

Winter 1995

# Distributed cell scheduling and quality of service in ATM networks

Paul Niel Goransson

*University of New Hampshire, Durham*

Follow this and additional works at: <https://scholars.unh.edu/dissertation>

---

## Recommended Citation

Goransson, Paul Niel, "Distributed cell scheduling and quality of service in ATM networks" (1995). *Doctoral Dissertations*. 1873.  
<https://scholars.unh.edu/dissertation/1873>

This Dissertation is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact [nicole.hentz@unh.edu](mailto:nicole.hentz@unh.edu).

## **INFORMATION TO USERS**

**This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.**

**The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.**

**In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.**

**Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.**

**Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.**

# **UMI**

A Bell & Howell Information Company  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
313/761-4700 800/521-0600



# **DISTRIBUTED CELL SCHEDULING AND QUALITY OF SERVICE IN ATM NETWORKS**

BY

**PAUL GORANSSON**

B.A., Brandeis University, 1975  
M.S., Boston University, 1981

**DISSERTATION**

Submitted to the University of New Hampshire  
in Partial Fulfillment of  
the Requirements of the Degree of

Doctor of Philosophy

in

Computer Science

December, 1995



UMI Number: 9617072

Copyright 1996 by  
Goransson, Paul Niel

All rights reserved.

---

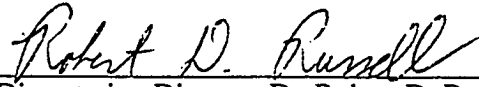
UMI Microform 9617072  
Copyright 1996, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized  
copying under Title 17, United States Code.

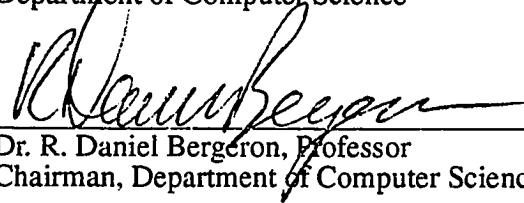
---

**UMI**  
300 North Zeeb Road  
Ann Arbor, MI 48103

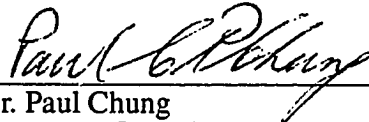
This dissertation has been examined and approved.



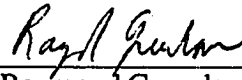
Dissertation Director, Dr. Robert D. Russell  
Associate Professor  
Department of Computer Science



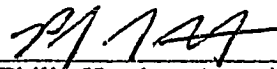
Dr. R. Daniel Bergeron, Professor  
Chairman, Department of Computer Science



Dr. Paul Chung  
Computer Consultant



Dr. Raymond Greenlaw, Associate Professor  
Department of Computer Science



Dr. Philip Hatcher, Associate Professor  
Department of Computer Science



Dr. Ernst Linder, Associate Professor  
Department of Mathematics

DECEMBER 7, 1995

Date

## **DEDICATION**

I dedicate this work to my father, Ake Gunnar Goransson, who did not live to see me reach the end of this journey, and to my son, Peter Ake Goransson, whose nightly prayers over the past seven years that I might succeed in this endeavor have finally been answered.

## ACKNOWLEDGEMENTS

First and foremost, I wish to express my most sincere thanks to my thesis advisor, Prof. Robert. D. Russell. He has been tireless in his efforts to help me develop my ideas into this dissertation. His criticisms, insights, and patience have been key in the successful completion of this thesis.

I wish also to thank Dr. Paul Chung, Prof. Ernst Linder, Prof. Daniel Bergeron, Prof. Raymond Greenlaw, and Prof. Philip Hatcher for reading my thesis and agreeing to serve on my thesis defense committee. I especially thank Drs. Greenlaw and Hatcher for their extra efforts in helping me improve the quality of the dissertation.

I have benefitted from the continued support of my employees at Meetinghouse Data Communications. They have been tolerant of my frequent absences due to work at the university and of my eccentricities that have accompanied this effort.

I thank my mother Ethelynne Goransson for giving me the desire and energy to pursue my dream of the doctoral degree. To my children, Jennifer and Peter, I can only say that I hope that the pride you feel can make up for the part of me you have given up so that I could accomplish this goal. I commit that time to you now. To Helen, my wife and friend, thank you for all the care and help over the last 20 years that has enabled me to realize this goal.

# TABLE OF CONTENTS

<b>Dedication</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xii</b>
<b>Abstract</b>	<b>xiii</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Thesis Goals.....	2
1.2 Quality of Service Parameters .....	3
1.3 Quality of Service Guarantees .....	4
1.4 Quality of Service Performance.....	5
1.5 The Role of Call Admission .....	6
1.6 The Role of Network Edge Policing.....	7
1.7 The Persistent Problem of Cell Loss.....	7
1.8 The Quality of Service Problem and Our Solution.....	12
1.9 Dissertation Road Map .....	13
1.10 References.....	13
<b>Chapter 2 Related Work</b>	<b>15</b>
2.1 Introduction.....	15
2.2 Distributing Schedules and Quality of Service State.....	15
2.3 Predicting Cell Arrivals .....	17
2.3.1 Burst-wise Reservation Approaches.....	17
2.3.2 Predictions for a Specific Traffic Source Type.....	18
2.4 Schedule Generation .....	20
2.4.1 Optimal Discard Policy .....	20
2.4.2 Delay Priority .....	21
2.4.3 Space Priority .....	21
2.4.4 Virtual Clock .....	22
2.4.5 MARS.....	22
2.4.6 Selective Packet Discard .....	23
2.5 References.....	23
<b>Chapter 3 Prediction-based Cell Scheduling: A Model</b>	<b>25</b>
3.1 Introduction.....	25
3.2 Cyclical Operation and Delay Buffers.....	26
3.3 Terminology .....	27

3.4	Statement of Problem and Goal .....	33
3.5	Traffic Types Supported.....	33
3.6	Fundamental Model Assumptions .....	34
3.6.1	Output Queued Switches .....	34
3.6.2	Only Arrivals from Neighbors Can Affect Congestion.....	35
3.7	Additional Assumptions .....	35
3.7.1	Regularity .....	35
3.7.2	Time Cycle Length .....	36
3.7.3	Computation Time for Scheduling .....	36
3.7.4	Variable Bit Rate Sources .....	36
3.7.5	Isochronous Service.....	36
3.7.6	Validity of the Prediction.....	37
3.7.7	Spatial Extent of the Prediction.....	37
3.7.8	Temporal Extent of the Prediction.....	37
3.7.9	Switch Fabric Delay .....	37
3.7.10	Switch Buffer Space .....	38
3.8	Distributed Cell Scheduling: A Model .....	38
3.8.1	Distributed Schedules .....	41
3.8.2	Predicting Cell Arrivals.....	41
3.8.3	Schedule Generation.....	42
3.9	Realizing an Accurate Cell Arrival Predictor: The Warning Shot.....	45
3.9.1	CBR Sources .....	46
3.9.2	VBR Sources .....	46
3.9.3	Determination of the Extent of the Prediction (PredictHorizon).....	49
3.9.4	Maintaining the Accuracy of the Prediction.....	49
3.10	Relaxation of Some Assumptions.....	49
3.10.1	Normalized ATM Networks .....	49
3.10.2	Determination of the Network Basic Time Unit .....	50
3.10.3	Realistic Distributed Scheduling: An Example.....	51
3.11	Extensions to the Model .....	55
3.12	References.....	55

## **Chapter 4 Accuracy of Predictions: Formal Proof 56**

4.1	Introduction.....	56
4.2	Overview of the Proofs .....	57
4.3	Network PredictState .....	57
4.3.1	Time-cycle Granularity.....	57
4.3.2	Notation .....	58
4.3.3	PredictComponents .....	59
4.4	Assumptions.....	67
4.5	The Restricted Prediction Theorem .....	68
4.6	Relaxed PredictHorizon .....	73
4.6.1	Impact on the Proof .....	73
4.6.2	PredictHorizon re: LinkScheduler .....	75
4.6.3	PredictHorizon re: AALSchedular .....	76
4.6.4	PredictHorizon re: Passive Components and PredictValidity .....	77
4.7	The Relaxed Prediction Theorem .....	79

<b>Chapter 5</b>	<b>Complexity of Prediction-based Scheduling</b>	<b>87</b>
5.1	Introduction.....	87
5.2	NP-completeness .....	88
5.2.1	Sequencing Within Intervals Problem (SWI).....	89
5.3	The Simple Quality of Service Scheduling Problem (SQSP).....	89
5.3.1	SQSP is NP-complete.....	90
5.4	The Complete Quality of Service Scheduling Problem (CQSP) .....	91
5.4.1	CQSP is NP-complete .....	92
5.5	Summary.....	92
5.6	References.....	93
<b>Chapter 6</b>	<b>Exposition of the Scheduling Algorithms</b>	<b>94</b>
6.1	Introduction.....	94
6.2	Baseline LinkScheduler .....	95
6.3	Expanded LinkScheduler .....	96
6.3.1	Allowing for Cell Queuing.....	96
6.3.2	Global Knowledge of QoS State .....	99
6.4	Heuristic LinkSchedulers.....	101
6.4.1	Predictive FIFO (PFIFO).....	101
6.4.2	Predictive FIFO with Displacement (PFIFO_D).....	102
6.4.3	Pseudo-Time Division Multiplexing (PTDM).....	102
6.4.4	Pseudo-Time Division Multiplexing with Displacement (PTDM_D).....	103
6.5	Implementation of the Heuristics.....	104
6.5.1	Virtual Circuit Control Block.....	104
6.5.2	Output Link Control Blocks (Neighbor Blocks) .....	109
6.5.3	Prediction Cells .....	114
6.6	Distance from Quality of Service Violation Metrics .....	116
6.6.1	Loss Cushion Depth .....	116
6.6.2	Time Since Last Cell Discard.....	117
6.6.3	Depth of Network Penetration.....	117
6.6.4	Multi-cell Packet Preservation .....	117
6.7	References.....	118
<b>Chapter 7</b>	<b>Results</b>	<b>119</b>
7.1	Introduction.....	119
7.2	Simulator.....	120
7.2.1	Traffic Source Types Supported .....	121
7.2.2	Link Types .....	122
7.2.3	Switch Characteristics .....	122
7.3	Simulated Network Configurations .....	123
7.4	Measuring Quality of Service Performance.....	125
7.4.1	Comparing the Heuristics: Ground Rules.....	125
7.5	CBR Jitter Guarantees Only .....	126
7.5.1	Simulation Description and Motivation .....	126
7.5.2	Jitter Violations.....	128
7.5.3	Upstream Delay Distributions .....	134
7.6	CBR Loss Guarantees Only (Local QoS Knowledge).....	138

7.6.1	Simulation Description and Motivation .....	138
7.6.2	Loss Violations .....	138
7.7	CBR Loss Guarantees Only (Global QoS Knowledge).....	140
7.7.1	Simulation Description and Motivation .....	140
7.7.2	Loss Violations .....	141
7.8	CBR Loss And Jitter Guarantees Together.....	144
7.8.1	Simulation Description and Motivation .....	144
7.8.2	Loss and Jitter Violations .....	144
7.8.3	Jitter Distributions .....	147
7.8.4	Observations.....	152
7.9	CBR/VBR: Loss and Jitter Guarantees Together.....	152
7.9.1	Simulation Description and Motivation .....	152
7.9.2	Overall QoS Violations.....	153
7.9.3	Jitter and Loss Violations Individually.....	155
7.9.4	Observations.....	159
7.10	Link Utilization.....	160
7.11	Throughput.....	163
7.12	Delay.....	166
7.13	Evaluation .....	169
7.14	References.....	171

## **Chapter 8 Future Work 172**

8.1	Intelligent Scheduling without Prediction .....	172
8.2	Prediction Cell Compression .....	173
8.3	Higher Level Protocol Packet Protection.....	173
8.4	ABR Flow Control Schemes.....	174
8.5	Phased VBR Sources .....	175
8.6	Incomplete Predictions .....	175
8.7	Multicast .....	176
8.8	Packet Networks .....	176
8.9	Global QoS Information About Cell Delay .....	176
8.10	Extending the Simulations in This Thesis .....	176
8.10.1	Comparison to Other Researchers' Jitter Results.....	176
8.10.2	Alternative Measures of DistanceFromQoSViolation.....	177
8.11	References.....	177

## **Chapter 9 Conclusion 178**

## **Appendix A Simulation Configurations 180**

A.1	Bottleneck_Jitter .....	180
A.1.1	Traffic Sources.....	181
A.1.2	Communications Links.....	182
A.1.3	Quality of Service Requirements of Sources.....	182
A.2	Bottleneck_Loss.....	183
A.2.1	Traffic Sources.....	183
A.2.2	Communications Links.....	184
A.2.3	QoS Requirements of Sources.....	184



A.3	Cross_Traffic_HomLink.....	184
	A.3.1 Traffic Sources.....	187
	A.3.2 Communications Links.....	188
	A.3.3 Quality of Service Requirements of Sources.....	189
A.4	Cross_Traffic_HetLink .....	189
	A.4.1 Traffic Sources.....	189
	A.4.2 Communications Links.....	189
	A.4.3 Quality of Service Requirements of Sources.....	189
A.5	Cross_Traffic_HetTraffic .....	189
	A.5.1 Traffic Sources.....	190
	A.5.2 Communications Links.....	190
	A.5.3 Quality of Service Requirements of Sources.....	190
	A.5.4 Switch Buffer Sizes .....	190
	A.5.5 Traffic Loading Level.....	191
	A.5.6 Overall Jitter Tolerance .....	191

**Appendix B Detailed Simulation Results 193**

B.1	Tables .....	193
B.2	Delay Distributions .....	217

**Appendix C Glossary of Acronyms 221**

# LIST OF FIGURES

FIGURE 1.	Network Utilization vs. QoS Violations .....	6
FIGURE 2.	No Congestion Scenario .....	9
FIGURE 3.	Inevitable Cell Loss Scenario .....	10
FIGURE 4.	Sample Network Topology .....	32
FIGURE 5.	Generic ATM Output Buffered Switch .....	34
FIGURE 6.	Statistical Multiplexing in a Single Switch Output Buffer .....	35
FIGURE 7.	Waves of Knowledge-need Preceding Waves of Arrivals .....	40
FIGURE 8.	Prediction Analysis Phase .....	43
FIGURE 9.	Arrival Schedule Repair Phase .....	45
FIGURE 10.	A PredictHorizon of Two Cycles .....	48
FIGURE 11.	Synchronizing Distributed Schedules: time -1 to + 2 .....	53
FIGURE 12.	Synchronizing Distributed Schedules: time +4 to +5 .....	54
FIGURE 13.	PredictState Network Components .....	61
FIGURE 14.	PredictState Network Components (detailed).....	62
FIGURE 15.	PredictHorizon of 3 and Warning Shot (W) Buffers.....	74
FIGURE 16.	Generalized PredictHorizon and Committed Schedules (S).....	75
FIGURE 17.	VciCTL Blocks in the Network .....	105
FIGURE 18.	The VciCTL Block .....	108
FIGURE 19.	Neighbor Blocks in the Network .....	109
FIGURE 20.	A Schedule for a Cycle .....	110
FIGURE 21.	Neighbor Blocks Detailed.....	111
FIGURE 22.	Ring of Schedules at Time i for Link K in Node Z. ....	112
FIGURE 23.	Ring of Schedules at Time i+1 for Link K at Node Z. ....	112
FIGURE 24.	Interrelationship of Structures and Scheduling Components (perspective: link xz) .....	114
FIGURE 25.	A Prediction Cell .....	116
FIGURE 26.	A 3-octet Prediction Cell Descriptor .....	116
FIGURE 27.	Traffic Flow Patterns for Bottleneck .....	124
FIGURE 28.	Traffic Flow Patterns for Cross_Traffic.....	124
FIGURE 29.	Jitter Tolerance.....	127
FIGURE 30.	CBR Connections Supported vs. Jitter Tolerance (PTDM, constant load) .....	130
FIGURE 31.	CBR Connections Supported vs. Jitter Tolerance (NORMFIFO, constant load) .....	130
FIGURE 32.	CBR Connections Supported vs. Jitter Tolerance (PTDM, increasing load) .....	131
FIGURE 33.	CBR Connections Supported vs. Jitter Tolerance (NORMFIFO, increasing load) .....	131
FIGURE 34.	Jitter at Host pp2a (PTDM).....	132
FIGURE 35.	Jitter at Host pp2a (NORMFIFO).....	132
FIGURE 36.	Jitter at Host pp2b (PTDM).....	133
FIGURE 37.	Jitter at Host pp2b (NORMFIFO).....	133
FIGURE 38.	CBR Time-in-Network Distribution at Link PP3 (PTDM).....	135
FIGURE 39.	CBR Time-in-Network Distribution at Link PP3 (NORMFIFO) .....	135
FIGURE 40.	CBR Time-in-Network Distribution at Link PP2a (PTDM).....	136
FIGURE 41.	CBR Time-in-Network Distribution at Link PP2a (NORMFIFO).....	136
FIGURE 42.	CBR Time-in-Network Distribution at Link PP2b (PTDM).....	137
FIGURE 43.	CBR Time-in-Network Distribution at Link PP2b (NORMFIFO).....	137
FIGURE 44.	Loss Violations vs. Loss Tolerance (PFIFO_D_NG, Bottleneck_Loss).....	140

FIGURE 45.	Loss Violations vs. Loss Tolerance (PFIFO_D, Cross_Traffic_HomLink).....	143
FIGURE 46.	Loss Violations vs. Loss Tolerance (PFIFO_D, Cross_Traffic_HetLink) .....	143
FIGURE 47.	Loss Violations, 90 cell buffer, PTDM_D vs. others .....	146
FIGURE 48.	Jitter Violations, 90 cell buffer, PTDM_D vs. others.....	146
FIGURE 49.	Jitter Violations, 9999 cell buffer, PTDM_D vs. others.....	147
FIGURE 50.	Jitter at Host 2b-1, Small Buffer (PTDM_D).....	148
FIGURE 51.	Jitter at Host 2b-1, Small Buffer (PFIFO_D).....	148
FIGURE 52.	Jitter at Host 2b-1, Small Buffer (PTDM) .....	149
FIGURE 53.	Jitter at Host 2b-1, Small Buffer (NORMFIFO).....	149
FIGURE 54.	Jitter at Host 2b-1, Large Buffer (PTDM_D).....	150
FIGURE 55.	Jitter at Host 2b-1, Large Buffer (PFIFO_D).....	150
FIGURE 56.	Jitter at Host 2b-1, Large Buffer (PTDM).....	151
FIGURE 57.	Jitter at Host 2b-1, Large Buffer (NORMFIFO).....	151
FIGURE 58.	Low Jitter Tolerance Overall QoS Violations .....	154
FIGURE 59.	Medium Jitter Tolerance Overall QoS Violations.....	154
FIGURE 60.	High Jitter Tolerance Overall QoS Violations .....	155
FIGURE 61.	Low Jitter Tolerance Jitter Violations Only .....	156
FIGURE 62.	Medium Jitter Tolerance Jitter Violations Only.....	156
FIGURE 63.	High Jitter Tolerance Jitter Violations Only .....	157
FIGURE 64.	Low Jitter Tolerance Loss Violations Only .....	157
FIGURE 65.	Medium Jitter Tolerance Loss Violations Only .....	158
FIGURE 66.	High Jitter Tolerance Loss Violations Only.....	158
FIGURE 67.	Link Utilization (%), Comparison of 4 policies, small buffer switches .....	162
FIGURE 68.	Link Utilization (%), Comparison of 4 policies, large buffer switches .....	162
FIGURE 69.	Throughput, Comparison of 4 policies, small buffer switches .....	165
FIGURE 70.	Throughput, Comparison of 4 policies, large buffer switches.....	165
FIGURE 71.	Time-in-system, pp2b-1, large buffer switch (PTDM) .....	168
FIGURE 72.	Time-in-system, pp2b-1, large buffer switch (NORMFIFO).....	168
FIGURE 73.	Bottleneck Simulation Topology .....	181
FIGURE 74.	Cross_Traffic Simulation Topology.....	186
FIGURE 75.	Time-in-system, pp3b, small buffer switch (PFIFO_D) .....	218
FIGURE 76.	Time-in-system, pp2b-1, small buffer switch (PFIFO_D).....	218
FIGURE 77.	Time-in-system, pp3b, small buffer switch (NORMFIFO) .....	219
FIGURE 78.	Time-in-system, pp2b-1, small buffer switch (NORMFIFO).....	219
FIGURE 79.	Time-in-system, pp3b, small buffer switch (PTDM) .....	220
FIGURE 80.	Time-in-system, pp2b-1, small buffer switch (PTDM).....	220

# LIST OF TABLES

TABLE 1.	No. of Cells Concurrently Propagating Down a Single ATM Hop.....	11
TABLE 2.	Scheduler Summary .....	102
TABLE 3.	Loss Tolerance for Bottleneck_Loss.....	184
TABLE 4.	Source Traffic Characteristics for Cross_Traffic_HomLink.....	188
TABLE 5.	Source Traffic Characteristics for Cross_Traffic_HetTraffic .....	192
TABLE 6.	Jitter Violations, constant load.....	194
TABLE 7.	Jitter Violations as a Percentage of Offered Traffic, constant load.....	195
TABLE 8.	Jitter Violations, increasing load.....	196
TABLE 9.	Jitter Violations as a Percentage of Offered Traffic, increasing load.....	197
TABLE 10.	Loss Violations, Local QoS Knowledge, Bottleneck_Loss .....	198
TABLE 11.	Loss Violations, Global QoS Knowledge, Cross_Traffic_HomLink, small buffer .....	199
TABLE 12.	Loss Violations, Global QoS Knowledge, Cross_Traffic_Hetlink, small buffer.....	200
TABLE 13.	Loss Violations, 90 cell buffer, PTDM_D vs. others .....	201
TABLE 14.	Loss Violations, 9999 cell buffer, PTDM_D vs. others .....	202
TABLE 15.	Jitter Violations, 90 cell buffer, PTDM_D vs. others.....	203
TABLE 16.	Jitter Violations, 9999 cell buffer, PTDM_D vs. others.....	204
TABLE 17.	Low Jitter Tolerance Overall QoS Violations .....	205
TABLE 18.	Medium Jitter Tolerance Overall QoS Violations.....	206
TABLE 19.	High Jitter Tolerance Overall QoS Violations .....	207
TABLE 20.	Low Jitter Tolerance Jitter Violations Only .....	208
TABLE 21.	Medium Jitter Tolerance Jitter Violations Only.....	209
TABLE 22.	High Jitter Tolerance Jitter Violations Only .....	210
TABLE 23.	Low Jitter Tolerance Loss Violations Only .....	211
TABLE 24.	Medium Jitter Tolerance Loss Violations Only .....	212
TABLE 25.	High Jitter Tolerance Loss Violations Only.....	213
TABLE 26.	Link Utilization (%), Comparison of 4 policies, small buffer switches .....	214
TABLE 27.	Link Utilization (%), Comparison of 4 policies, large buffer switches .....	214
TABLE 28.	Throughput, Comparison of 4 policies, small buffer switches .....	215
TABLE 29.	Throughput, Comparison of 4 policies, large buffer switches.....	215
TABLE 30.	Raw Cell Loss vs. Cell Loss Violations.....	216
TABLE 31.	Components of Additional Delay Due to Prediction-Based Scheduling (in cell times)..	216

## **ABSTRACT**

### **DISTRIBUTED CELL SCHEDULING AND QUALITY OF SERVICE IN ATM NETWORKS**

**by**

**Paul Goransson**

**University of New Hampshire, December, 1995**

This work investigates the improvement of network jitter and quality of service (QoS) in ATM networks through the use of more complex scheduling algorithms than have been heretofore presented. In particular we discuss the measure of QoS Performance as distinct from traditional measures of network performance. Results of the study demonstrate that with the judicious introduction of delay at strategic places in an ATM network, specifically at the network periphery and lesser amounts elsewhere, we can generate and distribute accurate predictions of cell arrivals to all queues within the entire network. With the existence of such predictions, we are able to apply more complex scheduling algorithms than would normally be feasible. This approach permits the specification and control of QoS parameters on a per-connection basis rather than compressing all connections into a small number of QoS "classes." We show that more complex scheduling algorithms can produce significant improvements in QoS performance, especially related to jitter. These ideas are investigated through simulations, comparing the QoS performance for different implementations of prediction-based cell scheduling with the performance of an ATM network performing strict FIFO cell scheduling.

