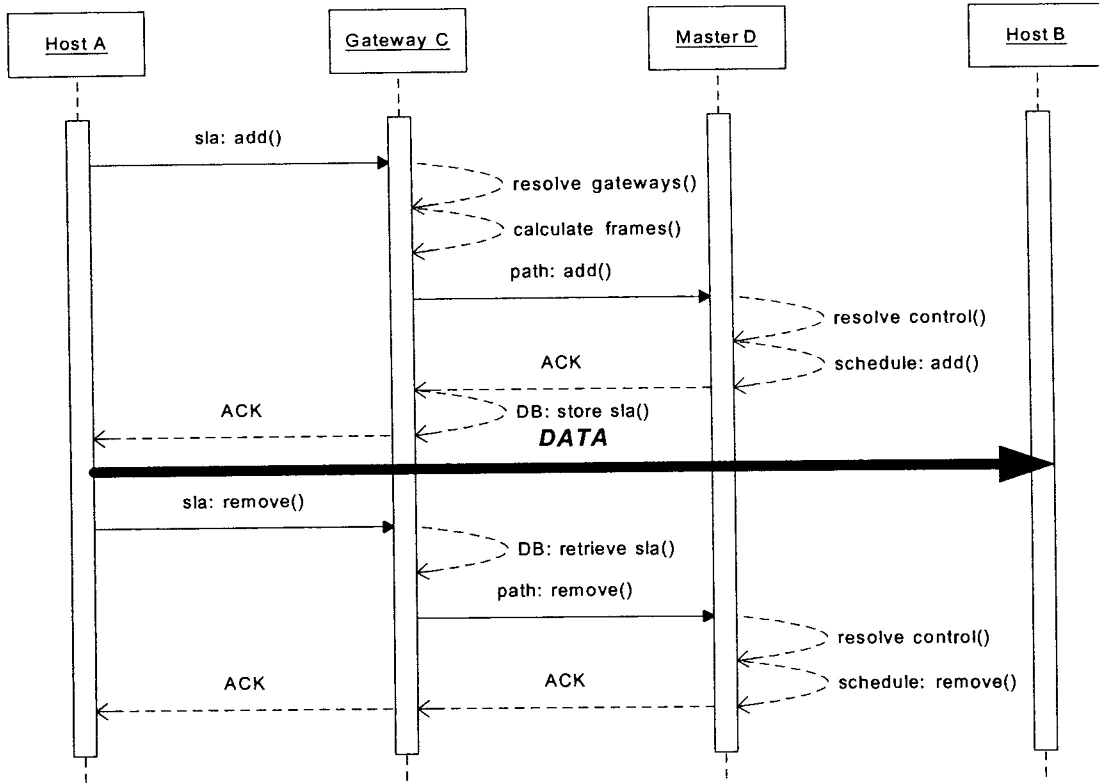

Figure A.7 – Single Ring ACK Sequence Chart

If the request wasn't successful (node C received a NACK), then node C informs host A that the stream could not be setup. The UML Sequence chart for this unsuccessful scenario is given in Figure A.8.

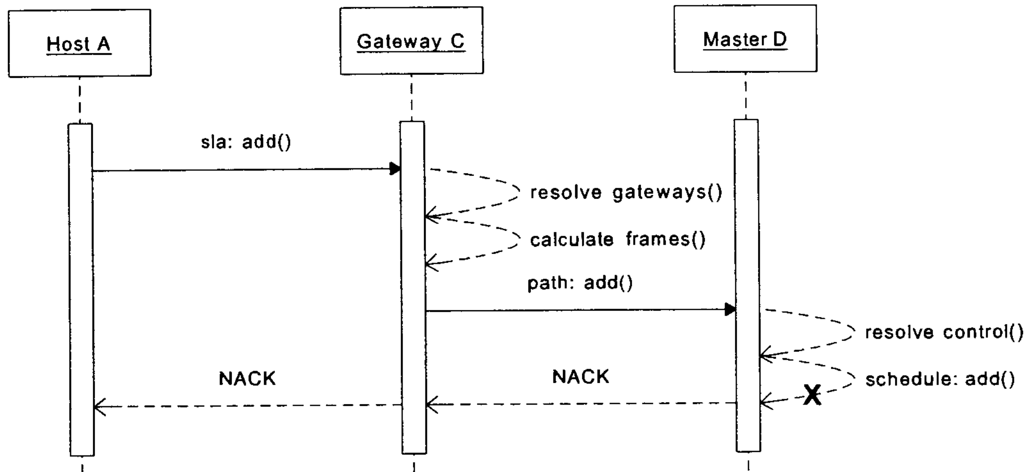

Figure A.8 – Single Ring NACK Sequence Chart

Assuming the SLA request was successful and host A has subsequently finished sending its data to host B, it is now the responsibility of host A to free up the allocated stream resources. Host A sends an SLA request once again to ingress node C, but this time with the command remove:

```
sla remove  –slaID 1
```

Node C retrieves the SLA from its database, and sends a Path remove request to its master – node D; the request is created from the SLA parameters retrieved from the database.

```
path remove     –srcGW 00-FA-01-01-01-04 –destGW 00-FA-01-01-01-03
                -frames 146
```

Node D receives the Path request and determines that it controls the scheduling. As before, an AND between the source gateway HRN address and the inverse of node D's mask gives the nodeID for scheduling.

```
-are gateways under my direct control? → Yes
HRN C  00-FA-01-01-01-04      HRN E     00-FA-01-01-01-03
MASK   FF-FF-FF-FF-FF-00      MASK      FF-FF-FF-FF-FF-00
       =================                =================
       00-FA-01-01-01-00                00-FA-01-01-01-00
?= HRN D? → YES               ?= HRN D? → YES
HRN C  00-FA-01-01-01-04
!MASK  00-00-00-00-00-FF
       =================
Src nodeID C: 04
```

Node D then tries to remove the requested 146 frames with Schedule ID 04 from its schedule.

```
schedule remove -node 04 -frames 146
```

If this is successful then node D can reply with an ACK to node C, otherwise it would send a NACK. This SLA removal process is illustrated at the bottom of the UML Sequence chart given in Figure A.7.

This simple scenario provides a lot of the details of how signaling is done between nodes which are on a single ring. The next scenario looks at adding complexity to the signaling for streams that need to traverse two rings; in particular, traversing the hierarchy bottom-up. That is, going from a sub-ring to a super-ring.

SLA Request from Sub-ring → Super-ring

In this scenario we add a level to the hierarchy and move the egress gateway (node E) to the higher ring. This new network layout is illustrated in Figure A.9. Note that moving the egress gateway means that the HRN address changes to 00-FA-01-01-02-00.