# Chapter 1

## INTRODUCTION

The world-wide communications infrastructure is undergoing a revolution due to the proliferation of extremely high bandwidth fiber optic communications trunks. The existence of this high bandwidth is spawning tremendous growth in the variety and volume of communications services. This growth has motivated increasing interest in high-speed hybrid networks, commonly known as Broadband Integrated Services Digital Networks (B-ISDN). Associated with the growth of these networks is increased concern about quality of service, including such factors as data loss and delay. Research has been presented on a number of approaches to address these concerns. In this thesis, we present a novel approach to improve B-ISDN performance which attempts to exploit more of the capabilities inherent to these networks than other work presented to date.

One of the transfer modes possible for B-ISDN is the Synchronous Transfer Mode (STM), which is based on Time Division Multiplexing (TDM) technology. TDM-based networks can readily support the variety of services demanded in hybrid networks, but with poor network utilization.

Packet switching technologies like X.25 have traditionally held an advantage over TDM technologies due to higher network utilization achieved through statistical multiplexing. Such packet switching techniques, however, are not scalable to high transmission speeds. Asynchronous Transfer Mode (ATM), using available technology, is thought to offer the high network utilization of packet switching with the capability to sustain end-to-end network throughput in the gigabit/second range. ATM has been chosen by the ITU (formerly known as the CCITT) as the *transfer mode of choice* for B-ISDN.

ATM cannot escape, however, the networking axiom that utilization gains from statistical multiplexing must always come at the expense of increased delay or loss. It is significant that the very high bandwidth services which are currently used to justify ATM have particularly rigid requirements on delay and loss. These rigid requirements, formally expressed as Quality of Service (QoS) requirements, must therefore be traded off against high network utilization. When a user requests a connection stipulating guarantees related to delay or loss, the request will be accompanied by specific QoS parameters.

## **1.1 Thesis Goals**

Considerable research has been done in the application of call admission, flow control, and rate-based policing in pursuit of this dual goal of QoS guarantees and high utilization. While all of these mechanisms can be effective, it is widely agreed that further improvements in the utilization/QoS violation quotient will require intelligent scheduling of cells in the network switches themselves. Earlier research on cell scheduling has focussed on relatively simple scheduling decisions due to the perception that more powerful (and hence more complex) scheduling decisions are not feasible at B-ISDN speeds.

Our research develops a model that permits more sophisticated scheduling in ATM switches through the use of predictions about future cell arrivals. We show how this produces improvements in performance not possible with simpler scheduling techniques. The scheduling computation relies on the model's ability to provide accurate short-term predictions about impending cell arrivals. We show how this foreknowledge of cell arrivals provides adequate time for individual switches to make decisions that better satisfy QoS requirements for individual connections while also increasing overall network utilization.

For constant bit rate (CBR) sources, predictions are trivial to produce. For variable bit rate (VBR) sources, insertion of small delay buffers at the network edge permits "warning shot" predictions to propagate through the network ahead of the actual data. Our use of predictions differs sharply from other reported attempts to use prediction, since our predictions are accurate forecasts and not estimates.

The presence of this foreknowledge provides the opportunity to investigate different aspects of intelligent cell scheduling. In particular, we wish to derive benefits from the following areas distinguishing the concept of QoS violations from the simpler but less meaningful concepts of cell loss and delay:

1. providing global information about the current QoS state to all switches in a connection's path so that local QoS decisions can have relevance to the end user QoS,
2. examining the benefits of controlled buffering of cells to increase utilization without the loss of tight control of QoS that is normally associated with queuing systems.

Therein lies the primary thrust of this thesis: we intend to provide accurate predictions of the future sufficiently in advance of the actual arrival of cells to allow for complex computations that generate a (nearly) optimal schedule. The schedule provides QoS guarantees on a per-connection basis. In this thesis we illustrate how the predictions can be made, and propose scheduling heuristics to address the inherent optimization problem.

Additionally, since our goal is to minimize QoS violations, and QoS is defined at the network traffic boundary, it is senseless to make locally optimal decisions about QoS in the absence of knowledge about cell loss that may be occurring at other nodes. We suggest that any solution to the QoS problem ultimately include global knowledge of the QoS state of each connection. Implementation of this global knowledge falls out conveniently from the transmission of predictions, and thus nicely complements the prediction-based cell scheduler by providing a means to improve QoS guarantees in ATM networks.

## 1.2 Quality of Service Parameters

Individual user end-to-end ATM connections are known as virtual circuits. They are *virtual* circuits in the sense that they provide logical connectivity across the network over potentially many individual physical circuits aligned in a tandem fashion. We identify the following QoS parameters on a per-virtual circuit basis:

1. average bandwidth
2. peak bandwidth
3. loss tolerance
4. jitter tolerance
5. delay tolerance
6. measurement interval length



While these terms or variations of them appear in other research, they have no universally accepted interpretation. Therefore we now provide the interpretations that we use in this thesis.

*Peak* (average) bandwidth describes the *maximum* (average) number of bits per second that the user of the virtual circuit may inject at the transmitter network boundary. *Loss tolerance* represents how many lost cells the connection is willing to tolerate over a specified *measurement interval*. *Delay tolerance* stipulates the maximum delay a cell from the virtual circuit can suffer without becoming useless at the receiver network boundary. *Jitter tolerance* describes the maximum deviation from a stipulated inter-cell time that may be tolerated at the receiver network boundary before the cell becomes useless to the receiver. Note that unlike loss tolerance, the definitions of delay and jitter tolerance do not include a notion of measurement time interval. We refer the reader to [6] for an expanded exposition of the basic terminology used in this thesis.

The terms defined above yield a natural classification of user traffic types into the four major classes: CBR, VBR, ABR and UBR. *Constant Bit Rate* (CBR) is traffic where peak bandwidth equals average bandwidth. *Variable Bit Rate* (VBR) is traffic in which peak bandwidth exceeds average bandwidth. Both CBR and VBR are usually considered jitter-sensitive and delay-sensitive, and somewhat loss-tolerant. *Available Bit Rate* (ABR) traffic is loss-sensitive and delay-tolerant. *Unspecified Bit Rate* (UBR) traffic is tolerant of both loss and delay. Peak bandwidth is greater than average bandwidth for both ABR and UBR types.

### 1.3 Quality of Service Guarantees

We define providing QoS guarantees as the forwarding of the virtual circuit's cells from the transmitter network boundary to the receiver network boundary at a throughput of up to *peak-bandwidth* bits per second such that its stipulated loss, jitter or delay tolerances are not exceeded at the receiver. Although the field of QoS research is new enough so that there is no widely accepted formal definition of "QoS Guarantees," the preceding definition is in accordance with the use of the term in the research literature (see Chapter 2). An important factor in our research is that QoS guarantees need to be managed on a per-connection basis. For example, one connection may specify that it can tolerate only a single loss per 100 ms, but it may have no special requirements related to delay. Another connection may indicate that it can tolerate no loss whatsoever and end-to-end delay of no more than 50 ms. A third connection may stipulate that it can tolerate 100 losses per second, but

that the variance in end-to-end delay between successive cells must not exceed 80  $\mu$ s. For each of these examples, providing the QoS guarantee has a different manifestation in the ATM switch. In all three cases, though, it is clear what quantity of loss or delay constitutes a violation of a QoS guarantee. When the network accepts a connection request, it enters into a contract with the user that it will, with definable probability, honor the requested guarantees.

While delay and loss are the basic measures of QoS, a number of other QoS measures may be derived from them. For more details on the issues related to providing QoS guarantees in high speed hybrid networks, see [6], [11], and [12].

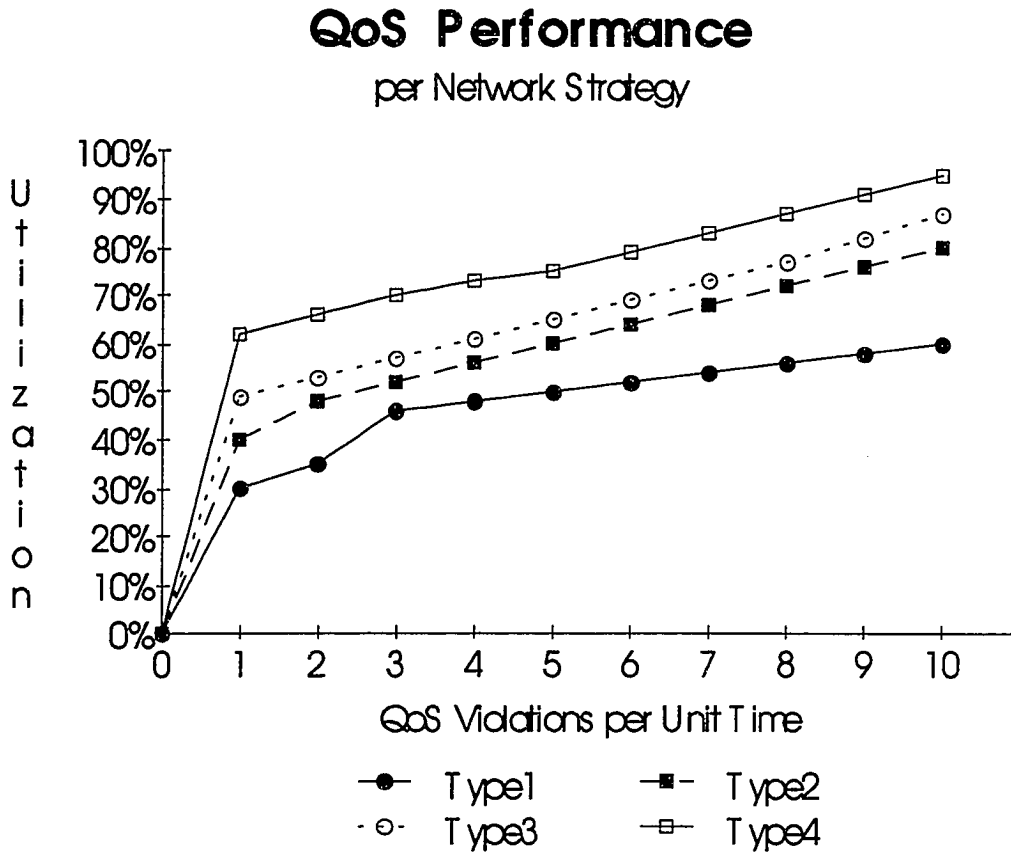
#### **1.4 Quality of Service Performance**

ATM has been chosen as the transfer mode of choice for B-ISDN largely due to the belief that it will offer a better utilization/QoS violation quotient than either TDM or packet switching. Even ATM cannot escape the correlation between increased network utilization and QoS violations. This reality is in fact a characteristic of any queueing system with stochastic traffic sources. Due to the varying load generated by the traffic sources, a queueing system with finite buffers cannot be run at 100% capacity without a non-zero probability of queue overflow [13].

This leads us to the frequent characterization of an individual ATM network implementation with graphs that plot network utilization vs. cell loss violations. Such plots are commonly presented in current ATM research papers (e.g., [3], [9], and [7]). We suggest that a more useful version of this plot is the network utilization vs. QoS violation. Such a plot for four different hypothetical ATM network implementations is shown in Figure 1. The four different implementations are depicted as type 1, type 2, type 3 and type 4 in the figure. Figure 1 clearly indicates that implementation type 4 achieves consistently higher utilization levels for all the QoS violation levels shown. Note that comparison of two or more curves on such a plot is only valid when the traffic characteristics and traffic distribution patterns are consistent for all curves at any point on the utilization axis.

Since an ideal ATM network would permit 100% utilization with zero QoS violations, it is generally desirable that the QoS performance curve rise as rapidly as possible. This curve can be improved by a variety of mechanisms. We review some of these in the following sections. This review shows that while QoS performance can benefit from these mechanisms, further enhancements are necessary.

FIGURE 1. Network Utilization vs. QoS Violations



### 1.5 The Role of Call Admission

We assume that call admission includes the function of allowing or disallowing the establishment of a given virtual connection as well as the selection of a path through the network for that connection. If all traffic sources were isochronous, then call admission would be the only factor in providing QoS guarantees and in fact would be a relatively straightforward accounting problem. Such a network would not really be an ATM network at all, but a TDM (also known as STM (*Synchronous Transfer Mode*)) network with its inherently lower utilization. All traffic sources are *not* isochronous, though, and support of VBR traffic is one of the key facets of the ATM network that provide it with an advantage over its TDM counterpart. VBR sources have a higher peak bit rate than average bit rate. *Perfect* QoS guarantees require *true* peak bit rate resource reservation [4], which ATM cannot widely apply and still retain an advantage over STM. This forces ATM call admission into a complex trade-off between network utilization and probability of QoS violations. Call admission is forced to *overbook* network resources to provide acceptable

utilization levels. By *overbook* we mean that the call admission process accepts more connections than can be supported if all connections simultaneously transmit at their peak rate.

In the face of mixed traffic types, the call admission process needs to evaluate how much it may overbook resources (communications bandwidth and switching) and still meet the QoS requirements of both the incoming connection request as well as the set of existing connections. Since call admission cannot see into the future and contemplate the combined traffic loads of all the sources over the durations of all connections, this problem has no deterministic solution at call admission time. The call admission implementation will be a heuristic tuned either for low utilization and low cell loss probability or for increased utilization with increased cell loss probability.

We make the assumption that, in any practical network, call admission *will* overbook resources and, therefore, cell loss is inevitable.

## 1.6 The Role of Network Edge Policing

Another mechanism that can improve the network utilization/QoS violation curve is network edge policing. The best known example of this is the *Leaky Bucket* policing mechanism. This mechanism forces either the average bandwidth or the peak bandwidth of the source to stay within some negotiated maximum. While this approach is popular and has an important role in determining the network utilization/QoS violation curve, it is generally accepted [5][8][2][1] that it cannot preclude undesirable cell loss within the network. This loss can result from the aggregation of groups of individually policed sources and variations in queuing and transmission delays for different virtual connections.

## 1.7 The Persistent Problem of Cell Loss

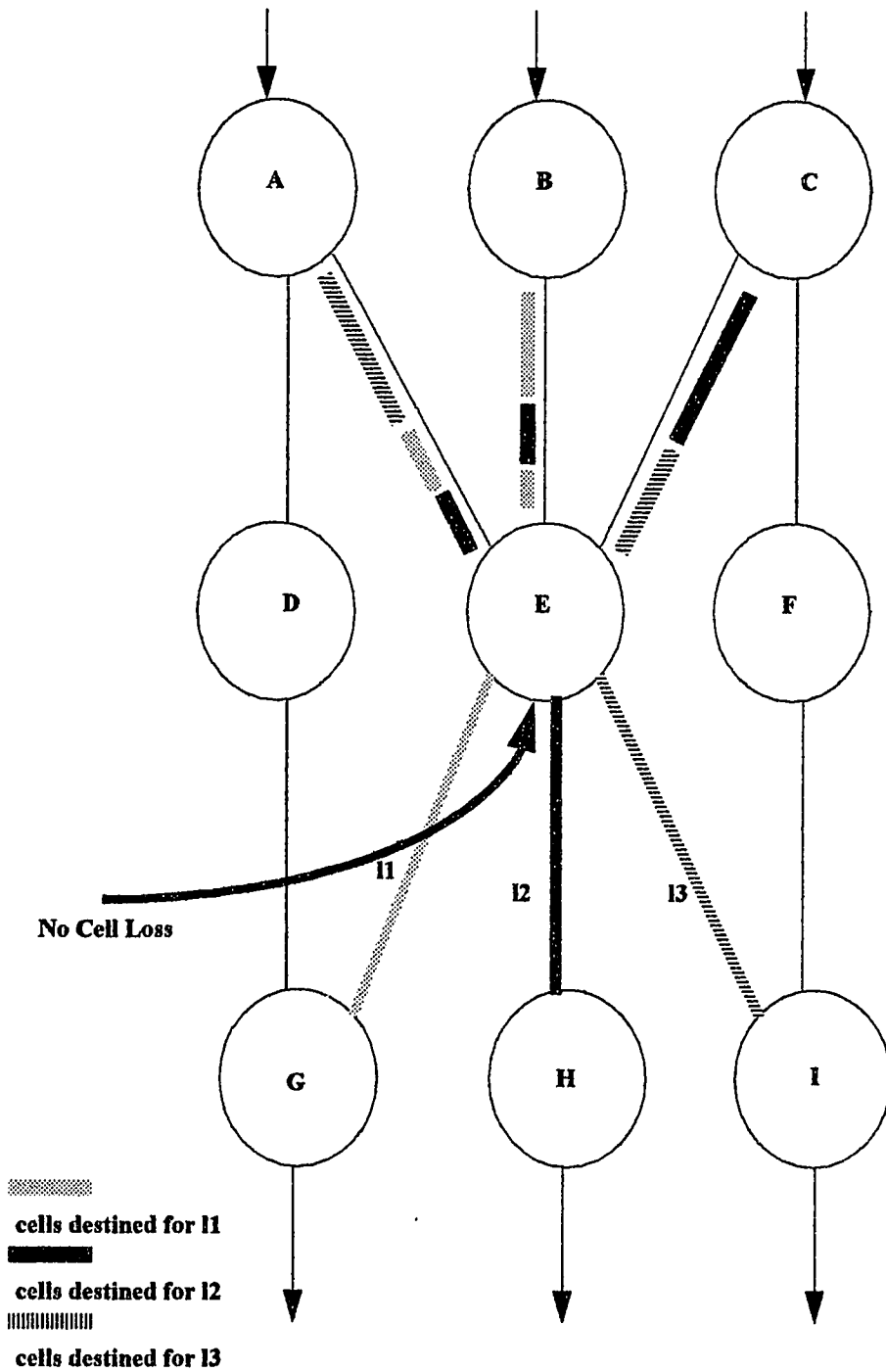
Even with a good call admission process and with network edge policing, cell loss internal to the network still occurs. Figure 2 shows a network of 9 ATM switches (A-I) interconnected by communications links. For simplicity we show all network traffic moving from top to bottom in the figure. The shaded bars depict a sample distribution of cells on the upstream links of switch E, according to their respective egress link *l1*, *l2* or *l3*. While the cell distribution shown is more coarse-grained than is likely to occur in reality, it serves to illustrate the point that as these cells arrive at switch E, they do not cause congestion at the servers for links *l1*, *l2* or *l3*. This is because they happen to be chronologically ordered so as to minimize the likelihood of congestion. Only a TDM system, though, can truly guar-

antee such chronological ordering. Support of variable rate sources precludes an explicit guarantee of this ordering. Since ATM cannot completely control the chronological ordering of arrivals, it is probable that cells will sometimes arrive in a chronologically inconvenient fashion, thereby producing congestion.

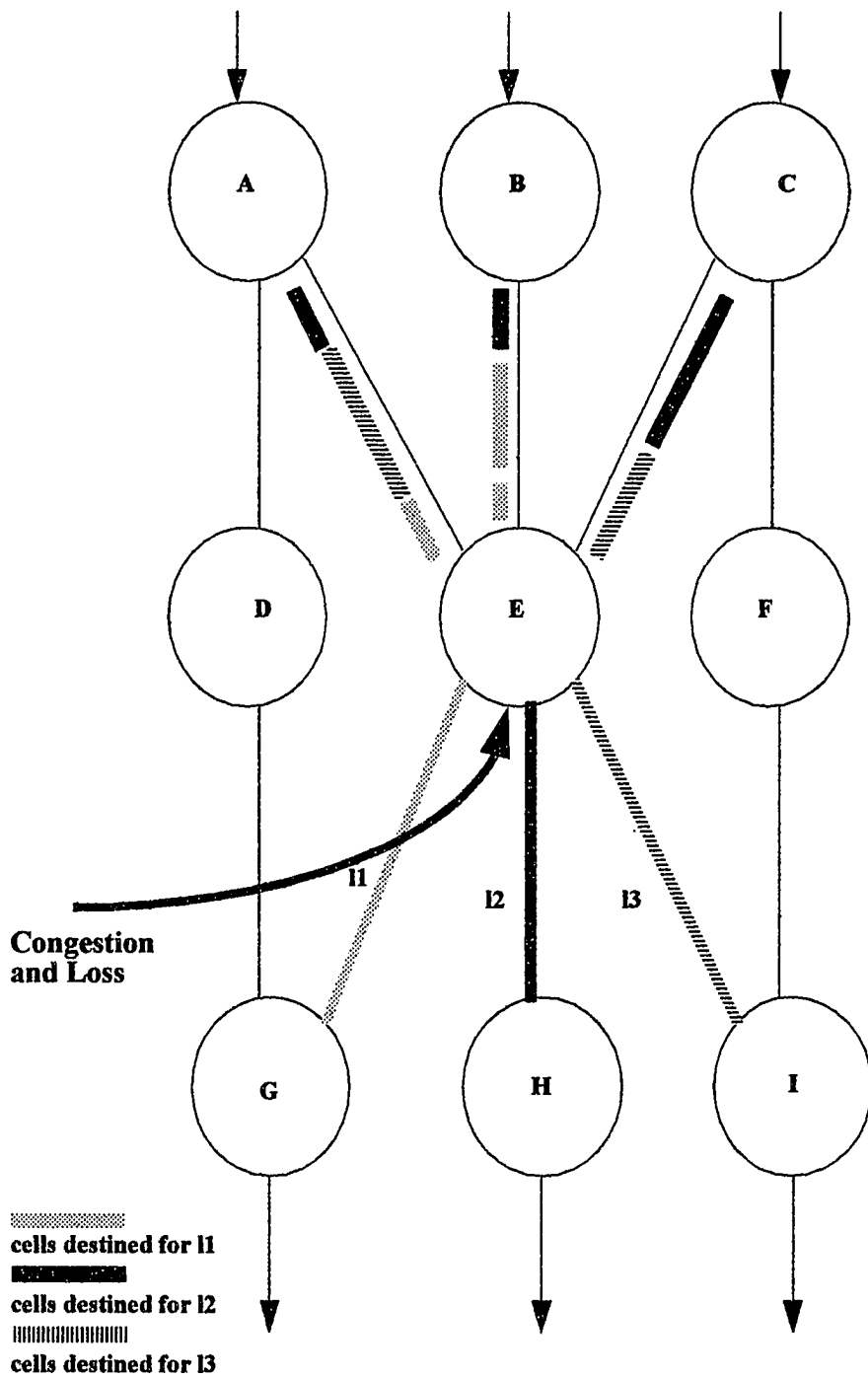
We depict such an ordering in Figure 3. It is clear that too many cells destined for link *l2* arrive during the same time period. The packet switching solution of allowing the excess bandwidth arrivals to queue at the link *l2* server is not directly applicable in ATM. This is because the cell arrival rate can be so high that the number of cells arriving during a period of congestion could far exceed any reasonable buffer capacity of the switch. Unlike traditional packet switching networks, the bandwidth-delay product of ATM is so high as to prohibit reliance on the end-to-end flow control methods that those networks have used successfully to control network queue overflow. The notion of *bandwidth-delay product* is discussed in detail in [6]. For an intuitive notion of what high bandwidth-delay product signifies, see Table 1. This table provides a tally of how many ATM cells are propagating down fiber optic trunks of various lengths for the three proposed ATM link speeds. The values in Table 1 were calculated assuming a propagation delay of 5085 nanoseconds/kilometer, which is characteristic of the optical fiber used for the Fiber Distributed Data Interface (FDDI) standard [14]. Thus, if we assume that the internodal links in Figure 3 are 1000 kilometer fiber optic links clocked at 155 MBps, approximately 2000 cells will be in transit on each link at any time. The period of congestion for link *l2* corresponds approximately to the last 25% of the cells in transit on the three upstream links. This implies that during a time interval in which *l2* can transmit 500 cells, there are 1500 arrivals. This would require a per-link output buffer for 1000 cells, which is quite large. This problem worsens linearly with an increase in link speeds. Since 156 Mbps is the lowest link speed proposed for ATM, with alternate speeds of 622 MBps and 2.5 GBps, it is widely agreed [4] [10] that cell buffering will not be adequate to prevent cell loss during congestion.

Since some cell loss is inevitable during these brief periods of congestion, the best one can hope for is to orient cell loss in such a way that no virtual connection's QoS guarantees are violated. The intent of this research is to propose and evaluate a means to improve the network's chances of meeting all QoS guarantees while maintaining high network utilization. In order to propose how this may be achieved, we first recap the problem in the next section.

**FIGURE 2. No Congestion Scenario**



**FIGURE 3. Inevitable Cell Loss Scenario**



**TABLE 1. No. of Cells Concurrently Propagating Down a Single ATM Hop**

<b>Hop Length (km)</b>	<b>OC-3 156 MBps</b>	<b>OC-12 622 MBps</b>	<b>OC-48 2.5 GBps</b>
Campus (10)	18.6	74.6	299
Metropolitan (100)	187	746	2985
Inter-city (1000)	1866	7464	29859
Transcontinental (5000)	9331	37324	149296



## 1.8 The Quality of Service Problem and Our Solution

We define the QoS problem in this section and illustrate it using Figure 3. During a period of congestion such as that evidenced for link  $l_2$  in Figure 3, an optimal link schedule is one that drops cells as necessary and reorders the transmission of the remaining cells so as to minimize the QoS violations incurred within that congested period. *Finding and enforcing this optimal schedule is the focus of this thesis.*

While we acknowledge that the optimal schedule may be difficult to achieve in practice, we postulate that this is merely a problem of computational complexity. Stated simply, if there were enough time to compute the ideal schedule, then such a schedule could be computed and enforced. One could simply compute all the possible schedules and then enforce the one with a minimum number of QoS violations.

Earlier work has dealt with a simpler version of the QoS problem, assuming that all connections fall into a small, fixed number of QoS classes. This restriction is placed on how QoS parameters may be stipulated precisely in order to render the scheduling problem less computationally intense. The combinations of possible values for the six QoS parameters are so diverse that we reject as overly restrictive the notion that QoS guarantees can be managed on the basis of a small number of classes into which each connection must be placed. Our solution provides these guarantees on a per-connection basis.

This thesis is motivated by the idea that a future transmission schedule can be calculated by looking upstream at the set of cells flowing towards the link in question. By judiciously introducing a small delay in the network, we can let information about these upstream cells precede the arrival of the cells themselves. We refer to the regular propagation of this information about future arrivals as *schedule distribution*. An essential side effect of computing a schedule based on predicted input is that this computation can be performed in parallel with the cell flow. That is, while we are computing a schedule for a future time period, the cells of the current time period flow through the switch unaffected by the ongoing schedule computation. We demonstrate that such use of the predictions can provide sufficient time to compute the (nearly) ideal schedule so that it may be enforced when the cells to which it pertains arrive.

In other words, in this thesis we provide accurate predictions of the future arrival sufficiently in advance of the actual arrival of cells to allow for complex computations that generate a (nearly) optimal schedule. The schedule provides QoS guarantees on a per-connection basis. We illustrate how the predictions can be made and propose scheduling heu-

ristics to address the inherent optimization problem. We also suggest that any solution to the QoS problem ultimately include global knowledge of the QoS state of each connection. Implementation of this global knowledge falls out conveniently from the transmission of predictions, and thus complements the prediction-based cell scheduler in providing a means to improve QoS guarantees in ATM networks.

## 1.9 Dissertation Road Map

Having provided basic definitions and an introduction to our research, it is now appropriate to give a more detailed outline of the thesis. Chapter 2 provides a review of current research related to the QoS problem. In Chapter 3 we introduce a prediction-based scheduling model that we use throughout the balance of the dissertation. The model provides a theoretical framework that aids us in defining specific, implementable heuristics addressing the QoS problem. Chapter 4 presents a formal proof that the predictions asserted to be universally correct in the model are indeed always correct. In Chapter 5 we discuss the computational complexity of our scheduling problem as defined in the model. We show that this problem is *NP*-complete. The intractability of the problem leads us to develop efficient heuristics for it. The approximations are practical and calculate sub-optimal but good schedules. They are described in Chapter 6. In Chapter 7 we select specific heuristics from those presented in Chapter 6 and perform extensive simulations of them running in different ATM network topologies with numerous traffic types under various loads. Numerical and graphical results of these simulations are compared to those from simulations of a generic ATM network for the purposes of evaluating our heuristics. Chapter 8 discusses several different research avenues that could be pursued by extending the work in this thesis. In Chapter 9 we conclude with a summary of our results and the implication that these results have on continuing research in the field of QoS guarantees in ATM networks.

## 1.10 References

- [1] J. Ameyo, A. Lazar, and G. Pacifici. Cooperative distributed scheduling for ATM-based broadband networks. In *IEEE INFOCOMM 92*, volume 1, pages 333–342, June 1992.
- [2] V. Friesen and J. Wong. The effect of multiplexing, switching and other factors on the performance of broadband networks. In *IEEE INFOCOMM 93*, volume 3, pages 1194–1203, March 1993.
- [3] G. Gallasi, C. Cappellini, L. Fratta, G. Rigolio, and F. Rossi. Performance

- evaluation of flow control mechanisms in multi-throughput class networks. In *IEEE Globecom 88*, volume 1, pages 511–516, June 1988.
- [4] J. Gechter and P. O'Reilly. Standardization of ATM. In *IEEE Globecom 88*, volume 1, pages 107–111, June 1988.
  - [5] S. Golestani. Congestion-free transmission of real-time traffic in packet networks. In *IEEE INFOCOM 90*, volume 1, pages 527–536, June 1990.
  - [6] P. Goransson. Bandwidth allocation in hybrid networks. Technical Report TR-93-02, University of New Hampshire, February 1993.
  - [7] R. Guerin and L. Gun. A unified approach to bandwidth allocation and access control in fast packet-switched networks. In *IEEE INFOCOMM 92*, volume 1, pages 1–12, June 1992.
  - [8] P. Guillemin, P. Boyer, A. Dupuis, and L. Romoef. Peak rate enforcement in ATM networks. In *IEEE INFOCOMM 92*, volume 2, pages 753–758, June 1992.
  - [9] J. Le Boudec. About Maximum Transfer Rates for Fast Packet Switching Networks. In *ACM SIGCOMM 91*, pages 295–304, September 1991.
  - [10] L. Trajkovic. Buffer requirements in ATM networks with leaky buckets, FIFOs, and weighted fair queueing scheduling mechanisms. ATM Forum 94-0077, January 1994.
  - [11] P. Goransson. ST2 and Resource Reservation. In *Proceedings of the First Annual Conference on Telecommunications R&D in Massachusetts*, volume 5, pages 38-49, October 1994.
  - [12] P. Goransson. Bandwidth reservation on a commercial router. *Computer Networks and ISDN*, volume 28, no. 3, pages 351-370, January 1996.
  - [13] L. Kleinrock. *Queueing Systems Volume II: Computer Applications*. Wiley, New York, 1976.
  - [14] ANSI X3.148-1987. Fiber Distributed Data Interface (FDDI) - Token Ring Physical Layer Protocol.

## **Chapter 2**

# **RELATED WORK**

### **2.1 Introduction**

In this chapter we review recent research related to our thesis. Our primary focus is to generate nearly optimal schedules based on cell predictions received from neighbors. In the next section of this chapter, we discuss other work related to the interchange of cell predictions between neighbors. The generation of the cell predictions themselves is a research topic in its own right, and we provide references and a brief discussion of that work in the second section. Related research in the area of schedule generation is the focus of the last section.

### **2.2 Distributing Schedules and Quality of Service State**

The distributed cell scheduling described in this thesis differs fundamentally from the related work on distributed schedules in a number of ways. Other work in this area has been based on grouping all connections into a small, fixed number of classes. Each connection in a class is presumed to have identical QoS requirements to other members of that class. We argue that QoS is a connection specific commitment and, hence, cannot be managed on the basis of a small number of traffic classes. Another fundamental difference is that the other research ignores the end-to-end aspect of QoS and instead focuses on QoS issues in a single network queue at a single switch.

The general idea of prediction of node-state and exchange of this information with neighbors in order to deal with congestion in high-speed networks has appeared in recent work by Amenyo [1] [2] and Ko [10]. Ko provides prediction-based feedback to neighbors to

alleviate congestion, but that work only considered a single class of traffic (Poisson) and ignores the QoS problem [10]. Amenyó extends the prediction-exchange paradigm of Ko to include three distinct QoS classes [1]. The entire scheduling approach in [1] is tightly coupled with the definition of the three classes. While providing QoS guarantees in accordance with a small number of classes is consistent with the current literature, and has clear implementation advantages, we argue that QoS guarantees can only be adequately provided on a connection-specific basis. Gupta and El-Zarki recognize the problem of over-generalizing connections' QoS requirements into too few classes [6]. In [6], they attempt to determine the minimum set of traffic classes that satisfy the QoS constraints of a set of connections. Gupta and El-Zarki cite an example that supports the idea that QoS needs to be specified and enforced on a per-connection basis. The example dispels the intuition that choosing the tightest QoS guarantees of a set of sources as the QoS guarantees for all should be sufficient to guarantee the requirements of all of them. They show this to be false, basically because the cell loss probabilities of individual streams are not necessarily equal to the overall cell loss probability. In layman's terms, if you are serving 10 streams, one of which is extremely bursty, and 9 smooth, the overall loss probability may be within limits, but the bursty stream will suffer a disproportionate amount of that loss. Despite the fact that Hyman's work is limited to three traffic classes, he acknowledges that QoS should ideally be guaranteed on a connection basis and only discounts this because of its evident computational complexity [7]. Despite the computational complexity involved, we argue that since QoS is a connection specific commitment, it cannot be adequately controlled using only a small number of traffic classes.

Because of the over-simplification of QoS into 3 classes, Amenyó is able to treat cell loss in a very straightforward manner [1] [2]. Cell loss is either to be avoided at all costs, or there is no penalty for loss, depending on the particular class. It is well known that some of the most important traffic types predicted to use ATM are tolerant of small amounts of loss [5]. The concentration of this loss is also significant. This thesis incorporates these factors as well as others into the notion of QoS violations, which is a much more relevant concept at the user traffic boundary than cell loss.

Similarly, since QoS is defined at the network traffic boundary, it is senseless to make locally optimal decisions about QoS in the absence of knowledge about cell loss that may be occurring at other nodes. The incorporation of this knowledge in traffic scheduling decisions is discussed extensively in Section 6.3.2, "Global Knowledge of QoS State." This factor is ignored by Amenyó [1] [2] and Ko [10] and, surprisingly, *we have seen no*

*other research on QoS optimization that accounts for the need for information about cell loss that may be occurring elsewhere in the network.*

While the preceding points are all significant, the most striking difference between the prediction and scheduling proposed by Amenyó and that proposed here is the fact that our predictions are 100% accurate forecasts of the near-term future, whereas Amenyó's are probabilistic predictions. Our forecasts are emitted and received on a highly regular basis as a function of the network topology. The accuracy and timeliness of the information eliminates the stochastic characteristics exhibited by [1].

## **2.3 Predicting Cell Arrivals**

Cell prediction schemes are usually based on making a probabilistic 'guess' about future performance based on past history. In some cases, attributes of the particular traffic source can render such guesses more accurate than the general case. The prediction scheme we propose in this thesis is unusual in that the predictions are completely accurate and work for any traffic type. This increased accuracy and generality is achieved at the price of increased network latency. Our predictions are distinguished from network edge-only predictions in that they are generated by each network hop, and their scope is limited to a single network hop. As we do not expect any explicit response or acknowledgment to our predictions, they do not become obsolete with increasing network size.

In Section 1.5 we described the role of call admission in providing QoS guarantees. One of the problems encountered in developing effective call admission strategies is that the granularity of the call parameters is too coarse. That is, as we indicate in [5], it is difficult to describe accurately the behavior of an entire connection with a few parameters such as  $E$  (*average bit rate*),  $S$  (*maximum bit rate*), or the standard deviation of the probability distribution function of such variables. In other words, for most connections, it is impossible to predict their behavior at connection establishment time. It is possible to describe a connection as a series of bursts interspersed with periods of relative inactivity. Normally, traffic behavior will vary less within a surge burst than it does over the course of the entire connection. Clearly, then, much more accurate traffic descriptions can be made if the traffic being described is just a single burst.

### **2.3.1 Burst-wise Reservation Approaches**

Burst-wise reservation usually induces considerable latency into the traffic stream. This is because the delay penalty paid for each reservation request, normally restricted to connec-

tion establishment, is now paid for each burst of the connection. One burst-reservation system, described by Gerla and Tai, addresses the problem of connectionless service over ATM [4]. Traditional approaches to this connectionless service problem are reviewed, including the virtual path approach and the connectionless server approach.

Gerla and Tai propose a *bandwidth advertisement* approach where the non-reserved bandwidth is continually advertised to the network edge. The NAL (see [5]) employs this almost-accurate description of available network bandwidth to time transmissions so that there is a high probability that the required bandwidth is available on the desired path. Gerla and Tai describe a bandwidth probing technique to achieve the bandwidth advertisement. *Burst reservations* precede the data burst to temporarily allocate bandwidth for the burst. The burst reservations in [4] differ significantly from the predictions in this thesis:

1. They generate reservations at the network edge only; we do so both at the edge and at each switch.
2. Their reservations are not generated at uniform time intervals; ours are.
3. In our paradigm the ‘reservations’ are really ‘warnings.’ There is no implication here of any guarantee that the warning must be heeded. In [4] it was implied that it would be obeyed.
4. Our paradigm does not require that bandwidth advertisements propagate to the network edge. As the size of the network increases, the bandwidth advertisements tend to be obsolete as they reach the network boundary.
5. In our approach, predictions about bandwidth usage are only exchanged between neighbors. Since they can only suffer the latency of a single network hop, our approach scales better with increases in network size than does [4].

Further analysis would be required to evaluate the recency/overhead trade-off of the information gathering/dissemination process that underlies their ideas.

### **2.3.2 Predictions for a Specific Traffic Source Type**

#### **MPEG Example**

When the predictions only need to be made for a specific traffic source type, such as Motion Picture Experts Group (MPEG) video, special source characteristics can facilitate intelligent predictions. Pancha and El-Zarki use and describe a predictor to estimate the next MPEG frame’s bandwidth requirements [12]. The predictor is a model where the

number of states of that model is determined by the ratio of peak rate to the standard deviation of cell arrivals. They say that while individual MPEG coders differ by their predictor parameters,  $N$  and  $q$ , their model normalizes them by their individual standard deviations. Thus, the model is somewhat universal across MPEG codecs. The value of  $c_n$ , the prediction of the cells produced in the  $n^{th}$  frame, is based on  $c_{n-1}$  (the cells produced in the  $n-1^{th}$  frame),  $\mu$  (the mean cell generation rate), and  $\sigma$  (the standard deviation for that codec). It is defined as:

$$c_n = \max(\mu, c_{n-1} + \sigma)$$

This type of prediction is tightly coupled with the characteristics of a particular traffic source type (e.g., MPEG CODECS), whereas the prediction mechanism we use is generic.

### **Dynamic Leaky Bucket**

Tedijanto and Gun propose a Dynamic Leaky Bucket policy mechanism where access-level, connection-specific leaky buckets monitor presented traffic and request more bandwidth of the network when an increase in load is detected [14]. The Dynamic Leaky Bucket needs to be able to rapidly detect increases in presented load. It does this through an *exponential substitution approximation* method. Simply stated, it uses a mathematical estimate called the *equivalent burst length*  $b$  that is calculated based on recent traffic history, and that can be compared with current traffic measurements, to determine if the traffic is changing enough to warrant a change in the bandwidth reservation. If such a change is detected, a request for a bandwidth update is sent to the network. After a delay of a few seconds, which is presumably the network end to end propagation delay, the network has either granted or refused the request for an increase in bandwidth. If the request is denied, the Dynamic Leaky Bucket begins to throttle the arriving traffic to stay within the original request by dropping cells. The Dynamic Leaky Bucket lets the perceived excess pass during the few second delay before the response can return. Tedijanto and Gun present graphs that show how a traditional leaky bucket can continue to saturate a network queue for a 30 second duration whereas the Dynamic Leaky Bucket already reacts in a matter of two to three seconds.

Tedijanto and Gun's work is an interesting mix of prediction and burst-level reservation. Since it does not rely on any feedback from other network components, the Dynamic Leaky Bucket does not have problems scaling to large increases in network dimension. Their work focuses only on the network boundary function and does not investigate how the network switches themselves collaborate in the burst reservation process. More importantly, the time scale of the reaction to the bandwidth changes is vast compared to the few



milliseconds we propose. While the method proposed in [14] can avoid prolonged overload conditions at internal network queues, thousands of QoS violations may be incurred before the burst-reservation feedback is received.

## **2.4 Schedule Generation**

Different practical cell scheduling policies have been proposed that are designed to improve the QoS-Utilization product of the network. These include Static Priority Scheduling (SPS), Partial Buffer Sharing (PBS), Push-Out, Virtual Clock and Magnet II real-time scheduling algorithms (MARS). All of these policies assume that traffic may be categorized according to a small number of QoS classes, and that these classes follow a strict hierarchy with respect to tolerance of packet loss. These algorithms avoid directly enforcing QoS guarantees on a per-connection basis because it seems too complex a function to perform at cell-time granularity. Our thesis asserts that by performing the schedule computation at greater than cell-time granularity, we can undertake more complex schedule generation and thus achieve better QoS performance. We conclude this section with a reference to a study that demonstrates via simulation that an intelligent cell discard policy improves ABR QoS in ATM networks.

Even a simple FIFO queue at the ATM switch output link enforces a cell scheduling policy. This ‘default’ scheduling policy is simply that arriving cells are transmitted in the order that they are received and when an arriving cell finds the buffer full it is discarded, regardless of whether less important packets are queued in front of it.

### **2.4.1 Optimal Discard Policy**

There is an often cited work by Petr and Frost that formalizes the idea of an optimal discard policy [13]. This applies to the problem of schedule generation since, if no reordering of cells is performed by the cell scheduler, then a scheduling policy is fundamentally just a discard policy. Petr and Frosts’ work is primarily interesting for the formalization of the comparison of two discard policies. Their method requires knowledge of traffic statistics and two traffic classes are contemplated. Their work is of theoretical interest; however, effective application of this idea is unlikely because the optimal discard policy must be re-derived for every change in the traffic mix, and the derivation itself is computationally intensive.

### 2.4.2 Delay Priority

In SPS, packets of each class enter a separate FIFO. A lower priority FIFO is only served when all higher priority FIFO's are empty [7]. According to [7] this algorithm is extremely efficient and achieves excellent network utilization levels while providing the packet loss guarantees for the higher priority class ([7] assumed only two classes). One drawback of the algorithm is that it gives unconditional priority to class 1. This affects not only packet loss characteristics but also packet delay characteristics. SPS is a *delay priority* scheduler in that class 2 packets may be dropped due to pending class 1 packets that could actually tolerate further delay without QoS violations. SPS is identical to Head-of-Line (HOL) priority queuing as presented in [9]. The unconditional priority afforded class 1 traffic by HOL priority queuing will service a class 1 arrival without delay, providing that the class 1 FIFO is empty. If class 1 arrivals represent a small percentage of the link's capacity, these cells experience very little jitter. As the percentage of arrivals attributable to class 1 traffic grows, however, class 1 cells suffer increased delay and jitter. This is due to the fact that HOL priority queuing exercises no explicit control over delay.

### 2.4.3 Space Priority

Both PBS [11][3][8] and Push-Out [11] are *space priority* scheduling methods that, unlike the SPS, attempt to decouple lower delay service from lower cell loss. PBS and Push-Out are space-priority schedulers since they allow newly arriving cells to displace cells already in the buffer. This displacement can result in the displaced cell being dropped. In PBS, a class 2 buffer threshold is defined as some value less than the maximum buffer size. If the buffer fills to this threshold level, arriving class 2 cells are discarded. In Push-Out, Kroner et al. select which class 2 cell to discard when there is a full buffer and a class 1 cell arrives [11]. The exact nature of the intelligent selection mechanism is not specified in [11]. Kroner et al. do acknowledge that since cell sequence must be preserved, complicated buffer management logic is necessary, making the push out strategy much more complicated to implement than partial buffer sharing.

Note that despite the elimination of the tight coupling between loss priority and delay priority, neither PBS nor Push-Out explicitly schedules cells in accordance with QoS delay requirements, which is desirable in a general QoS scheduler. As with SPS, both PBS and Push-Out are characterized by the fact that service is still ultimately FIFO.

#### 2.4.4 Virtual Clock

Virtual Clock differs sharply from the preceding approaches in that the cell service discipline is not fundamentally FIFO [15][16]. Virtual Clock generates schedules by calculating the ideal time when each arriving cell should be transmitted and *sequencing* the transmission of cells in accordance with those times. It is important to note that Virtual Clock does not enforce the ideal transmission time, but merely the *order* of cell transmission. Cell loss and delay QoS guarantees are provided on a connection basis. The limitation of Virtual Clock is that scheduling is done assuming that all traffic is of the CBR type. Since ATM is justified to a large extent by the presence of VBR traffic with stringent QoS requirements, Virtual Clock is not sufficiently general for the QoS problem as defined in this thesis.