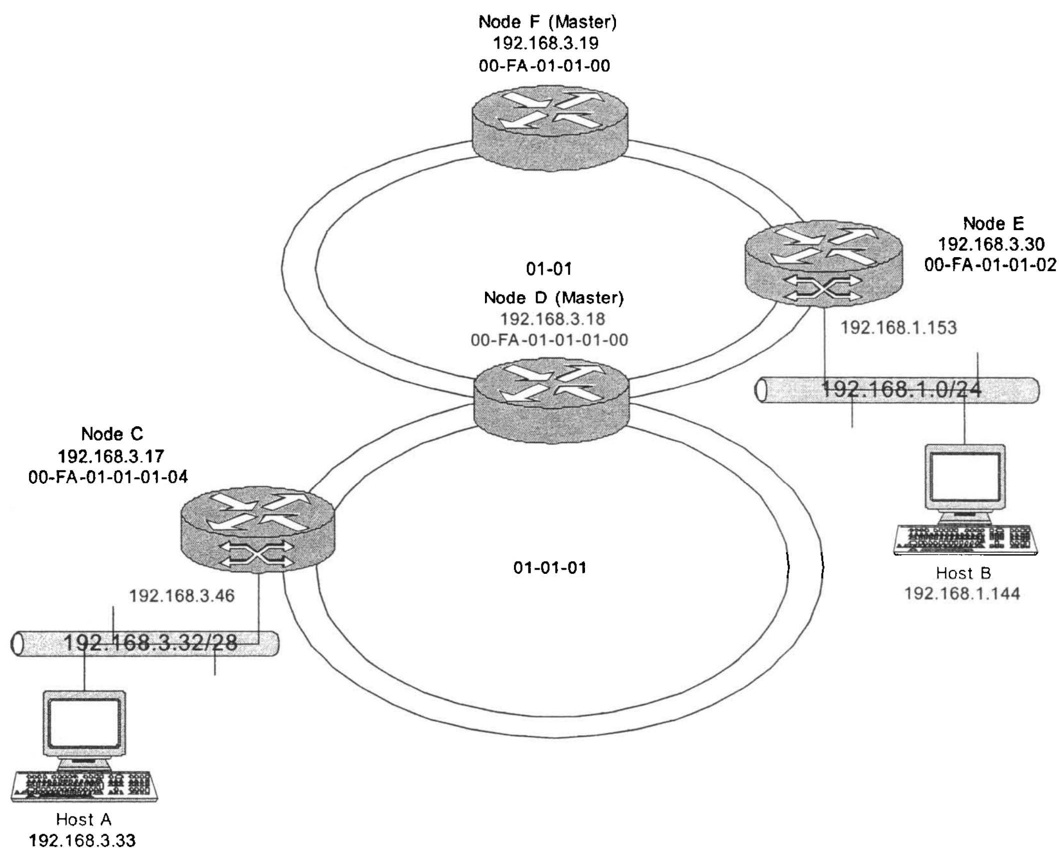

Figure A.9 – Network Layout: Two-level HRN

Once again we want to setup a 10Mbps stream between host A and host B. Host A with IP address 192.168.3.33 sends a request to ingress node C with IP address 192.168.3.46 (we use the Ethernet interface address for external requests).

```
sla add -srcIP 192.168.3.33 -destIP 192.168.1.144
        -bw 10000
```

Node C resolves the ingress and egress gateway addresses – ingress: 192.168.3.17 (itself) and egress: 192.168.3.30 (we use the IPFS interface addresses for internal HRN requests) to HRN addresses – ingress: 00-FA-01-01-01-04 and egress: 00-FA-01-01-02-00. A Path request is sent to master node D with IP address 192.168.3.18.

path add –srcGW **00-FA-01-01-01-04** –destGW **00-FA-01-01-02-00** -frames 146

Node D determines that it doesn't control all of the resources necessary for this request; however it does control the scheduling for the ring on which the ingress gateway resides. At this point, master D reserves a schedule of 146 frames on its local ring for node C with Schedule ID 04.

schedule reserve -node **04** -frames 146

If the reservation is successful then node D continues with requesting admission onto the upper ring (controlled by node F). But if there are not enough resources on its ring, then the request stops at this point and is denied; a NACK is then sent to node C. Assuming the reservation was successful, node D sends a request to its master node F with IP address 192.168.3.19.

path add –srcGW **00-FA-01-01-01-00** –destGW **00-FA-01-01-02-00** -frames 146

Note how node D changed the source gateway to itself since it is the one requesting access to the upper ring; this is an example of schedule aggregation.

Node F determines that it does control all of the resources for this request so it tries to schedule 146 frames with Schedule ID 01.

schedule add -node **01** -frames 146

If this schedule is successful then an ACK is sent back to node D, else a NACK is sent. When node D receives the ACK reply from master F it commits the previously reserved schedule and replies to node C with an ACK.

schedule commit -node **04** -frames 146

If node D receives a NACK from master F then it frees its previous reservation and sends a NACK to node C.

schedule free -node **04** -frames 146

The rest of the process is as in the first scenario. The UML Sequence chart illustrating the successful two ring SLA request is given in Figure A.10. As before, the source host is responsible for freeing up the allocated stream resources once they are finished with the stream; this removal process is not discussed, as it is just a simple expansion of the process for a single ring. The UML Sequence chart given in Figure A.11 shows two ways this request might fail and what happens with the signaling.