#### 2.4.5 MARS

MARS, proposed by Hyman, is a more complicated and sophisticated scheduling algorithm than any of the approaches cited above [7]. MARS deals with three classes of traffic: class 1 and class 2 traffic have guaranteed QoS, with class 1 having stricter constraints on loss and delay than class 2. Class 3 reflects non-guaranteed traffic (e.g., data).

MARS delays class 1 and class 2 cells as much as possible if there is waiting class 3 traffic. This is intended to maximize overall throughput and network utilization while still maintaining the QoS commitments for classes 1 and 2. Thus, rather than merely being decoupled, the cell delay QoS and cell loss QoS parameters of a class each play an active role in MARS' schedule computation.

The MARS scheduler makes the determination of which cells to transmit on the basis of cycles. The maximum length of a cycle is the maximum tolerable delay of class 1 cells, denoted as  $H$ . The actual cycle length is often less than  $H$ , and is a function of the number of cells of the three classes that arrive in the previous cycles. The cycle length is always selected with the goal of transmitting the maximum number of class 2 and 3 cells without causing any class 1 cell to miss its deadline or exceeding the loss tolerance of class 2. The calculation of this maximum is based on a prediction of the class 1 and class 2 arrivals over a fixed number of future cycles  $h$ . This prediction is based on class 1 and class 2 arrivals during the previous  $h$  cycles.

While the attempt to manage delay in MARS is intriguing, the delay is managed using local information alone. Despite its clever handling of delay, the MARS system does not

account for the drift that can result from multiple MARS schedulers in tandem. The algorithm fails to provide QoS at a finer granularity than the three classes. The predictions are statistical, unlike ours, so the algorithm can ‘guess wrong.’

#### 2.4.6 Selective Packet Discard

Romanow and Floyd demonstrate that selective packet discard policies that take into account higher layer protocol structures can increase throughput for ABR traffic [17]. This can be achieved without violating protocol layer boundaries if the ATM header includes a “continuation flag” that indicates that a cell is part of a multi-cell higher layer protocol data unit. (The ATM Forum has proposed that a bit be designated for this purpose.)

Romanow and Floyd simulated a single ATM switch connected to a receiver function via a 141 megabit per second link. The switch was fed cells from 10 TCP senders. Two selective discard policies were evaluated:

1. If a cell must be dropped, discard all remaining cells in the same TCP packet.
2. When the switch reaches a pre-defined threshold, actively begin discarding complete cell-sequences pertaining to a TCP packet.

The authors found that while both policies provided better throughput than no selective discard, the second was superior to the first. Their discard policy could easily be integrated into the scheduling heuristics we describe in Chapter 6. Although we are unable to further pursue this line of research here, we include it in Section 8.3 as an interesting area to pursue.

## 2.5 References

- [1] J. Ameyo. *Real-Time Distributed Scheduling and Buffer Management for Congestion Control in Broadband Networks*. Ph.D. thesis, Columbia University, NY 1991.
- [2] J. Ameyo, A. Lazar, and G. Pacifici. Cooperative distributed scheduling for ATS-based broadband networks. In *IEEE INFOCOMM 92*, volume 1, pages 333–342, June 1992.
- [3] A. B. Bondi. An analysis of finite capacity queues with preemptive and non-preemptive priority scheduling and common or reserved waiting areas. *Computers and Operations Research*, 16(3):217–234, 1989.
- [4] M. Gerla and T. Tai. LAN/MAN interconnection to ATM: A simulation study. In *IEEE INFOCOMM 92*, volume 3, pages 2270–2279, June 1992.
- [5] P. Goransson. Bandwidth allocation in hybrid networks. Technical Report TR-93-02, University of New Hampshire, February 1993.

- [6] S. Gupta and M. El-Zarki. Traffic consideration for round-robin scheduling schemes in ATM networks. In *IEEE INFOCOMM 93*, volume 2, pages 820–827, March 1993.
- [7] J. Hyman, A. Lazar, and G. Pacifici. Real-time scheduling with quality of service constraints. *IEEE Journal on Selected Areas in Communications*, 9:1052–1063, September 1991.
- [8] F. Kamoun and L. Kleinrock. Analysis of finite storage in a computer network node environment under general traffic conditions. *IEEE Transactions on Communications*, COM-28(7):992–1003, 1980.
- [9] L. Kleinrock. *Queueing Systems Volume 2: Computer Applications*. Wiley, New York, 1976.
- [10] K. Ko, P. Mishra, and S. Tripathi. Predictive congestion control in high-speed wide-area networks. In *Proceedings of the Second IFIP WG6.2/WG6.4 International Workshop on Protocols for High Speed Networks*, November 1990.
- [11] H. Kroner, G. Hebuterne, P. Boyer, and A. Gravey. Priority management in atm switching nodes. *IEEE Journal on Selected Areas in Communications*, 4:418–427, April 1991.
- [12] P. Pancha and M. El-Zarki. Bandwidth requirements of variable bit rate MPEG sources in ATM networks. In *IEEE INFOCOMM 93*, volume 3, pages 902–909, March 1993.
- [13] D. Petr and V. Frost. Optimal packet discarding: an ATM-oriented analysis model and initial results. In *IEEE INFOCOM 90*, volume 1, pages 537–542, June 1990.
- [14] T. Tedijanto and L. Gun. Effectiveness of dynamic bandwidth management mechanisms in {ATM} networks. In *IEEE INFOCOMM 93*, volume 1, pages 358–367, March 1993.
- [15] L. Zhang. *A New Architecture for Packet Switching Protocols*. Ph.D. thesis, MIT, 1989.
- [16] L. Zhang. Virtualclock: A new traffic control algorithm for packet switching networks. In *ACM SIGCOMM 90*, pages 19–29, September 1990.
- [17] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. In *ACM SIGCOMM 94*, pages 79–88, October 1994.

# Chapter 3

## PREDICTION-BASED CELL SCHEDULING: A MODEL

### 3.1 Introduction

Our ultimate goal in this research is to derive a means of improving the ATM network's ability to provide good QoS performance. It is apparent that the ability to make more complex cell scheduling decisions based on additional information about arriving cells would lead us to this end. We introduce here the *Prediction-based Scheduling* model that provides this capability. This model provides both the means to obtain added information about arriving cells as well as the time to perform a more complex schedule computation.

The key to a model that generates accurate predictions is to base those predictions not on probabilistic estimates of the future but on firm knowledge already obtained. Our intuition is that this firm knowledge does indeed exist at the network periphery in the form of arriving cells. It is easy to understand that the only really unpredictable traffic in the network is that entering the network at the periphery. Note that traffic types that lend themselves to prediction such as CBR and some VBR (e.g., Pancha and El-Zarki [9]) are also accommodated by this model. The challenge, of course, is to predict an inherently unpredictable source.

If we take a snapshot of cells arriving from an unpredictable traffic source and then hold the cells in a delay buffer, we may inject that snapshot into the network as an accurate prediction of what will emerge from that delay buffer. We call that snapshot a *Warning Shot*

and the delay buffer the *Warning Shot buffer (W)*. The function that injects that *Warning Shot* into the network is called the *AALScheduler*. The amount of time cells spend in *W* is exactly equal to the amount of time the *Warning Shot* precedes the actual entry of the cells into the network. This measures how far into the future we are predicting, which we call *PredictHorizon*.

When the predictions arrive at a switch, they may be joined by other, equally accurate predictions emanating from other points on the network periphery. The switch now has at its disposal a complete picture of what cells will arrive *PredictHorizon* cells in the future. Since we wish for each link to be able to be independently scheduled, it is important that the arriving predictions be shunted into outbound link-specific prediction databases (*D*). The shunting process is called *PShunt*. The process of continuing to propagate accurate predictions from the switch is the responsibility of the *LinkScheduler* that controls each outbound link. The *LinkScheduler* utilizes the information in its prediction database *D* to generate a *committed schedule (S)* for the cells that it will transmit *PredictHorizon* in the future. Two points should be noted here. The first is that *LinkScheduler* may emit an accurate prediction to its downstream neighbor since, like *AALScheduler*, it already knows what it will send *PredictHorizon* in the future. The second point is that *LinkScheduler* will be designed so that the schedule generated from *D* will minimize *QoS Cost*. Note that the *LinkScheduler* has an *Enforcer* component assigned the task of actually transmitting cells in accordance with the schedule *S* at the appropriate time. *LinkScheduler* iterates and thus maintains accurate predictions throughout the operation of the network.

### 3.2 Cyclical Operation and Delay Buffers

We described above the process of taking a snapshot of arriving cells and injecting a *Warning Shot* prediction about them into the network *PredictHorizon* cells before their actual entry into the network. The implementation of this concept in an ATM network requires that the predictions themselves travel as cells and that these fixed-size overhead cells be packed efficiently. This leads us to the idea that we will predict several user data cells in a single prediction cell. The number of user data cells predicted in a single prediction cell is denoted as *CellsInPrediction*. This determines the fundamental cell cluster *L* of our network: a single prediction cell followed by *CellsInPrediction* user cells. All communication links of our network are densely packed with these cell clusters. Even when no user data is flowing, the cell clusters flow with prediction cells predicting no arrivals.

The nature of these cell clusters in turn determines our network time cycle. The time interval from the arrival of one cell cluster to the start of the next is the fundamental network time cycle ( $T$ ). Since *LinkScheduler* (and *AALScheduler*) must emit a prediction cell at time cycle granularity, it follows that *LinkScheduler* has one time cycle of computation time available for each schedule computation. One of the major benefits of this model is that scheduling decisions need not be made at cell time granularity like traditional schedulers, but, rather, at cycle granularity. Later in this dissertation we describe simulations where the additional time that this model provides us can be used to make more intelligent scheduling decisions that improve QoS performance.

There are three other classes of delay buffers that are defined for this model. Their roles are generally to phase-synchronize the arrival of prediction cells with the cell clusters they predict. The first of these is the *AAL data only delay buffer* ( $\beta$ ). This is necessary to ensure that even though we cannot emit the prediction of the full cluster until the arrival of the final cell of the cluster, the prediction cell itself remains a full *PredictHorizon* cycles in front of the first cell of the predicted cluster. The second delay buffer is the round-up delay buffer,  $B^R$ .  $B^R$  is used to normalize the ATM network such that all the communications links exhibit a propagation delay that is an integral multiple of the network time cycle. The third is the *Switch data only delay buffer*,  $B^D$ . If predicted cells were not delayed for one cycle in  $B^D$  during the time taken by *LinkScheduler* to compute and predict the next downstream schedule, they would emerge from a switch one cycle time closer to their predictions than when they entered the switch. The insertion of  $B^D$  at each switch maintains the validity of the model's assumption that predictions refer to cells arriving exactly *PredictHorizon* cycles in the future. Simulations conducted in this research illustrate that while the delays produced by these different delay buffers have a negative impact on end-to-end network delay, overall QoS performance can be significantly improved through application of this model.

In the balance of this chapter we formally define the model just introduced in terms of the general graph topology in Figure 4. We first introduce terminology required to state the problem formally.

### 3.3 Terminology

The definitions provided here refer to Figure 4, "Sample Network Topology," on page 32. We assume that the terms switch, node, and link are all familiar to the reader. In Figure 4, nodes are labeled with capital letter A, B, C,...I, and links with numbers 1, 2, 3,...13.



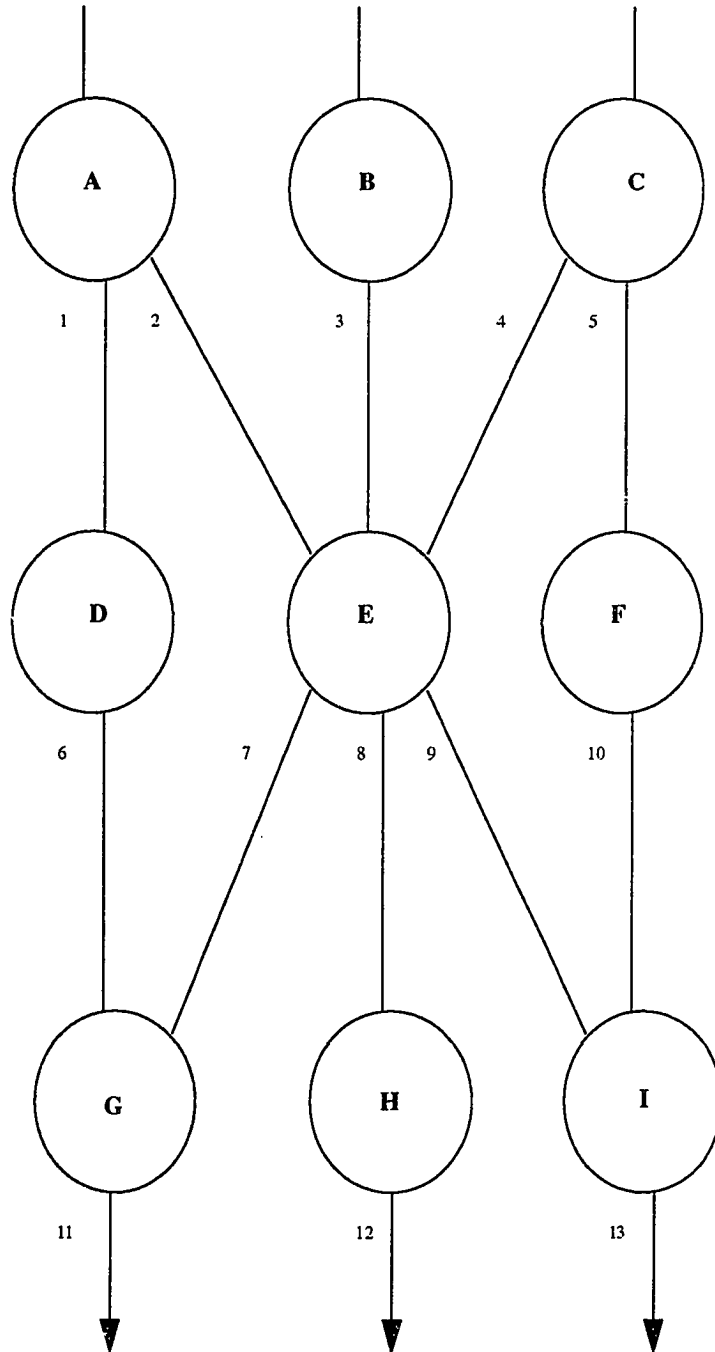
1. *Constant Bit Rate (CBR) traffic*. This is defined in Section 3.5
2. *Variable Bit Rate (VBR) traffic*. This is defined in Section 3.5.
3. *Available Bit Rate (ABR) traffic*. This is defined in Section 3.5.
4. *Unspecified Bit Rate (UBR) traffic*. This is defined in Section 3.5.
5. *UNI*. The User Network Interface.
6. *NNI*. The Network-Network Interface.
7. *ATM Adaptation Layer (AAL)*. This is implemented at the UNI and is analogous to the Network Adaptation Layer (NAL) defined in [1].
8.  $neighbor(X, i)$ . The neighbor of node  $X$  reached over link  $i$ .
9.  $t_i$ . The start of the  $i^{th}$  time interval. These time intervals are aligned at all switches in the network and are all of length  $T$ .
10.  $T_i$ . The time interval  $(t_i, t_{i+1})$  (i.e.  $t_{i+1} = t_i + T$ ).
11.  $O_X^l$ . The outbound link  $l$  at node  $X$ .
12.  $I_X^l$ . The inbound link  $l$  at node  $X$ .
13.  $xy$ . The link connecting  $X$  to node  $Y$ . Cells flow in the direction from  $X$  to  $Y$ . From the perspective of node  $X$ , and assuming  $l = xy$ , this is the same as  $O_X^l$ . From the perspective of node  $Y$ , and assuming  $l = xy$ , this is the same as  $I_Y^l$ .
14.  $\lambda_X^l$ . The set of all inbound links at node  $X$  that may generate cells to be transmitted on link  $l$ . These are the links from the upstream neighbors of node  $X$  from the perspective of  $O_X^l$ .
15.  $\bar{y}x$ . Relative to any node  $X$ , the set of links coming from  $Neighbors(X) - \{Y\}$ .
16.  $P_X^l(j)$ . The predicted schedule of cells to be transmitted on  $O_X^l$  during  $T_j$ .
17.  $S_X^l(j)$ . Committed Schedule, formed by *LinkScheduler* during  $T_{j-1}$ . This is the actual schedule of cells to be transmitted on  $O_X^l$  during  $T_j$ .
18.  ${}_{xy}S_i^l$ . Alternate terminology for  $S_X^l(j)$ , assuming  $l = xy$ .
19.  $S_X^{l \rightarrow a}(j)$ . That component of  $S_X^l(j)$  destined for network link  $a$ . Normally, network link  $a$  is not adjacent to  $X$ , but is exactly one network hop downstream from node  $X$ .

20.  $P_X^{l \rightarrow a}(j)$ . Defined similarly to  $S_X^{l \rightarrow a}(j)$ , but is a prediction for period  $T_j$  rather than a real schedule.
21.  $QoS\Cost(\Omega, T_i)$ . The number of failures to satisfy QoS commitments with regards to loss, jitter or delay occurring in  $\Omega$  during time period  $T_i$ .  $\Omega$  may be either an entire node,  $X$ , or an individual output link on a node,  $O_X^l$ . (With regards to cell loss, it is related to the amount of loss, but is different in a critical way. If loss of a cell does not violate the QoS guarantee of the connection to which that cell belongs, then it does not increase the  $QoS\Cost$  function.)
22. *DistanceFromQoSViolation*. An instantaneous measure of a connection's tolerance of cell loss, delay or jitter.
23. *ScheduleTime*. The running time of *NodeScheduler*. For brevity, this is sometimes denoted as  $s$ .
24. *MaxLinkLength*. The length, in integral units, of the longest internodal link in the network.
25.  $\Delta_i$ . The geographical link propagation delay of link  $i$  expressed in cycles ( $T$ ). This is the exact number of basic time units (not necessarily integral) of physical propagation delay between the nodes at the ends of link  $i$ .
26.  $\delta_i$ . The logical (or normalized) link propagation delay of link  $i$ , expressed as an integral number of cycles ( $T$ ). This is distinguished from  $\Delta_i$  in Section 3.10.1, but may be considered the same as  $\Delta_i$  in the earlier sections.
27. *MaxPropDly*. The  $\delta_i$  of the network link whose length is *MaxLinkLength*.
28. *PredictHorizon*. The number of time units by which predictions precede their corresponding arrivals.
29. *SkeletonDepth*. The number of schedule skeletons, or link prediction data bases, that *LinkScheduler* maintains for each link. This depth determines the maximum queueing delay CBR and VBR cells can experience in a switch.
30. *CellsInPrediction*. The number of cell time slots predicted in one prediction cell which is also one less than the length of the basic cycle time in cell times.
31.  $B^W$ . *Warning Shot* delay buffer, a transmitter network boundary delay introduced by the AAL function to permit generation of accurate schedules. This is explained in Section 3.9.2 on page 46.

32.  ${}_X B_l^R$ . *Round-up* delay buffer added for  $I_X^l$ . This delay is used to normalize the communication lines in the network as explained in Section 3.10.1.
33.  ${}_X B_l^D$ . *Data-only* delay buffer added for  $I_X^l$ . This delay is used to shunt data temporarily to prevent the data cells from overrunning their prediction. This is explained in Section 3.10.1.
34.  ${}_{xy} B_i$ . The *data-only* delay buffer present in AAL X. Assuming  $l = xy$ , this is alternative terminology for  ${}_X B_l^D$ .
35.  ${}_{xy} D_i$ . The Link Prediction Data Base (commonly referred to as *schedule skeletons* in Chapter 6).
36.  ${}_{xy} L_i^n$ . The  $n^{th}$  cluster of cells on link  $xy$  at time  $i$ .
37.  ${}_{xy} L_i^1$ . The first cell in the  $n^{th}$  cluster of cells on link  $xy$  at time  $i$ . This is always a prediction cell in our model.
38. *NodeScheduler*. Given accurate representations of all cells that will arrive at node  $X$  during time period  $T_i$ , this process can produce a set of schedules for the set of outbound servers  $O_\beta, \beta \in Neighbors(X)$ , adjacent to  $X$  such that  $QoSCost(X, T_i)$  is minimized. *NodeScheduler* executes once per time cycle. It consists of executing *PShunt* in parallel for all links in the switch followed by executing *LinkScheduler* in parallel for all links in the switch. During each execution, *NodeScheduler* also executes *Enforcer* in parallel for all links, *CellsIn-Prediction* times, at cell-time granularity.
39. *PShunt*.  $PShunt_{yx}$  copies all the cells that enter switch  $X$  from link  $xy$  at time  $t_i$  into  ${}_{xy} B_i$ . It provides each *LinkScheduler* with a copy of the first cell entering each  ${}_X B_l^D$  in each time cycle.
40. *LinkScheduler* $_{xy}(P, T_i)$ . This process returns a committed schedule  $S$  for  $xy$  to be used for the period  $T_i$  based on the prediction  $P$  provided by *NodeScheduler*.
41. *AALSchedular*. A simple version of *LinkScheduler* that pertains to AALs and not to switches. Like *LinkScheduler*, this emits predictions and produces future transmission schedules that correspond to them. Unlike *LinkScheduler*, it does not have to resolve congestion caused by fan-in from multiple input links. It does not receive predictions at all, but is the initial generator of the predictions at the network edge. The predictions are generated by inspecting new arrivals before shunting them into a  $B^W$  delay buffer.

42. *Enforcer*. A function that transmits cells in accordance with a committed schedule  $S$  produced by *LinkScheduler*. It also drops any new arrivals scheduled for deletion.
43.  $\Leftarrow$ . This relation, when used with schedules, implies that the schedule on the left hand side is derived from the schedule(s) on the right hand side. For example, the expression  $S_E^8(r+1) \Leftarrow S_A^{2 \rightarrow 8}(r) \oplus S_B^{3 \rightarrow 8}(r) \oplus S_C^{4 \rightarrow 8}(r)$  implies that the schedule for link 8 at node  $E$  during  $T_{r+1}$  is derived from the three schedules at  $E$ 's upstream neighbors ( $A$ ,  $B$  and  $C$ ) at  $T_r$ .
44.  $\oplus$ . This operator, from the preceding definition of the relation  $\Leftarrow$ , is used to describe the operation of a *LinkScheduler* on predictions that have arrived from upstream *LinkSchedulers* or *AALSchedulers*.

**FIGURE 4. Sample Network Topology**



### 3.4 Statement of Problem and Goal

This model is constructed to address the following problem:

*Given a set of cells waiting to be transmitted on an output link, select for deletion zero or more cells from those waiting cells, and order the transmission of remaining cells in such a way that this discarding and reordering minimizes the number of QoS violations resulting from this process.*

Using Figure 4 as an example, if the aggregate of the schedules  $S_A^2(k)$ ,  $S_B^3(k)$  and  $S_C^4(k)$  is such that congestion occurs during time period  $T_m$ ,  $m > k$ , on one or more of the downstream links  $O_E^i$ ,  $7 \leq i \leq 9$ , we wish to selectively 'repair' the schedules  $S_A^2(k)$ ,  $S_B^3(k)$  and  $S_C^4(k)$  so as to eliminate that congestion while minimizing  $QoSCost(E, T_m)$ . In this context, 'repair' signifies 'to drop cells from.'