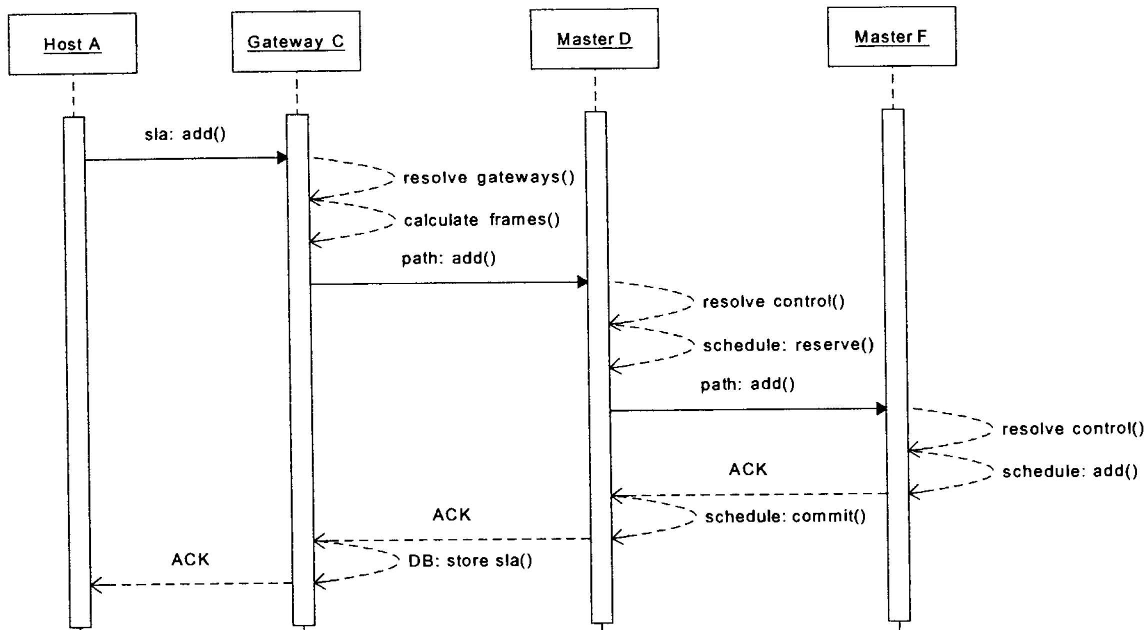

Figure A.10 – Sub-ring → Super-ring ACK Sequence Chart

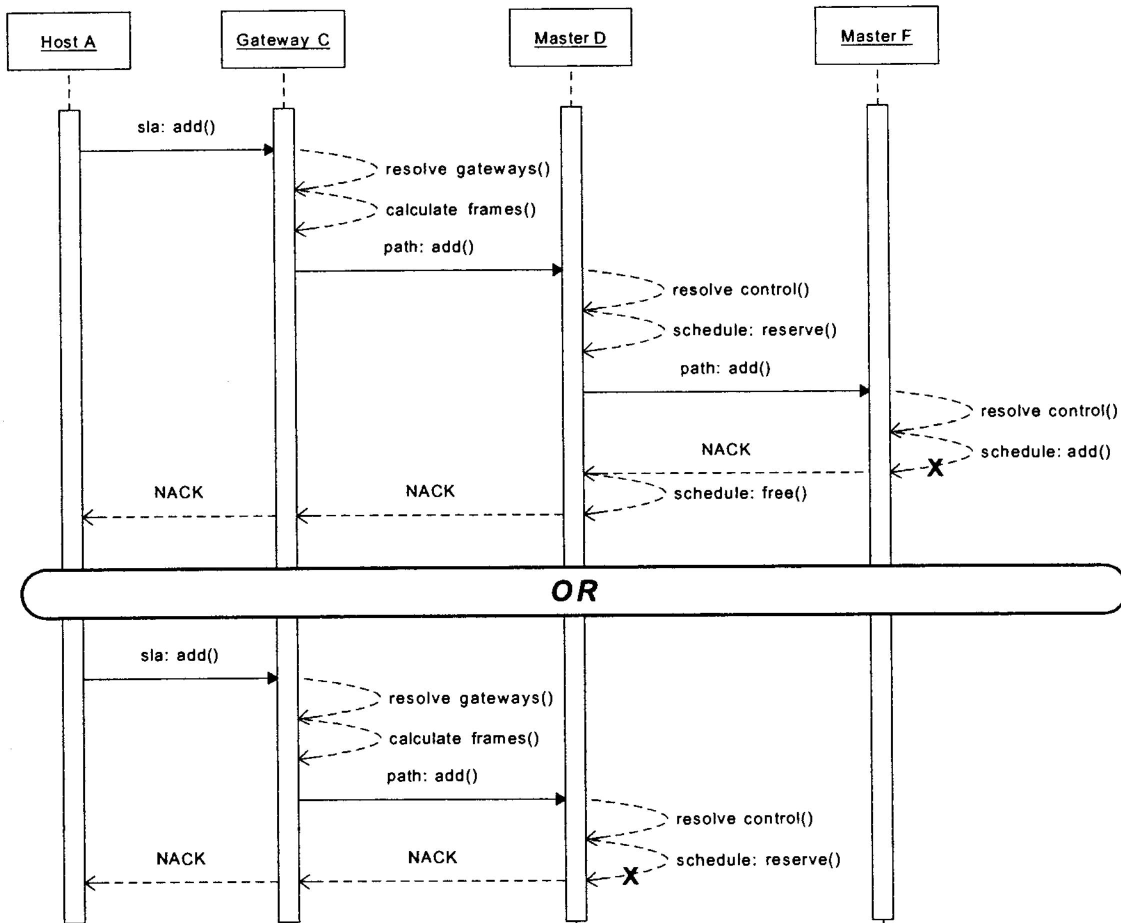

Figure A.11 – Sub-ring → Super-ring NACK Sequence Chart

This scenario looked at sending a stream up the HRN, but there is one more scenario to consider – sending a stream down the HRN. That is, going from a super-ring to a sub-ring.