Next we present a novel network model offering accurate short-term prediction of congestion. We later utilize this model as the basis for developing cell scheduling methods that exploit these accurate short term predictions to improve network QoS performance.

### 3.5 Traffic Types Supported

Our model is intended to support all of the traffic types presently considered for ATM. We list these four types below. For a further discussion of the differences between these classes, the reader is referred to [2].

#### 1. CBR: Constant Bit Rate

The bit rate is constant. That is, traffic whose peak bit rate is equal to the average bit rate. It is intolerant of delay variation and tolerant of loss to within a connection-specific limit. The service is normally connection-oriented.

#### 2. VBR: Variable Bit Rate

The bit rate is variable, but a time relation exists between the source and the destination. It is more tolerant of delay variation than CBR, but requires a low upper bound on delay variation, and is tolerant of loss up to connection specific limits. The service is normally connection-oriented.

#### 3. ABR: Available Bit Rate

Highly tolerant of variations in delay and highly intolerant of cell loss, the service is normally connection-oriented. The bit rate is usually variable.

#### 4. UBR: Unspecified Bit Rate

It is highly tolerant of variations in cell delay and loss and the service is normally connectionless. The bit rate is usually variable.

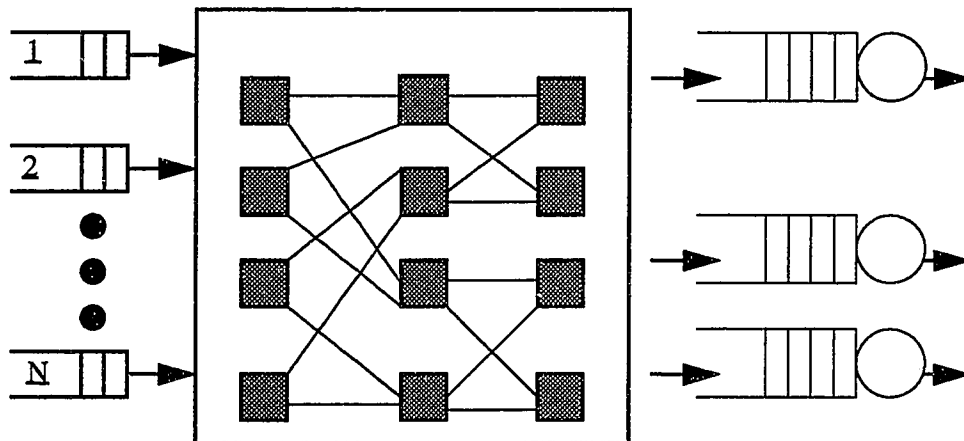
### 3.6 Fundamental Model Assumptions

The assumptions listed in this section are maintained throughout this work.

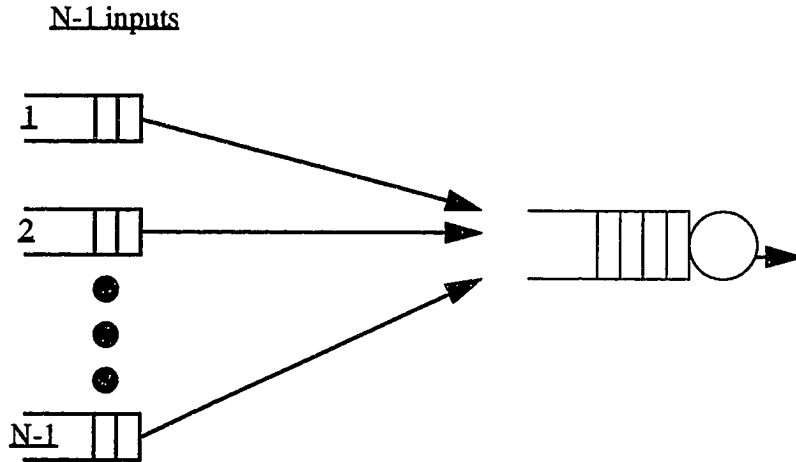
#### 3.6.1 Output Queued Switches

The ATM switches are output queued. A conceptual diagram of an output-queued ATM switch is found in Figure 5. Each of the  $n$  output links of an  $n \times n$  switch thus serves as a multiplexor (see Figure 6) of cells arriving on  $n - 1$  input links destined for that output.

**FIGURE 5. Generic ATM Output Buffered Switch**



**FIGURE 6. Statistical Multiplexing in a Single Switch Output Buffer**



### 3.6.2 Only Arrivals from Neighbors Can Affect Congestion

We assume that the entire traffic load on link  $O_E^l$  during  $T_r$  is derived from the small set of possible inbound links between E's immediate neighbors and itself ( $\lambda_E^l$ ). For example, in Figure 4, when analyzing the short-term load on  $O_E^8$ , we may restrict our analysis to loads at  $\lambda_E^8$  (aka  $\bar{h}e$ ), displaced temporally by the propagation delay over those links. In Figure 4 this consists of the links  $O_A^2$ ,  $O_B^3$  and  $O_C^4$ . Thus, we would restrict our analysis to  $S_A^2(T_{i-\delta_2})$ ,  $S_B^3(T_{i-\delta_3})$  and  $S_C^4(T_{i-\delta_4})$ .

## 3.7 Additional Assumptions

The assumptions listed next are made to simplify the initial presentation of the problem and the solution. Some of these assumptions will be relaxed as the details of the solution are developed. While the remaining non-fundamental assumptions could be relaxed as well, we have not explored these in this research.

In order to present the concept of distributed cell scheduling, we rely heavily on the following assumptions:

### 3.7.1 Regularity

We assume that all internodal links in the ATM network are of equal length and that the propagation delay on those links is an integral multiple of cycle time  $T$ . This implies



$\Delta_l = \delta_l$  for all links  $l$ . Unless otherwise specified, the length of these links will be 1000 km, which is typical of intercity links in the United States.

We additionally assume that all packets are of equal size. This is indeed true for ATM, but this assumption may well be relaxed since, with this one change, the model can be applied to general high-speed packet switching networks. Many researchers and engineers believe that ATM will not completely displace such networks and, in particular, that a gigabit datagram-based Internet may soon exist. The ideas outlined here may have applicability in high-speed routers as the model is not fundamentally tied to the concept of a fixed cell size. We relax this assumption in Section 3.10.

### 3.7.2 Time Cycle Length

The basic network time cycle  $T$  is equal to the propagation delay of the network links (i.e.  $\Delta = \delta = 1$ ). We relax this assumption in Section 3.10.

### 3.7.3 Computation Time for Scheduling

Any computation related to developing cell schedules is assumed to take zero time. We relax this assumption in Section 3.9.

### 3.7.4 Variable Bit Rate Sources

The ideas expressed in this model require that variable bit rate (i.e. VBR, ABR, UBR) sources announce an impending burst slightly before the burst begins to enter the network. If it is not desirable to implement such functionality at the source itself, this functionality may be implemented as a network *traffic boundary* function (see [1]). Thus, there may be two types of variable bit rate traffic sources: the traditional sort where the traffic boundary assumes the task of prediction and the new type where the traffic source uses its own resources to perform the prediction. This is discussed in Section 3.9.2.

We initially assume that the variable bit rate source assumes responsibility for announcing the burst. We relax this assumption in Section 3.9.

### 3.7.5 Isochronous Service

For CBR sources, we assume that the network provides isochronous support, and that the CBR traffic presented at the transmitter network boundary is indeed isochronous. This

assumption trivializes the problem of predicting cell arrivals for CBR sources. If this assumption did not hold for a CBR source we would treat it as a VBR source.

### 3.7.6 Validity of the Prediction

We assume that the predictions are absolutely accurate. It is possible to study the impact of relaxing this assumption so that a certain amount of deviation from the prediction is contemplated. It would be of interest to measure how much error is possible in the prediction without adversely impacting QoS performance.

### 3.7.7 Spatial Extent of the Prediction

Since we assume above that *only arrivals from neighbors can affect congestion*, we may assume that we only need to be able to predict arrivals from our neighbors. We may state a theorem (to be proven in Chapter 4) that says that:

*If accurate predictions about user traffic are available at each network traffic boundary, then accurate predictions may be made throughout the network.*

### 3.7.8 Temporal Extent of the Prediction

We assume a *PredictHorizon* of 1.

If the time span of the prediction is short enough, it is possible to predict with high accuracy arrivals from an upstream switch. By “short enough,” we contemplate a small multiple of the maximum propagation delay over an intercity link. We suggest that the temporal extent be as short as possible based on the intuition that the difficulty and/or cost of accurate prediction increases in direct proportion with the temporal extent of the prediction.

### 3.7.9 Switch Fabric Delay

We assume that there is no delay introduced by the switch fabric of the ATM switch.

We acknowledge that in real switches this delay is non-zero, but, providing that this delay has a tight upper limit, the existence of this delay does not substantially affect our analyses and examples. In this dissertation, for the purpose of simplification, this delay is subsumed by the delay attributed to the delay buffer in our switch architecture. (In fact, such switch fabric delay can be used to reduce the size of the *data-only* delay buffer  $x B_i^D$  discussed later in Section 3.10.1.) It is also conceivable that the switch fabric delay may not be con-

stant. As long as there is an upper bound to this delay, however, this does not pose a fundamental problem for our architecture. When cells are scheduled, the cell scheduler simply uses the worst-case *total switch delay* to derive the earliest possible scheduling time. This *total switch delay* is comprised of any delay suffered by an arriving cell from the time the last bit of the cell has entered the switch until the time that the cell could be transmitted on an output link, assuming no output queue.

### 3.7.10 Switch Buffer Space

We assume that the switch buffer space is allocated on a per-link basis and that no sharing of this buffer space between output links is possible. We assume that these buffers are of equal size for all links in the switch. (These assumptions could be relaxed without impacting our model, but all simulations carried out in this research were performed under this assumption.)

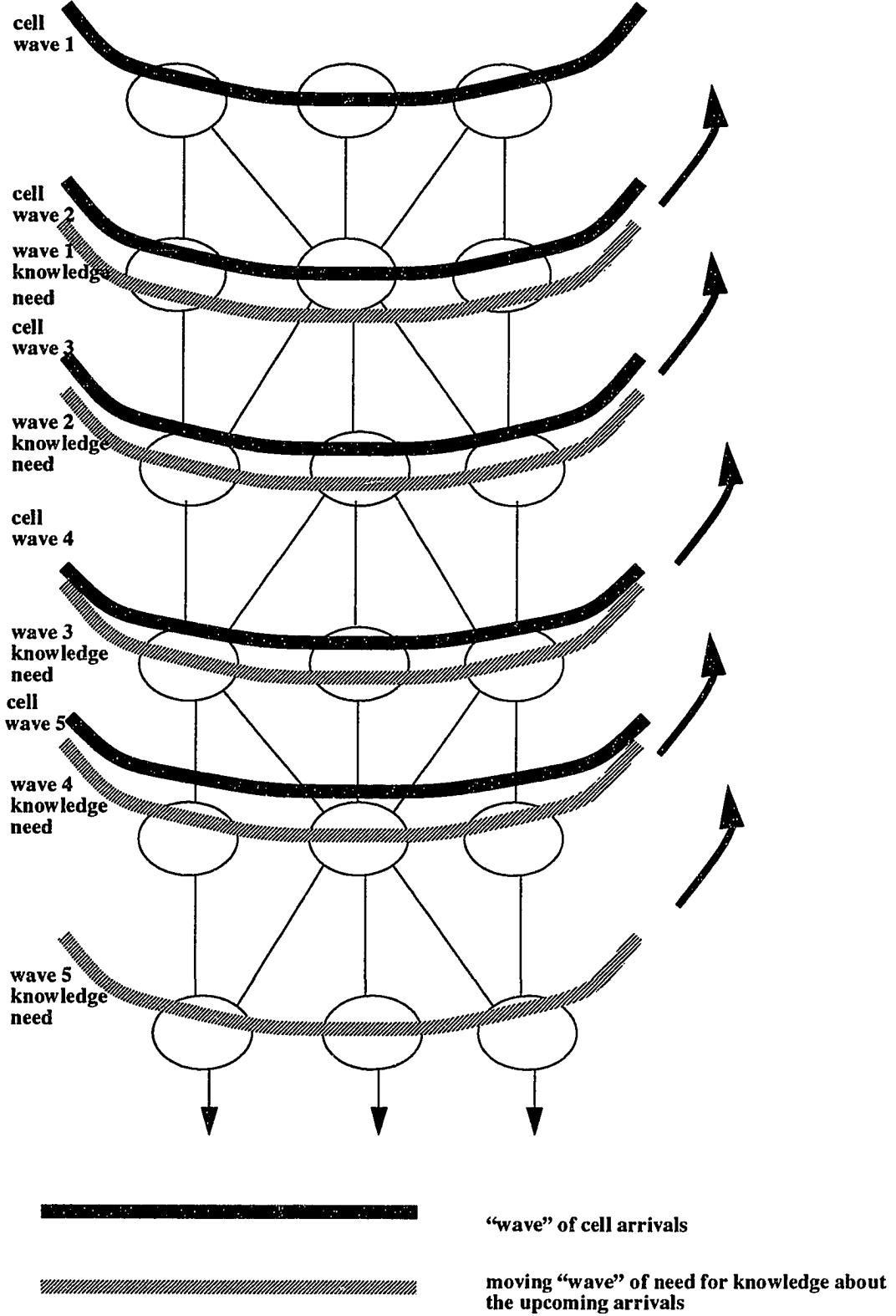
## 3.8 Distributed Cell Scheduling: A Model

In Figure 3 we introduced the idea that overlapping bursts result in downstream queue overflow. We have suggested that the best we can do in ATM for CBR/VBR traffic is to orient the inevitable cell loss in such a way as to minimize *QoS Cost*. In order to make such an optimal decision, it is important to have knowledge of all the cells that arrive at the queue during the congested time period and to have sufficient time to intelligently develop cell schedules based on this information. The knowledge must be acquired sufficiently prior to the cells' arrival at the downstream queue to allow for this computation time. The following sections propose a general model for how this knowledge can propagate *on time* to the locations where it may be used to minimize *QoS Cost*.

The concept that the information about arrivals is needed prior to the actual arrivals is illustrated in Figure 7. This figure depicts that the need for knowledge about arrivals precedes the arrivals themselves. If we assume that we group arrivals into sets of cells that arrive at a node during a particular time period  $T_i$ , we need to know about that set of arrivals during some earlier time period  $T_{i-p}$ . Assuming that the size of the time intervals in the network in Figure 7 is equal to the link propagation delay, the 'need' for the knowledge about the arrivals for  $T_i$  is shown to always arrive during time period  $T_{i-1}$ . This *knowledge-need* is satisfied in the model by predictions that precede the actual arrivals by a fixed temporal displacement of  $p$  ( $p \geq 1$ ) time periods. Since the displacement is fixed, the set of predictions that arrive during period  $T_{i-p}$  corresponds exactly to the set of cells

that arrive in time period  $T_i$ . How this prior knowledge of arrivals can be applied to achieve *distributed cell scheduling* is explained in the following section.

**FIGURE 7. Waves of Knowledge-need Preceding Waves of Arrivals**



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

### 3.8.1 Distributed Schedules

This thesis is motivated by the idea that any output queueing (and hence congestion and loss) that can occur at  $O_X^l$  during  $T_i$  can be anticipated by looking upstream at the set of cells destined to arrive at  $O_X^l$  during  $T_i$ . These cells are displaced both *spatially* and *temporally* from  $O_X^l$ , but the magnitude of these displacements may be small enough to suggest use of this information. Indeed, it is easy to imagine that any queue that may exist at  $O_X^l$  exists in a *distributed* fashion at the set of upstream neighbors of  $X$  (spatial displacement) slightly before they arrive at  $X$  (temporal displacement).

The difficulty in communicating global information about network queues in high-speed networks is that the queues are so transient that the queues may change radically many times during the time it takes for one cell to propagate over a single internodal link. For this reason, rather than treat distributed queues, we use distributed arrival schedules. This is reasonable since a network queue can always be determined when one knows the traffic arrivals to that queue, the initial state of the queue, and the service rate of the queue (which, in bits per unit time, is constant in ATM).

As we argued above for queues, the arrival schedule for  $O_X^l$  during  $T_i$  may be found by looking upstream at the distributed components of that schedule. The schedule components are displaced spatially and temporally as explained above. We state (and later justify) that the distributed schedule components for  $O_X^l$  during  $T_i$  are present at the upstream neighbors of  $X$  during time period  $T_{i-\delta}$ .

Using as an example link 8 in Figure 4, this may be expressed formally as:

$$S_E^8(r + \delta) \Leftarrow S_A^{2 \rightarrow 8}(r) \oplus S_B^{3 \rightarrow 8}(r) \oplus S_C^{4 \rightarrow 8}(r)$$

Recall that we remain under the assumption that  $\delta_l = 1$  for all links  $l$ . Thus, for the sake of clarity, references to the '+ $\delta$ ' in expressions like the preceding will be shown as '+1' in the balance of Section 3.8. We then rewrite the preceding expression as follows:

$$S_E^8(r + 1) \Leftarrow S_A^{2 \rightarrow 8}(r) \oplus S_B^{3 \rightarrow 8}(r) \oplus S_C^{4 \rightarrow 8}(r)$$

### 3.8.2 Predicting Cell Arrivals

Our work assumes that accurate short-term predictions of forthcoming cell arrivals are feasible. By *short-term* we mean that the arrivals during time period  $T_i$  must be accurately

predicted during time period  $T_{i-p}$ , where  $p$  ( $p \geq 1$ ) is the (small) constant *PredictHorizon*. We have already defined the prediction  $P_X^l(j)$ . Since the prediction is accurate,

$$S_X^l(j) = P_X^l(j)$$

The prediction  $P_X^l(j)$  is actually composed of smaller predictions specific to the individual output links of *neighbor*( $X, i$ ). For example, returning to Figure 4:

$$P_E^8(r+1) \Leftarrow P_A^{2 \rightarrow 8}(r) \oplus P_B^{3 \rightarrow 8}(r) \oplus P_C^{4 \rightarrow 8}(r)$$

Substituting this into the earlier relation,

$$S_E^8(r+1) \Leftarrow P_A^{2 \rightarrow 8}(r) \oplus P_B^{3 \rightarrow 8}(r) \oplus P_C^{4 \rightarrow 8}(r)$$

Note that the '+ 1' in this example is an artifact of the assumption that the propagation delay of the internodal links is exactly one basic time cycle. *It is not related to any assumption about the size of PredictHorizon.*

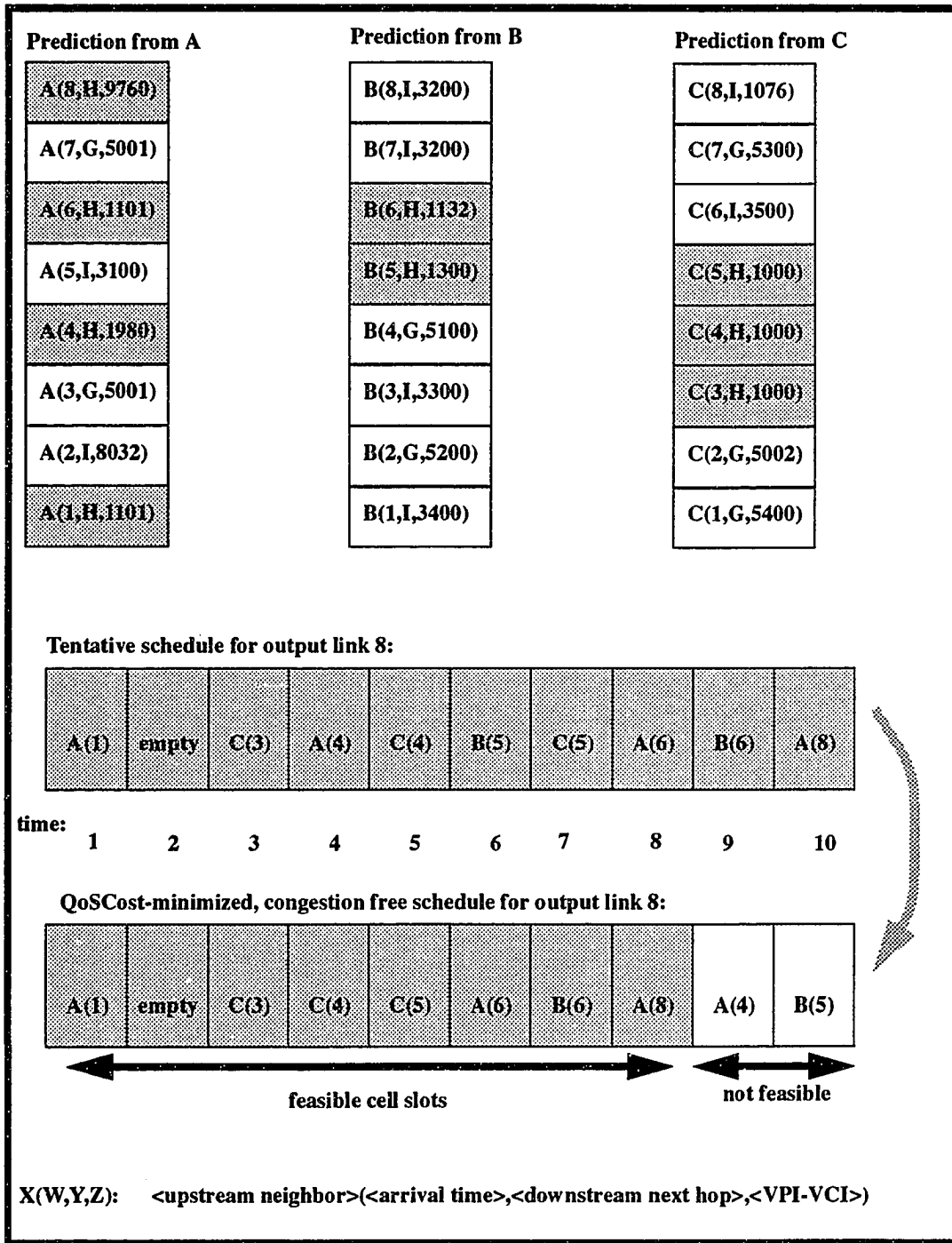
### 3.8.3 Schedule Generation

One of the fundamental premises of this model is the fact that the predicted terms in the above relationship are known during time period  $T_{r-p}$ . We can utilize this fact to send the predictions downstream  $p$  full time periods before the actual arrivals begin. From the perspective of dealing with congestion at a single output link  $O_E^8$ , all the information that it needs to detect local congestion during  $T_r$  is present during  $T_{r-p}$ . If congestion will occur, a scheduler at  $O_X^l$  can *proactively* determine how many cells must be pruned from the schedule to eliminate the congestion, and which cells should be pruned to minimize the occurrence of QoS violations.

#### 3.8.3.1 Prediction Analysis Phase

At time  $t_{r-1}$ , the predictions  $P_A^2(r)$ ,  $P_B^3(r)$  and  $P_C^4(r)$  are at  $E$  and are decomposed into their components specific to  $O_E^7$ ,  $O_E^8$  and  $O_E^9$ . This decomposition is depicted in Figure 8. In the case of  $O_E^8$ , the relevant components are  $P_A^{2 \rightarrow 8}(r)$ ,  $P_B^{3 \rightarrow 8}(r)$  and  $P_C^{4 \rightarrow 8}(r)$ . These three components represent a *tentative*  $S_E^8(r+1)$ .