SLA Request from Super-ring → Sub-ring

For this example we'll use the same layout as the previous, however suppose host B wants to send 10Mbps traffic to host A; this is in the opposite direction. Host B sends a request to ingress node E (192.168.1.153).

```
sla add -srcIP 192.168.1.144 -destIP 192.168.3.33
        -bw 10000
```

Node E resolves the gateway addresses (ingress: 192.168.3.30 → 00-FA-01-01-02-00, egress: 192.168.3.17 → 00-FA-01-01-01-04). From this, the node creates a Path request to be sent to its master node F (192.168.3.19)

```
path add    –srcGW 00-FA-01-01-02-00 –destGW 00-FA-01-01-01-04
            -frames 146
```

Master node F reserves 146 frames in its schedule for node ID 02

```
schedule reserve -node 02 -frames 146
```

and requests scheduling from the master of ring 00-FA-01-01-01-00 (node D – 192.168.3.18).

```
path add    –srcGW 00-FA-01-01-01-00 –destGW 00-FA-01-01-01-04
            -frames 146
```

Node D then schedules itself for 146 frames.

```
schedule add    -node 00 -frames 146
```

Assuming success at all stages, node D then ACKs node F which commits its schedule for node C, and so on.

```
schedule commit -node 02 -frames 146
```

This example seems to be almost exactly like the previous one, however we have glossed over some details – particularly, where node F requests from node D. Unlike the previous example, node D is not the master of node F. So how does node F know to request scheduling from node D? Looking at the bit-wise operations shows us how.

```
HRN E  00-FA-01-01-02-00      HRN C      00-FA-01-01-01-04
MASK   FF-FF-FF-FF-00-FF      MASK       FF-FF-FF-FF-00-FF
       =================                 =================
       00-FA-01-01-00-00                 00-FA-01-01-00-04
?= HRN F? → YES               ?= HRN F? → NO
```

```
HRN D  00-FA-01-01-02-00      HRN E     00-FA-01-01-01-04
!MASK  00-00-00-00-FF-00      !MASK     00-00-00-00-FF-00
       =================                ================
Src nodeID C: 02              Dest nodeID C: 01
```

Based on these calculations, node F can go ahead and reserve a schedule for node ID 02, but because the destination node is not directly located on its ring, node F must then forward the request to the next node in the path – node D with HRN address 00-FA-01-01-01-00. This HRN address is the concatenation of the area code 00-FA, ringID 01-01, and the destination nodeID 01 (left-aligned and 00-filled). This HRN address is resolved to IP address 192.168.3.18 using RARP. The UML Sequence chart for the successful case is given in Figure A.12.

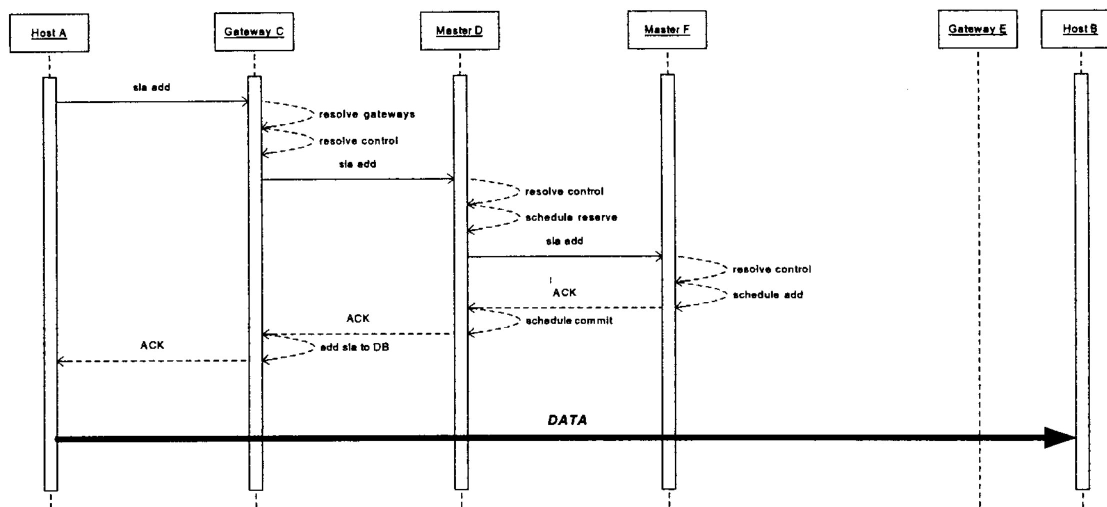

Figure A.12 – Super-ring → Sub-ring ACK Sequence Chart