FIGURE 8. Prediction Analysis Phase





### 3.8.3.2 Minimization of QoS Cost and Downstream Prediction Generation

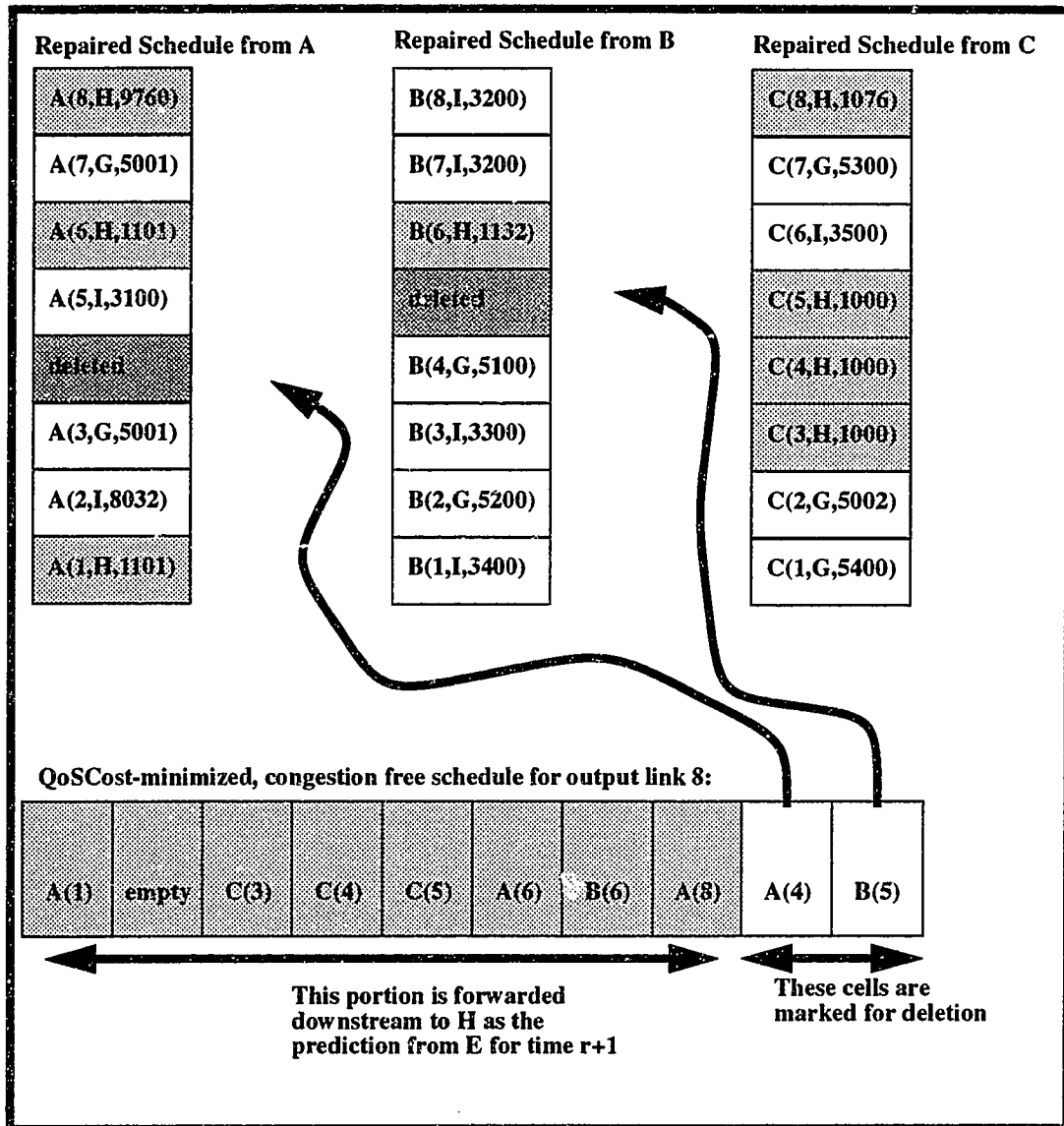
The tentative  $S_E^8(r+1)$  is analyzed to determine if it can be served by  $O_E^8$  during  $T_{r+1}$  without any cell loss at all. If this is not possible, sufficient cells will be dropped from this schedule so that all cells *remaining* in the schedule will be served without loss. The decision about which cells to drop is made so that the minimum QoS cost is incurred. This is performed by invoking a *LinkScheduler* function providing the tentative schedule as input. The *LinkScheduler* function produces a *committed* schedule  $S_E^8(r+1)$  and a corresponding downstream prediction  $P_E^8(r+1)$ . Since we assume that the computation time for the *LinkScheduler* is zero, *LinkScheduler* will have finished generating  $P_E^8(r+1)$  and can forward  $P_E^8(r+1)$  from node  $E$  at time  $t_r$ .

### 3.8.3.3 Prediction Pruning

The predicted schedule  $P_E^8(r+1)$  is then decomposed into upstream neighbor-specific components. Each of  $O_E^7$ ,  $O_E^8$  and  $O_E^9$  contribute components for each of the upstream neighbors,  $A$ ,  $B$  and  $C$ . In the case of  $A$ , the individual components amount to *pruned* versions of the individual components  $P_A^{2 \rightarrow 7}(r)$ ,  $P_A^{2 \rightarrow 8}(r)$  and  $P_A^{2 \rightarrow 9}(r)$  derived from  $P_A^2(r)$  during the prediction analysis phase. The thus-pruned  $P_A^{2 \rightarrow 7}(r)$ ,  $P_A^{2 \rightarrow 8}(r)$  and  $P_A^{2 \rightarrow 9}(r)$  are then reassembled into a committed, congestion-preventing arrival schedule for  $I_E^2$  over the period  $T_r$ . This reassembly is shown in Figure 9. The cells marked *deleted* in Figure 9 will be dropped before they reach the  $O_E^l$ ,  $7 \leq l \leq 9$ , for which they were originally intended. Note that this reassembly may be performed *in parallel* with the processing of the arrival schedules for  $I_E^2$  and  $I_E^2$  as there are no memory conflicts and the decision processes are independent.

In this manner, the cycle of prediction/schedule building occurs continually throughout the network with a fundamental cycle time of  $T$ .

FIGURE 9. Arrival Schedule Repair Phase



### 3.9 Realizing an Accurate Cell Arrival Predictor: *The Warning Shot*

The mechanism for performing distributed cell scheduling presented in Section 3.8 relies on making accurate predictions about cell arrivals at least one time period in advance of the actual arrivals. While cell arrival prediction can be a very difficult problem, in this section we illustrate how relatively simple operations at the network edge can make accurate predictions throughout the network a reality.

### 3.9.1 CBR Sources

The definition of a CBR source implies that it is inherently predictable. Since the source emits cells at a regular rate, *it is trivial to predict the arrivals from a CBR source at the transmitting network boundary*. We maintain the assumption made in Section 3.7.5 that a CBR source is an isochronous source at the transmitting network boundary, and that this isochronous characteristic is maintained throughout the network.

### 3.9.2 VBR Sources

VBR sources are characterized by the fact that their cell rate may vary greatly from time period to time period, and that in general there is no way to predict with 100% accuracy in  $T_i$  what the cell arrivals will be in  $T_{i+1}$ . The *Warning Shot* prediction is a means of escape from this dilemma.

We know *precisely* what cells the VBR source transmits during  $T_i$  at  $t_{i+1}$  (since they have already been transmitted). If, at the traffic boundary, we actually direct the cell stream into a delay buffer  $B^W$ , such that  $|B^W| = 2T$ , we can ‘predict’ (i.e., *report faithfully*) the arrivals during  $T_{i+2}$  by immediately forwarding a *warning shot* at  $t_{i+1}$ . The warning shot fired at  $t_{i+1}$  propagates a *wave of knowledge* (see Figure 7) about  $T_{i+2}$  that precedes the actual ‘data wave’ by one time period. If instead we chose to direct the cell stream into a delay buffer with a delay of  $3T$ , we can ‘predict’ the arrivals during  $T_{i+3}$  by immediately forwarding a *warning shot* at  $t_{i+1}$ . In this second example, the warning shot fired at  $t_{i+1}$  propagates a *wave of knowledge* (see Figure 10) about  $T_{i+3}$  that precedes the actual ‘data wave’ by two time periods. In general, the temporal extent of the prediction can be increased by increasing the size of the delay buffer. In theory, we could ‘predict’ one hour in advance by sending the cell stream through a one hour delay buffer. Of course, this is ridiculous both because the delay buffer would be prohibitively large and because there are few user traffic types that can tolerate more than a few seconds delay through the network, to say nothing of one hour.

This principle may be put into practice in a manner that is both feasible from a hardware cost perspective and that increments the end-to-end network delay by a tolerable percentage. As an example, assume that our basic time unit is 1 ms and we wish to predict 12 time units into the future. This implies that we need a delay buffer of 13 ms. This increases the end-to-end delay for any VBR stream by 13 ms. Even an example involving only three intercity hops will incur more propagation delay than 13 ms (the delay is 15 ms for three 1000 kilometer hops). To this figure one must add switching, queueing, and segmentation

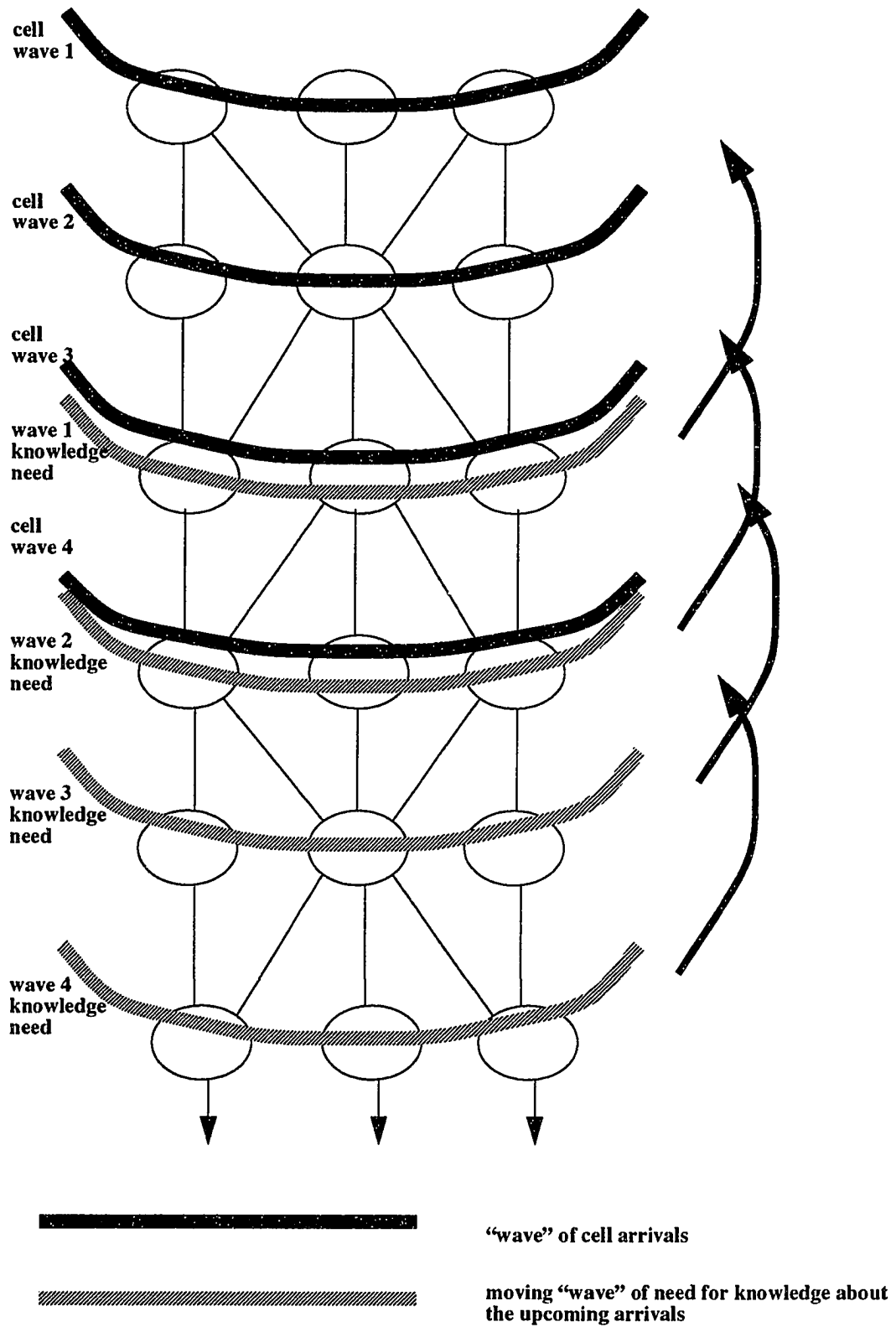
and reassembly delays. We thus argue that an artificial increase of 13 ms would be acceptable, if there is a great improvement in utilization and QoS to be derived from the increase.

There is also a hardware cost for maintaining a delay buffer of 13 ms. At 150 MBps, approximately four hundred 53-byte ATM cells can arrive in one millisecond. Thus, a delay buffer of 13 ms. requires a 275,600 byte ( $400 \times 13 \times 53$ ) shift register. This already reasonable figure is further mitigated by the fact that 1) such large delay lines would only be required at the UNI interface (i.e. at the network traffic boundary), and 2) that a UNI interface, unlike NNI interfaces, is highly unlikely to carry by itself a traffic load equal to the network trunk speeds.

This approach supports generic VBR sources that have no prediction capability. The network architecture could support enhanced VBR sources that have the capability to generate their own predictions. This could be advantageous for certain types of traffic where the user indeed does have some foreknowledge of the future time periods, allowing the user to predict, for example, 13 ms in advance without explicitly delaying the data flow by 13 ms.

Finally, for the VBR source that is absolutely intolerant of even a small increase in delay, and that has no inherent prediction capabilities, the option of *peak bandwidth reservation* can still be provided by the network. This effectively turns the VBR source into a CBR source from the scheduling perspective. That is, it would require that null cells be generated whenever the VBR source is emitting below its peak rate. Use of this type of source would be discouraged, as it has a direct negative impact on network utilization.

**FIGURE 10. A *PredictHorizon* of Two Cycles**



### 3.9.3 Determination of the Extent of the Prediction (*PredictHorizon*)

Since an increase in the temporal extent of the prediction (*PredictHorizon*) results in an increase in the end-to-end network delay, we should strive to make this as short as possible. How near into the future may we predict? The mechanism described in Section 3.10.3 requires that the predictions be able to travel from an upstream node to its downstream neighbor, and be processed *by the time the predicted cells arrive*. Thus,

$$PredictHorizon > ScheduleTime$$

That is, the predictions only need to precede their corresponding arrivals by the time it takes to perform the required computations on those predictions. Since, under our current assumptions, *ScheduleTime* is zero, *PredictHorizon* may be as small as one time cycle. The relationship between *PredictHorizon* and *ScheduleTime* is important. In particular, when we allow non-zero *ScheduleTime* (Section 3.10), we must remember that longer scheduling time directly increases the size of *PredictHorizon* and hence increases network latency.

### 3.9.4 Maintaining the Accuracy of the Prediction

As explained above, an accurate prediction, whether from the user or from the traffic boundary function itself, is available at each traffic boundary in the network. If all predictions at all network traffic boundaries are accurate, then all predictions at every switch in the network are equally accurate by enforcement of the distributed schedules described earlier. This is shown in Chapter 4 by a proof by induction.

## 3.10 Relaxation of Some Assumptions

At this point we can relax the assumptions related to network regularity, time cycle length, and computation time for scheduling. We may relax the network regularity assumption (Section 3.7.1) by *normalizing* a standard ATM network. We define a normalized ATM network below, and explain how a 'raw' ATM network may be normalized without changing the physical topology in any way. The network time cycle length  $T$  is defined in accordance with the definition of the normalized network.

### 3.10.1 Normalized ATM Networks

In a normalized ATM network, every communications link has a propagation delay that is an integral multiple of the basic time unit  $T$  for that network. Since the actual physical dis-

tances separating two nodes cannot be guaranteed to display such regularity, a delay buffer  $B^R$  is added at each input link  $I_X^l$  to ‘round up’ the delay to the next higher acceptable threshold.

Even with the alignment afforded by this delay buffer, when the predictions arrive for  $t_i$ , they are delayed by the processing of *NodeScheduler* and are not forwarded downstream until the start of  $t_{i+1}$ . In order to prevent the wave of arrivals from gaining ground on the wave of predictions preceding it, it is necessary to insert an additional *data-only* delay of  $s$  time units at each input link  $I_X^l$ . (This data-only delay is created by shunting data cells into the  $B_l^D$  buffer upon arrival at the switch.) In Section 3.10.2 below we explain that the minimum allowable length of  $s$  is fixed by *ScheduleTime*. By data-only, we imply that the cell flow is inspected as it flows past the start of this final delay buffer. If the cell is marked as a prediction cell, it is immediately detoured from the cell stream by *PShunt* which distributes relevant predictions to *LinkSchedulers*. If it is any other cell, it does not logically ‘arrive’ at the node for one additional time period. This kind of real time in-line inspection of a delay buffer is a well-known hardware technique that is used for token detection and ring cleaning in FDDI [3] [4] networks.

We do, of course, incur a penalty for normalizing the network in this fashion. If we assume that the basic time unit of a network is 1 ms, insertion of the  $B^R$  and  $B^D$  delay buffers increases the average link’s propagation delay by 1500  $\mu s$ , and, in the worst case, by 1999  $\mu s$ . A virtual path including 5 hops has its end-to-end network delay artificially increased by an average of 6 ms, and in the worst case 10 ms. In general,

$$DelayPenalty \leq |B^W| + MaxHops \times (|B^D| + \max(|B^R|))$$

### 3.10.2 Determination of the Network Basic Time Unit

It thus is desirable for us to make the basic time unit of the network ( $T$ ) as short as possible in order to minimize this *DelayPenalty*. Each  $O_X^l$  in the network must emit one prediction per time unit. Thus,

$$|T| \geq ScheduleTime$$

*Aside: We mentioned in Section 3.9.3 that the PredictHorizon also has to be longer than ScheduleTime. If we reduce ScheduleTime in order to reduce DelayPenalty, we risk leaving too little computation time to perform the intelligent scheduling we need for improved*

*QoS performance. A fundamental trade-off is that increasing the amount of computation time directly increases network latency.*

### 3.10.3 Realistic Distributed Scheduling: An Example

The example presented in this section is based on the normalized topology depicted in Figure 11. Note that the geographical topology from which this normalized topology was derived is one where the following relationships about physical propagation delays  $\delta_2, \delta_3,$  and  $\delta_4$  hold:

1.  $3 < \delta_2 \leq 4$
2.  $0 < \delta_3 \leq 1$
3.  $1 < \delta_4 \leq 2$

We also assume that  $ScheduleTime < 1$  (e.g. 0.999 time units). We can thus assign  $PredictHorizon = 1$ .

We originally motivated the distributed schedule concept in terms of Figure 4, with the following relationship:

$$S_E^8(r+1) \Leftarrow S_A^{2 \rightarrow 8}(r) \oplus S_B^{3 \rightarrow 8}(r) \oplus S_C^{4 \rightarrow 8}(r)$$

Based on the explanation of the normalized network in Section 3.10.1 and the example in Figure 11, this relationship now becomes:

$$S_E^8(r+5) \Leftarrow S_A^{2 \rightarrow 8}(r) \oplus S_B^{3 \rightarrow 8}(r+3) \oplus S_C^{4 \rightarrow 8}(r+2)$$

This is due to the fact that the waves of arrivals that contribute to the load at  $O_E^8$  during any time period  $T_i$  departed their upstream output links at different times. Specifically, the cells that load  $O_E^8$  during time period  $T_{r+5}$  depart node  $A$  during  $T_r$ , node  $B$  during  $T_{r+3}$ , and node  $C$  during  $T_{r+2}$ . We illustrate the overall process in the following timeline, which relates to Figure 11 and Figure 12:

1.  $[t_{r-1}] P_A^2(r)$  departs  $O_A^2$ .
2.  $[t_r] P_A^2(r+1)$  departs  $O_A^2$ ,  $P_B^3(r+1)$  departs  $O_B^3$ ,  $P_C^4(r+1)$  departs  $O_C^4$ , and  $P_E^8(r+1)$  departs  $O_E^8$ . The first cell predicted by  $P_A^2(r)$  departs  $O_A^2$ .



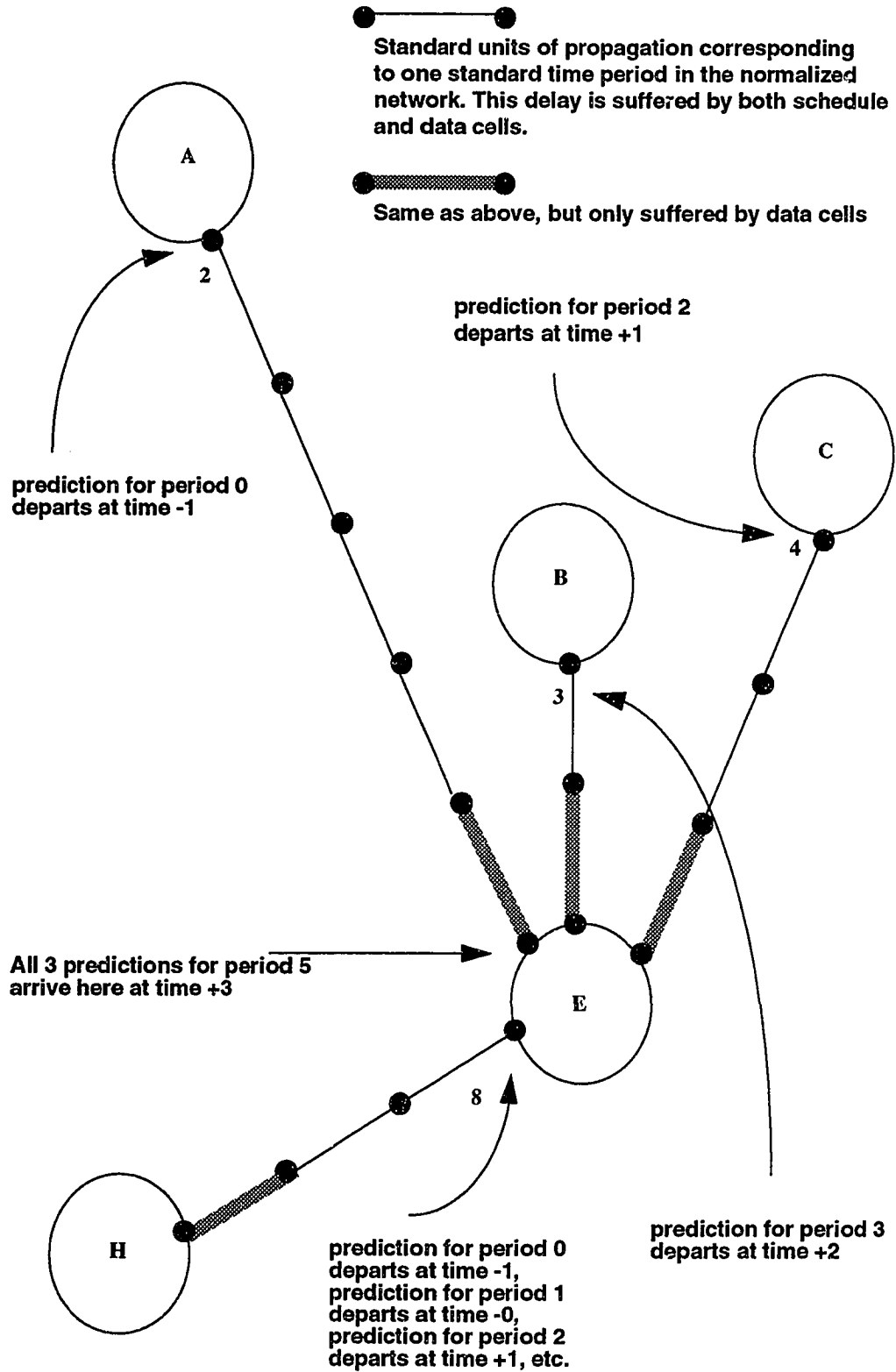
Note: None of the predictions departing at  $t_r$  are depicted in Figure 11 and Figure 12 since none of them are relevant to the particular cycle described in this example. We list them here to underscore the fact that there are many active overlapping *prediction/congestion detection/schedule generation* cycles occurring between each pair of neighbors. For clarity, this time-line does not further document these contemporaneous but irrelevant events.

3.  $[t_{r+1}] P_C^4(r+2)$  departs  $O_C^4$ .
4.  $[t_{r+2}] P_B^3(r+3)$  departs  $O_B^3$ . The first cell predicted by  $P_C^4(r+2)$  departs  $O_C^4$ .
5.  $[t_{r+3}] P_A^2(r)$ ,  $P_B^3(r+3)$  and  $P_C^4(r+2)$  simultaneously 'detour' their final delay buffers and enter *NodeScheduler*. The first cell predicted by  $P_B^3(r+3)$  departs  $O_B^3$ .

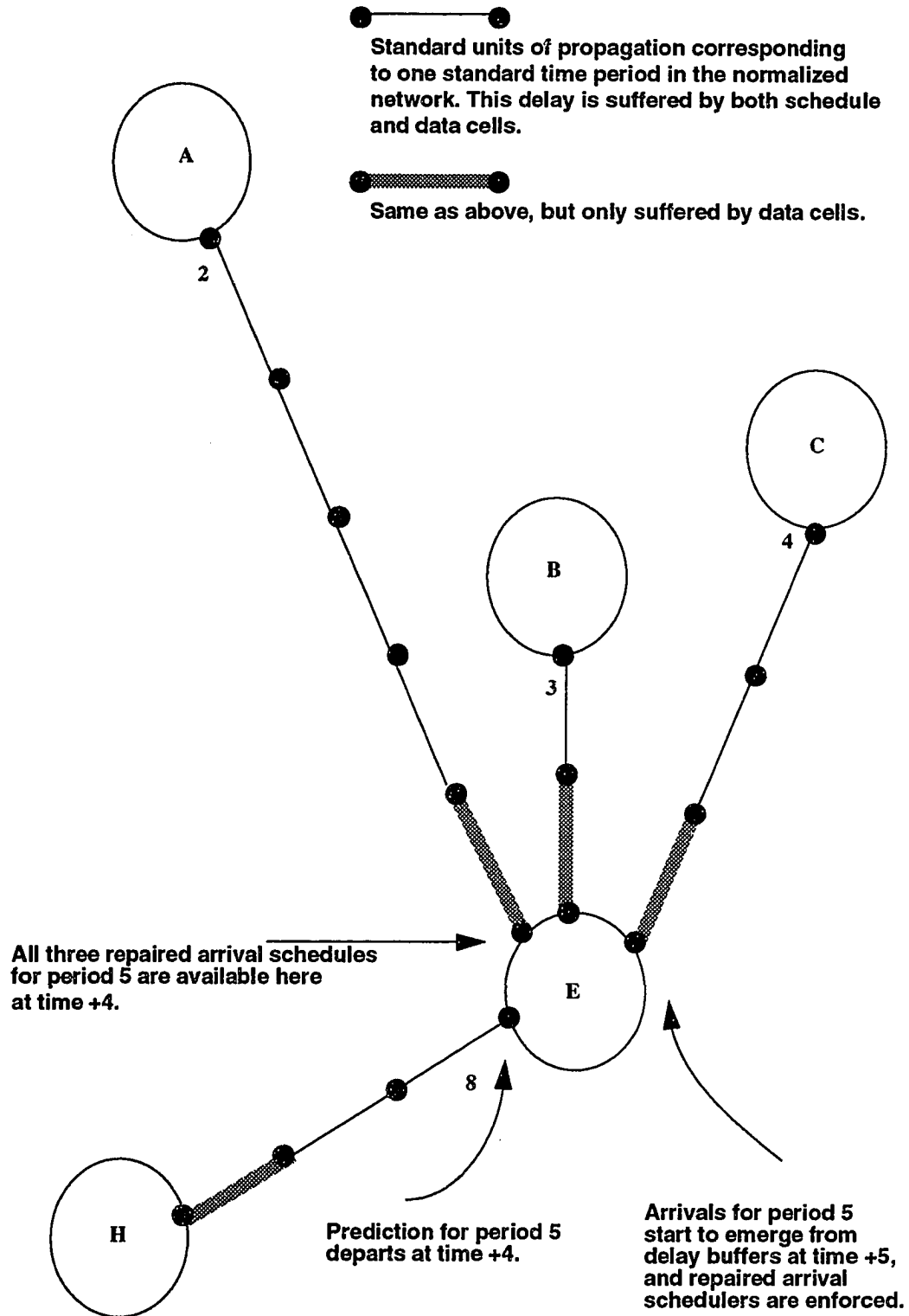
Note: Despite the differing *upstream* predicted periods ( $r, r+3, r+2$ ), these three predictions all refer to cells that arrive at node  $E$  during the same time period  $t_{r+4}$ . This is due to the fact that  $\delta_2 = \delta_3 + 3 = \delta_4 + 2$ .

6.  $[t_{r+4}] P_E^8(r+5)$  departs  $O_E^8$ , and the committed arrival schedules for  $I_E^2, I_E^3$ , and  $I_E^4$  are available to *Enforcer* (so that any cells marked for deletion can be discarded as they arrive). The first cells predicted by  $P_A^2(r)$ ,  $P_C^4(r+2)$ , and  $P_B^3(r+3)$  enter  $E B_2^D$ ,  $E B_3^D$ , and  $E B_4^D$ , respectively.
7.  $[t_{r+5}]$  The cells corresponding to  $P_A^2(r)$ ,  $P_C^4(r+2)$ , and  $P_B^3(r+3)$  begin to emerge from their respective data-only delay buffers and arrive at  $O_E^8$ . The committed arrival schedules that became available at  $t_{r+4}$  are enforced (i.e. cell discard is performed). The cells not thus pruned (by *Enforcer*) are delivered to  $O_E^8$ . Because of this intelligent cell pruning, no additional cells from this group of arrivals are lost at  $O_E^8$ .

**FIGURE 11. Synchronizing Distributed Schedules: time -1 to + 2**



**FIGURE 12. Synchronizing Distributed Schedules: time +4 to +5**



### 3.11 Extensions to the Model

The model has thus far focussed on scheduling a small number of cell arrivals competing for access to the same outbound link. We have ignored the possibility that previously predicted cells may also still be candidates competing for that same bandwidth. In order to avoid excessive cell loss, some queueing of cells is desirable. To support this, we extend the concept of prediction pruning so that an unschedulable cell need not be dropped, but may wait in a prediction database  $D$ . When *LinkScheduler* is invoked, it may schedule predictions already waiting in the prediction database  $D$  in addition to those cells predicted in the current cycle.

We have specified that *LinkScheduler* makes scheduling decisions such that QoS Cost is minimized. Making such trade-offs about relative QoS Cost between connections entails maintaining some QoS history for each connection. Quality of Service is perceived at the receiver traffic boundary and is therefore ultimately an 'end-to-end' network concept. Thus, it is desirable that the QoS history used for the scheduling decisions capture the *global* QoS state of the connection.

Specific details of how these extensions can be implemented are provided in Chapter 6.

### 3.12 References

- [1] P. Goransson. Bandwidth allocation in hybrid networks. Technical Report TR-93-02, University of New Hampshire, February 1993.
- [2] R. Vetter. ATM concepts, architectures, and protocols. *Communications of the ACM*, Vol. 38 No. 2:31–38, February 1995.
- [3] ANSI X3.139-1987. *Fiber Distributed Data Interface (FDDI) - Token Ring Media Access Control*.
- [4] ANSI X3.148-1987. *Fiber Distributed Data Interface (FDDI) - Token Ring Physical Layer Protocol*.

# Chapter 4

## ACCURACY OF PREDICTIONS: FORMAL PROOF

### 4.1 Introduction

This chapter contains two theorems proving that, under the assumptions stated in Section 4.4, all predictions generated by our network model are accurate. The theorems are stated formally below.

#### Theorem 1

*If accurate predictions about user traffic are available at each network traffic boundary, and the network PredictHorizon is equal to one, then accurate predictions may be made throughout the network.*

#### Theorem 2

*If accurate predictions about user traffic are available at each network traffic boundary, then accurate predictions may be made throughout the network. This holds true for all  $1 \leq P \leq M$ , where  $P$  is PredictHorizon and  $M$  is MaxLinkLength.*

Both theorems are proven by induction. Before proceeding with the proofs of the main results of the chapter, we first provide an intuitive overview of the proofs and then follow with some definitions necessary for the proofs.

## 4.2 Overview of the Proofs

We assume that the network begins operation empty of all traffic. The Warning Shot buffers and *LinkScheduler* databases throughout are empty and, thus, by definition, are accurate for an empty network. At this time, since all Warning Shot buffers and *LinkScheduler* databases are empty, the initial predictions generated by all *AALSchedulers* and *LinkSchedulers* throughout the network are null. This situation continues until the first user traffic is presented to a Warning Shot buffer at an AAL. When this first user traffic arrives at the AAL, an accurate prediction is produced by *AALScheduler*, in accordance with its definition. This is repeated cyclically as user traffic enters the network throughout the operation of the network. Hence, all predictions injected at the network edge are accurate. We need to show that all predictions injected within the network (i.e. by *LinkSchedulers*) are accurate.

We illustrate this by defining a network state *PredictValid* where all predictions are accurate. Assuming that all prediction databases  $D$  and propagating predictions are *PredictValid* at  $t_i$ , the defined operation of *LinkScheduler* at  $t_{i+1}$  will inject new predictions into the network at  $t_{i+1}$  and leave all the  $D$  databases *PredictValid* for the operation of the *LinkSchedulers* at  $t_{i+2}$ . Those predictions injected at  $t_{i+1}$  are *PredictValid* by the definition of the *LinkSchedulers* and by the assumption that they operate on input that is *PredictValid* at  $t_i$ . We prove, via induction on the network time cycle  $T_i$ , that the network is permanently in the *PredictValid* state. This proof is presented formally in the first half of this chapter, assuming a *PredictHorizon* of one, and then is extended in the second half of the chapter to allow for a generalized *PredictHorizon*.

## 4.3 Network PredictState

### Definition 4.1

A network's *PredictState* is either *PredictValid* or *PredictInvalid*. The state is *PredictValid* if all the network *PredictComponents* (Section 4.3.3) are individually *PredictValid*.

### 4.3.1 Time-cycle Granularity

A Network's *PredictState* is defined at network time cycle granularity. This is an abstraction of the physical network since, in this way, we consider that all events that can happen in the interval  $\{t_i, t_{i+1}\}$  occur instantaneously at  $t_i$ . For example, all of the  $P$  cells that

are transmitted during  $\{t_i, t_{i+1}\}$  are instantaneously transmitted and undergo all the propagation that they experience in the actual interval  $\{t_i, t_{i+1}\}$  at the instant  $t_i$ . No *PredictState* related activities then occur until  $t_{i+1}$ . This last requirement is necessary or the cycle-granularity abstraction loses synchronization with the network it represents. The abstraction transmits  $P$  cells per cycle time rather than  $P$  cells per  $P$  cell times. Exactly the same number of cells is transmitted per cycle in both cases.

We observe that this kind of abstraction is not at all extraordinary when modelling networks. Indeed, while it is quite familiar to deal with the transmission of cells at cell-time granularity, that is an abstraction of the more realistic view of a cell's bits transmitted piecemeal at bit-time granularity. Even this is an abstraction of the fact that a bit on a communications line may involve multiple state changes on the physical medium on which the bit is represented!

### 4.3.2 Notation

1.  $\delta_{xy}$ . The logical (or normalized) link propagation delay of link  $xy$ , in integral basic time units. This is the same as the  $\delta_i$  defined in Section 3.3.
2.  $P$ . The number of cells in a basic network time cycle. Equivalent to one more than *CellsInPrediction* defined in Section 3.3.
3.  $\equiv$ . *Cell-equivalence*. For two schedules  $A$  and  $B$ ,  $A \equiv B$  means that for each cell  $A_k$  in position  $k$ ,  $1 \leq k \leq P$ , the identical cell exists in  $B$  ( $A_k = B_k$ ). When no cell is present at position  $A_k$ , then no cell is present in  $B_k$ . We denote *not* cell-equivalent by  $\neg(A \equiv B)$ .
4.  $\approx$ . *Schedule-equivalence*. For two schedules  $A$  and  $B$ ,  $A \approx B$  means that the cell (connection) indicated in position  $A_k$ ,  $2 \leq k \leq P$ , is identical to the cell (connection) indicated in  $B_k$ . This relationship is distinct from cell-equivalence primarily in that we may use schedule-equivalence to compare 1) prediction cells, 2) schedules, 3) cell clusters on communications links and 4) the contents of delay buffers. Each of these four classes of objects denotes an ordered set of VCI indicators. Schedule-equivalence compares these ordered sets rather than the items themselves. We denote *not* schedule-equivalent by  $\neg(A \approx B)$ . (Note that  $A \equiv B \Rightarrow A \approx B$ ).

5.  $\leftarrow$ . The schedule assignment operator. After execution of  $LHS \leftarrow RHS$ , then  $LHS \equiv RHS$ .
6.  ${}^C_A X_B^D$ . A general format of notation to uniquely identify an individual network component of type  $X$ . The  $A$  subscript denotes the link to which the component pertains, and is usually designated in the form  $pq$ , where  $p$  is the transmitting node of the link and  $q$  is the receiving node of the link. The  $B$  subscript denotes the network time cycle to which we refer. The  $C$  superscript, when present, refers to the time cycle-offset on a link, counting from one, and starting at the transmitting node  $p$  of the link  $pq$ . The  $D$  superscript, when present, refers to a cell slot position within the set of cells defined by the rest of the expression.
7.  $\bar{y}x$ . Relative to any node  $X$ , the set of links coming from  $Neighbors(X) - \{Y\}$ . For example,  $\bar{z}y B_i^{yz}$  signifies all the data-only delay buffers ( $B$ ) in switch  $Y$  at time  $i$  except that buffer receiving traffic from node  $Z$ .
8.  $K_i$ . A scratchpad used by *LinkScheduler* during schedule computation. It denotes the cells present in the link prediction database at the time schedule computation begins at time  $i-1$  that should be dropped from that database before schedule computation begins at time  $i$ .
9.  $s_i$ . A scratchpad used by *LinkScheduler* during schedule computation. Its use is explained in Definition 4.3.
10.  $\sigma_i$ . A scratchpad used by *AALScheduler* during schedule computation. Its use is explained in Definition 4.4.

### 4.3.3 PredictComponents

In this section we define each *PredictComponent* and discuss how to determine if it is *PredictValid* from its contents and the contents of other *PredictComponents*. There are two classes of *PredictComponents*, passive and active.

Figure 13 depicts a small network showing its various passive *PredictComponents* at time  $t_i$ . These *PredictComponents* are more clearly labeled in the magnified network view in Figure 14. They include the delay buffers  ${}_{xy}W_i$ ,  ${}_{xy}\beta_i$ , and  ${}_{xy}B_i$ ; Link Prediction Data Bases



${}_{xy}D_i$ ; Committed Schedules  ${}_{xy}S_i$ ; propagating cell clusters  ${}_{xy}L_i^n$ ; and the prediction cell carried in each of those clusters,  ${}_{xy}L_i^1$ . All of these terms were originally introduced in Section 3.3.

As indicated above, the network PredictState is uniquely determined by the PredictState of each of these passive *PredictComponents*. From one time cycle to the next, the PredictState of these components is affected only by the effect of the active *PredictComponents* on the existing passive components. The active *PredictComponents* include cell propagation, the prediction distributor *PShunt*, the cell scheduling algorithms *LinkScheduler* and *AALSchedular*, and, in the case of  ${}_{xy}W_i$ , the arrival of new cells at the network edge.

We do not need to prove that the active *PredictComponents* operate correctly. Those active *PredictComponents* that are algorithms are assumed to operate in accordance with their definitions. Since cell propagation and cell arrivals are merely descriptions of physical occurrences, they are correct by definition.

FIGURE 13. *PredictState* Network Components

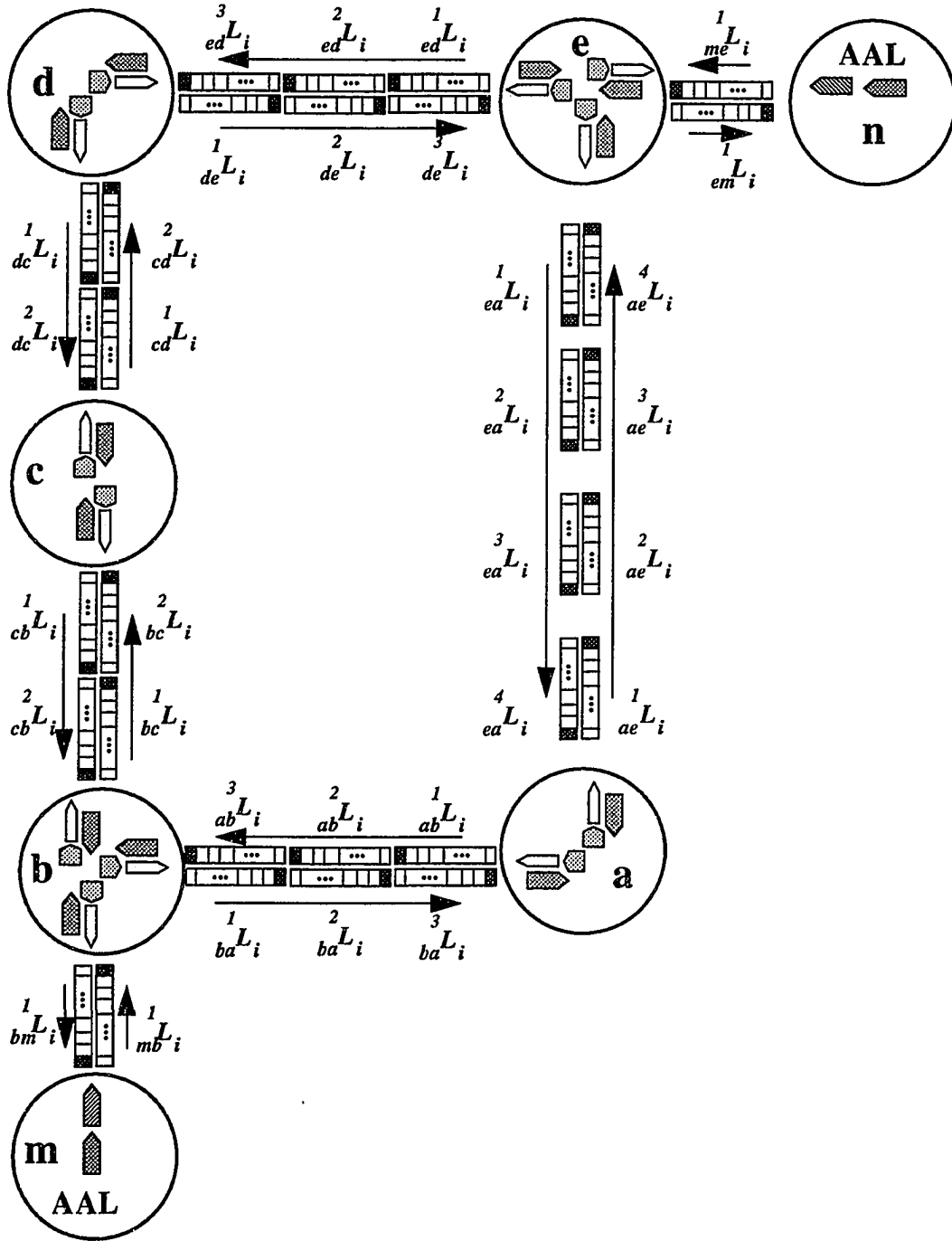
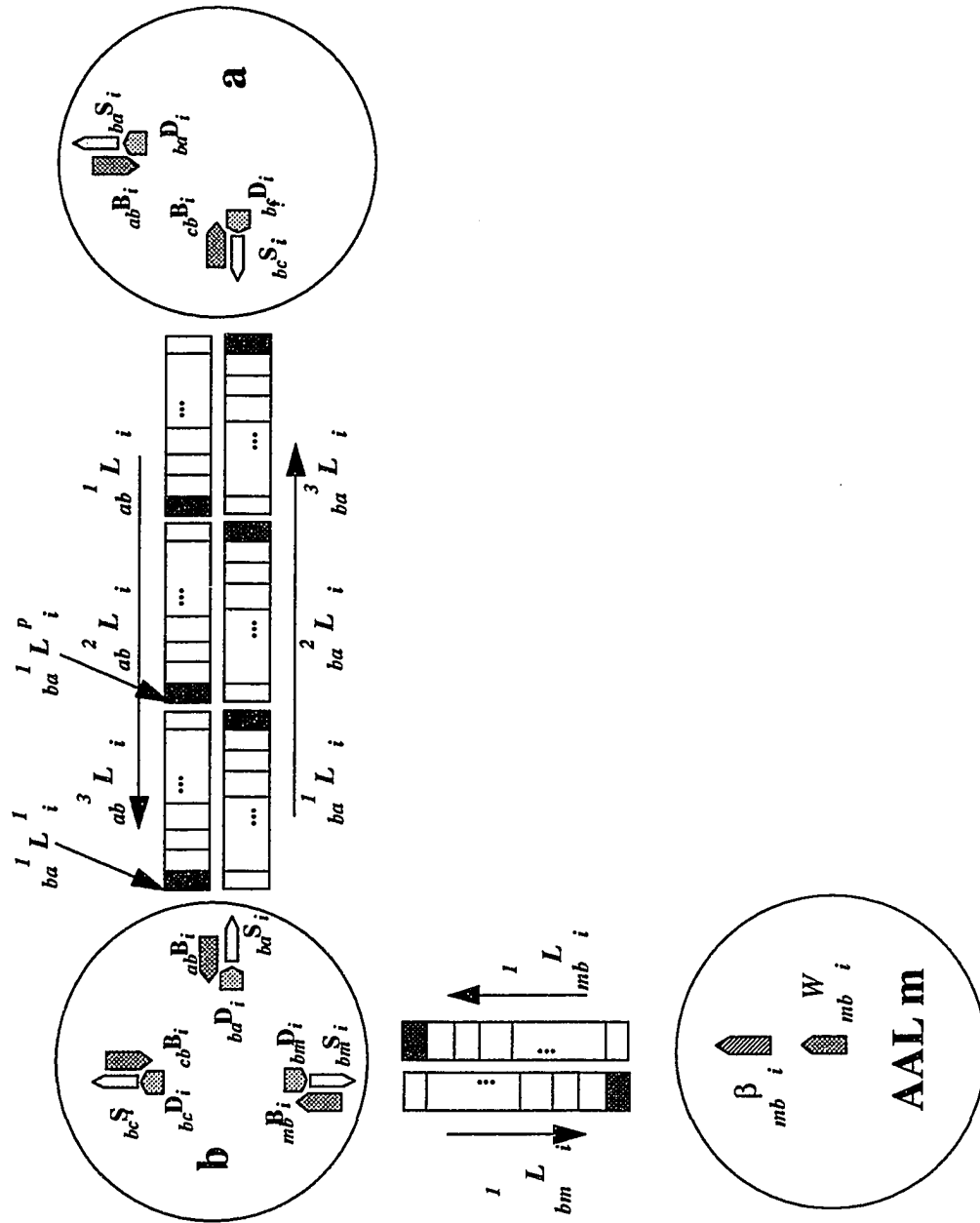


FIGURE 14. PredictState Network Components (detailed)



## PShunt

### Definition 4.2

$PShunt_{yx}$  copies all the cells that enter switch  $X$  from link  $xy$  at time  $t_i$  into  ${}_{xy}B_i$ . Thus, the following relationship is preserved by  $PShunt_{yx}$ :

$${}_{xy}B_i = \delta_{xy} L_{i-1}$$

Additionally, at time  $t_i$ ,  $PShunt_{xy}$  shunts pertinent predictions from  ${}_{xy}B_i^1$  into the appropriate  ${}_{yx}D_{i+1}$  link prediction databases. If the cell is not a prediction cell, then a null prediction cell is passed to *NodeScheduler*. Note that it is always a prediction cell according to our model except during network ramp-up following  $t_0$ . We express this formally as follows:

$${}_{yx}A_{i+1} = PShunt_{xy}({}_{yx}B_i^1)$$

We introduce the term  $A$  for notational convenience only. It is not a separate *PredictComponent*. It simply allows us to reference subsets of the  ${}_{pq}B_i^1$  prediction cells that are relevant to a specific output link. The  ${}_{pq}B_i^1$  cells are part of our network *PredictComponents*.

## LinkScheduler

LinkScheduler is run independently for each output link once per time cycle. The description of LinkScheduler here is limited to how it receives and forwards predictions and cells such that the network state remains PredictValid. The actual decision regarding which cells to schedule is deferred to an arbitrary function  $f$ . While this research emphasizes study of the scheduler policies presented in Section 6.4, the function  $f$  is free to make any scheduling decisions as long as it only schedules cells that reside in  $D$  at the time of the schedule computation. When it decides that a cell should be dropped, it reports this information. The function  $f$  is defined as follows:

$$\{S_{i+1}, K_{i+1}\} \leftarrow f(D_i)$$

$K$  is not a network PredictComponent.  $K$  represents those cells that  $f$  has designated should be dropped. It is merely a scratchpad used for communication between  $f$  and *LinkScheduler*.

### Definition 4.3

For each output link  $xy$  *LinkScheduler* performs the following steps at  $t_i$  using the scratchpad  $s$ :

1.  ${}_{xy}S_{i+1} \leftarrow {}_{xy}S_i$
2.  $\{{}_{xy}S_{i+1}, {}_{xy}K_{i+1}\} \leftarrow f({}_{xy}D_i)$
3. populates  ${}_{xy}S_{i+1}^1$  such that  ${}_{xy}S_{i+1}^1 \approx {}_{xy}S_{i+1}$
4.  $D_{i+1}$  is derived from  $D_i$  by adding any pertinent new predictions ( $-A_i$ ) received from *PShunt* and by deleting any predictions that have just been scheduled ( $S_{i+1}$ ) as well as any explicit deletion requests ( $K_i$ ) from  $f$ . That is,

$${}_{xy}D_{i+1} \leftarrow {}_{xy}D_i + \{-A_{i+1}\} - \{{}_{xy}S_{i+1} \cup {}_{xy}K_{i+1}\}$$

5.  ${}_{xy}L_{i+1}^1 \leftarrow {}_{xy}S_{i+1}$

The effect of the  $A$ ,  $S$  and  $K$  terms above is that they ensure that only valid predictions are added to the data base (i.e.  $A$ ) and that any predictions no longer eligible for scheduling (i.e.,  $S$ ,  $K$ ) are removed. Although we show step 5 as the responsibility of *LinkScheduler* for the purposes of this proof, step 5 is actually performed at cell-time granularity by *Enforcer*. Since this proof is conducted at cycle-time granularity, it is convenient to subsume the function of *Enforcer* into *LinkScheduler*.

### **AALSchedular**

### Definition 4.4

*AALSchedular* performs the following steps at  $t_i$  using the scratchpad  $\sigma$ :

1.  ${}_{xy}\sigma_{i+1} \leftarrow {}_{xy}\beta_i$
2.  ${}_{xy}\beta_{i+1} \leftarrow {}_{xy}W_i$
3. populates  ${}_{xy}\sigma_{i+1}^1$  such that  ${}_{xy}\sigma_{i+1}^1 \approx {}_{xy}\beta_{i+1}$
4.  ${}_{xy}L_{i+1}^1 \leftarrow {}_{xy}\sigma_{i+1}$

5.  ${}_{xy}W_i \leftarrow \{c_1, c_2, \dots, c_p\}$ , where  $c_k$  is the user cell that arrives at AAL  $X$  during the  $k^{th}$  cell time slot of  $\{t_{i-1}, t_i\}$  and  $c_k$  is null if no cell arrived during that slot.

### Propagation Delay

#### Definition 4.5

We define the propagation delay function as follows: given the current location of a cell and a number  $t$  of time cycles, propagation delay determines the location of that cell after that number of time cycles. We define this formally for three cases:

1. case  $1 < k+t < \delta+1$ :

$${}_{xy}L_{i+t}^{k+t} \equiv {}_{xy}L_i^k$$

2. case  $k+t = \delta+1$

$${}_{xy}B_{i+t} \equiv {}_{xy}L_i^k$$

3. case  $k+t > \delta+1$  or  $k+t < 1$

The propagation delay function is not defined.

### External Input

The only external input that can influence the network PredictState is the arrival of cells from a traffic source into a  $W$ -class buffer. That is, we assume that no cells are injected into the network except by passing through a Warning Shot buffer monitored by an *AALScheduler*. Other than these cells, all other influence on network PredictState is the result of the action of a network PredictComponent.

### Passive Components and PredictValidity

In this section we define *PredictValidity* for each of the passive network PredictComponents. Each of these were introduced in Section 3.3. Section explains how the contents of  ${}_{xy}W_i$  and  ${}_{xy}\beta_i$  are derived.

#### Definition 4.6

A Warning Shot buffer  ${}_{xy}W_i$  is *PredictValid* if  ${}_{xy}W_i \equiv \{c_1, c_2, \dots, c_p\}$ , where  $c_k$  is the user cell that arrives at AAL  $X$  during the  $k^{th}$  cell time slot of  $\{t_{i-1}, t_i\}$  and

$c_k$  is null if no cell arrived during that slot. (Note that  ${}_{xy}W_i$  is always *PredictValid* by Definition 4.4.)

**Definition 4.7**

A delay buffer  ${}_{xy}\beta_i$  is *PredictValid* if  ${}_{xy}\beta_i \approx {}_{xy}W_{i-1}$ .

That is, a delay buffer  $\beta$  should contain those cells that resided in the corresponding warning shot buffer  $W$  in the preceding time cycle.

**Definition 4.8**

A link prediction database  ${}_{xy}D_{i+1}$  is *PredictValid* if

$${}_{xy}D_{i+1} \equiv {}_{xy}D_i + \{-A_{yx,i+1}\} - \{{}_{xy}S_{i+1} + {}_{xy}K_{i+1}\} .$$

That is, a link's prediction database  $D$  should consist of those predictions that were present in the database in the preceding time cycle plus any predictions that arrived from *PShunt* in the preceding time cycle less any cells scheduled or deleted in the current time cycle.

**Definition 4.9**

A committed schedule  ${}_{xy}S_i$  is *PredictValid* if  ${}_{xy}D_{i-1}$  is *PredictValid*. A null  ${}_{xy}S_i$  is always *PredictValid*.

Stated simply, if *LinkScheduler* operates on a *PredictValid* database, then its output schedule  $S$  is *PredictValid*.

**Definition 4.10**

The prediction cell carried in the first cell position of every cell group  $L$  is *PredictValid* if it accurately predicts the cells in the cell group immediately following it. We state this formally in three separate cases as follows:

**case 1** ( $n = 1$ , X is a switch):  ${}^1L_i$  is *PredictValid* if  ${}^1L_i \approx {}_{xy}S_i$ .

**case 2** ( $n = 1$ , X is a AAL):  ${}^1L_i$  is *PredictValid* if  ${}^1L_i \approx {}_{xy}\beta_i$ .

case 3 ( $1 < n \leq \delta_{xy}$ ):  ${}_{xy}L_i^n$  is *PredictValid* if  ${}_{xy}L_i^n \approx {}_{xy}L_i^{n-1}$ .

**Definition 4.11**

A cell group  $L$  is *PredictValid* if an accurate prediction of the cells in  $L$  is contained in the prediction cell in the first cell position in the preceding cell group. We state this formally in two separate cases as follows:

case 1 ( $1 \leq n < \delta_{xy}$ ):  ${}_{xy}L_i^n$  is *PredictValid* if  ${}_{xy}L_i^n \approx {}_{xy}L_i^{n+1}$ .

case 2 ( $n = \delta_{xy}$ ):  ${}_{xy}L_i^\delta$  is *PredictValid* if  ${}_{xy}L_i^\delta \approx {}_{xy}B_i^1$ .

In light of Definition 4.10, this implies that *PredictValidity* of cell group  $L$  is logically equivalent to *PredictValidity* of the prediction cell in the preceding cell group.

**Definition 4.12**

A delay buffer  ${}_{xy}B_i$  is *PredictValid* if  ${}_{xy}B_i \approx {}_{xy}B_{i-1}^1$ .

That is, the delay buffer  $B$  should contain those cells predicted by the prediction cell residing in  $B$  in the preceding time cycle.

**4.4 Assumptions**

**Assumption 4.1**

All network hardware is synchronized to a global clock and is synchronously initialized as follows at  $t_0$ :

- i) All communication links are assumed to carry null cells in all cell slots.
- ii) All delay buffers  $\beta_0$  and  $B_0$  throughout the network are initialized to null cells.
- iii) All *Warning Shot* buffers (type  $W_0$ ) are initialized to null cells.
- iv) All Link Prediction Databases  $D_0$  are initialized to empty.
- v) All Committed Schedules  $S_0$  are initialized to empty.



#### Assumption 4.2

In all cases, lack of a prediction cell in a slot time reserved for a prediction cell is processed equivalently to an empty prediction cell. *This assumption is necessary to conveniently handle the fact that network links of different lengths require different amounts of ramp-up time.*

#### Assumption 4.3

As stated above, the only external event that can influence the network PredictState is the arrival of cells from a user source into a  $W$  buffer. This specifically excludes the possibility of hardware failures and bit errors on communication links from this proof.

#### Assumption 4.4

Reference to the contents of a predict component at a time earlier than  $t_0$  automatically assumes that it was empty (or full of null cells, as appropriate) at that time. Thus,  ${}_{xy}W_{-1}$  or  ${}_{xy}B_{-1}$  are assumed to be empty. This assumption is necessary for the start-up conditions of the proof and is consistent with our assumptions about how the network buffers will be initialized. That is, there can be no history of non-null cells prior to  $t_0$  anywhere in the network as no cells arrived prior to  $t_0$ .

#### Assumption 4.5

*PredictHorizon* is assumed to be equal to 1.

### **4.5 The Restricted Prediction Theorem**

In this section we prove Theorem 1, stated earlier in Section 4.1.

#### **Base Case of the Induction**

We need to prove that the composite network PredictState is *PredictValid* at  $t_0$  by demonstrating that the individual passive PredictComponents are all *PredictValid*:

1.  ${}_{xy}W_0$ .  
 ${}_{xy}W_0$  is empty at  $t_0$  (Assumption 4.1(iii)).

Since no user cell can have arrived in  $\{-1, 0\}$ , this is *PredictValid* (Definition 4.6).

2.  ${}_{xy}\beta_0$ .

${}_{xy}\beta_0$  is empty at  $t_0$  (Assumption 4.1(ii)).  ${}_{xy}W_{-1}$  is empty (Assumption 4.4). Thus, since  ${}_{xy}\beta_0 \equiv {}_{xy}W_{-1}$ ,  ${}_{xy}\beta_0$  is *PredictValid* (Definition 4.7).

3.  ${}_{xy}D_0$ .

${}_{xy}D_0$  and  ${}_{xy}K_0$  are empty at  $t_0$  (Assumption 4.1(iv)).  ${}_{xy}S_0$  is empty at  $t_0$  (Assumption 4.1(v)).  ${}_{yx}A_0$  is empty at  $t_0$  (Assumption 4.1(ii), Definition 4.2). Since  ${}_{xy}D_{-1}$  is empty (Assumption 4.4), then  ${}_{xy}D_0 = {}_{xy}D_{-1} + \{{}_{yx}A_0\} - \{{}_{xy}S_0 + {}_{xy}K_0\} = \phi$ . Thus,  ${}_{xy}D_0$  is *PredictValid* (Definition 4.8).

4.  ${}_{xy}S_0$ .

${}_{xy}S_0$  is empty at  $t_0$  (Assumption 4.1(v)). Thus,  ${}_{xy}S_0$  is *PredictValid* (Definition 4.9).

5.  ${}_{xy}L_0^1$ .

**case 1** ( $n = 1$ , X is a switch): Since  ${}_{xy}L_0^1 \approx {}_{xy}S_0 \approx \phi$  (Assumption 4.1(i,v)),  ${}_{xy}L_0^1$  is *PredictValid* (Definition 4.10).

**case 2** ( $n = 1$ , X is an AAL): Since  ${}_{xy}L_0^1 \approx {}_{xy}\beta_0 \approx \phi$  (Assumption 4.1(i,ii)),  ${}_{xy}L_0^1$  is *PredictValid* (Definition 4.10).

**case 3** ( $1 < n \leq \delta_{xy}$ ): Since  ${}_{xy}L_0^1 \approx {}_{xy}L_0^{n-1} \approx \phi$  (Assumption 4.1(i)),  ${}_{xy}L_0^1$  is *PredictValid* (Definition 4.10).

6.  ${}_{xy}L_0^n$ .

${}_{xy}L_0^n$  is empty at  $t_0$ , by the initialization assumption.

**case 1** ( $1 \leq n < \delta_{xy}$ ): Since  ${}_{xy}L_0^n \approx {}_{xy}L_0^{n+1} \approx \phi$  (Assumption 4.1(i)),  ${}_{xy}L_0^n$  is *PredictValid* (Definition 4.11).

**case 2** ( $n = \delta_{xy}$ ): Since  ${}_{xy}L_0 \approx {}_{xy}B_0^1 \approx \phi$  (Assumption 4.1(i,ii)),  ${}_{xy}L_0$  is *PredictValid* (Definition 4.11).

7.  ${}_{xy}B_0$ .

Since  ${}_{xy}B_0 \approx {}_{xy}B_{-1}^1 \approx \phi$  (Assumption 4.1(ii) and Assumption 4.4),  ${}_{xy}B_0$  is *PredictValid* (Definition 4.12).

This concludes the base case.

### Inductive Hypothesis

The network *PredictState* is *PredictValid* at  $t_k$ ,  $0 \leq k \leq i$ .

At time  $t_{i+1}$  the three active *PredictComponents*, *PShunt*, *LinkScheduler* and *AALSched-  
uler* operate and perform the steps in Definition 4.2, Definition 4.3, and Definition 4.4, respectively. We need to show that the composite network *PredictState* is *PredictValid* at  $t_{i+1}$  by demonstrating that the individual passive *PredictComponents* are all *PredictValid* at  $t_{i+1}$ .

### Inductive Step

1.  ${}_{xy}W_{i+1}$ .

By the action of the *AALSched-  
uler* (Definition 4.4, step 5),  ${}_{xy}W_{i+1}$  is *PredictValid* by Definition 4.6.

2.  ${}_{xy}\beta_{i+1}$ .

By Definition 4.4 (step 2),  ${}_{xy}\beta_{i+1} \leftarrow {}_{xy}W_i \cdot {}_{xy}\beta_{i+1}$  is therefore *PredictValid* (Definition 4.7).

3.  ${}_{xy}S_{i+1}$ .

${}_{xy}D_i$  is *PredictValid* by the inductive hypothesis. Therefore, by Definition 4.9,

${}_{xy}S_{i+1}$  is *PredictValid*.

4.  ${}_{xy}D_{i+1}$ .

$A_{i+1}$  and  $K_{i+1}$  are automatically *PredictValid* as part of our assumption that the *LinkScheduler* operates according to its Definition 4.3 at time  $t_i$ .  $D_i$  is *PredictValid* by assumption, and  $S_{i+1}$  was shown to be *PredictValid* in step 3 above. The action of *LinkScheduler* operating at time  $t_i$  (Definition 4.3, step 4) ensures that

${}_{xy}D_{i+1} = {}_{xy}D_i + \{-A_{i+1}\} - \{S_{i+1} \cup K_{i+1}\}$ . Thus,  ${}_{xy}D_{i+1}$  is *PredictValid*, by Definition 4.8.

5.  ${}_{xy}L_{i+1}^n$ . This part of the proof is broken into five cases.

**case 1** ( $n = 1$ , X is switch): Assume  ${}_{xy}L_{i+1}^1$  is not *PredictValid*. By Definition

4.10 (case 1), this means that  $\neg({}_{xy}L_{i+1}^1 \approx {}_{xy}S_{i+1})$ . By Definition 4.3 (step 5),

${}_{xy}L_{i+1}^1 \leftarrow {}_{xy}S_{i+1}$ , directly implying that  ${}_{xy}L_{i+1}^1 \equiv {}_{xy}S_{i+1}^1$ . From Definition 4.3

(step 3), we know that  ${}_{xy}S_{i+1}^1 \approx {}_{xy}S_{i+1}$ . Therefore, by substitution, we have

${}_{xy}L_{i+1}^1 \approx {}_{xy}S_{i+1}$ , contradicting our assumption.  ${}_{xy}L_{i+1}^n$  is therefore *PredictValid* by contradiction.

**case 2** ( $n = 1$ , X is an AAL): Assume  ${}_{xy}L_{i+1}^1$  is not *PredictValid*. By Definition

4.10 (case 2), this means that  $\neg({}_{xy}L_{i+1}^1 \approx {}_{xy}\beta_{i+1})$ . By Definition 4.4 (step 3),

${}_{xy}\sigma_{i+1}^1 \approx {}_{xy}\beta_{i+1}$  and (step 4)  ${}_{xy}L_{i+1}^1 \leftarrow {}_{xy}\sigma_{i+1}$ . Therefore, by substitution, we

have  ${}_{xy}L_{i+1}^1 \approx {}_{xy}\beta_{i+1}$ , contradicting our assumption.  ${}_{xy}L_{i+1}^n$  is therefore *PredictValid*, by contradiction.

**case 3** ( $n = 2$ , X is a switch): Assume  ${}_{xy}L_{i+1}^2$  is not *PredictValid*. By Definition

4.10 (case 3), this means that  $\neg({}_{xy}L_{i+1}^2 \approx {}_{xy}L_{i+1}^1)$ . Since  ${}_{xy}L_{i+1}^2 \equiv {}_{xy}L_i^1$  (Definition 4.5) and  ${}_{xy}L_{i+1}^1 \equiv {}_{xy}S_i$  (Definition 4.3), by substitution we have

$\neg({}_{xy}L_i^1 \approx {}_{xy}S_i)$ , which means that  ${}_{xy}L_i^1$  was not *PredictValid* (Definition 4.3).

This contradicts the inductive hypothesis, so  ${}_{xy}L_{i+1}^2$  is *PredictValid*.

**case 4** ( $n = 2$ ,  $X$  is an AAL): Assume  ${}_{xy}^2L_{i+1}^1$  is not *PredictValid*. This means that  $\neg\left({}_{xy}^2L_{i+1}^1 \approx {}_{xy}^1L_{i+1}\right)$ . Since  ${}_{xy}^2L_{i+1}^1 \equiv {}_{xy}^1L_i^1$ , (Definition 4.5), and  ${}_{xy}^1L_{i+1} \equiv {}_{xy}\beta_i$  (Definition 4.3), by substitution we have  $\neg\left({}_{xy}^1L_i^1 \approx {}_{xy}\beta_i\right)$ , which means that  ${}_{xy}^1L_i^1$  was not *PredictValid* (Definition 4.3). This contradicts the inductive hypothesis, so  ${}_{xy}^2L_{i+1}^1$  is *PredictValid*.

**case 5** ( $2 < n \leq \delta_{xy}$ ): Assume  ${}_{xy}^nL_{i+1}^1$  is not *PredictValid*. This means that  $\neg\left({}_{xy}^nL_{i+1}^1 \approx {}_{xy}^{n-1}L_{i+1}\right)$ . Since  ${}_{xy}^nL_{i+1}^1 \equiv {}_{xy}^{n-1}L_i^1$  and  ${}_{xy}^{n-1}L_{i+1} \equiv {}_{xy}^{n-2}L_i$  (Definition 4.5), by substitution we have  $\neg\left({}_{xy}^{n-1}L_i^1 \approx {}_{xy}^{n-2}L_i\right)$ , which means that  ${}_{xy}^{n-1}L_i^1$  was not *PredictValid*. This contradicts the inductive hypothesis, so  ${}_{xy}^nL_{i+1}^1$  is *PredictValid*.

6.  ${}_{xy}^nL_{i+1}$ . We prove this result via four cases.

**case 1** ( $n = 1$ ,  $X$  is switch): Assume that  ${}_{xy}^1L_{i+1}$  is not *PredictValid*. This implies that  $\neg\left({}_{xy}^2L_{i+1}^1 \approx {}_{xy}^1L_{i+1}\right)$ . Then, by Definition 4.3 and Definition 4.5,  $\neg\left({}_{xy}^1L_i^1 \approx {}_{xy}S_i\right)$ . This contradicts the inductive hypothesis that  ${}_{xy}^1L_i^1$  is *PredictValid*, so  ${}_{xy}^1L_{i+1}$  is therefore *PredictValid*.

**case 2** ( $n = 1$ ,  $X$  is an AAL): Assume that  ${}_{xy}^1L_{i+1}$  is not *PredictValid*. This implies that  $\neg\left({}_{xy}^2L_{i+1}^1 \approx {}_{xy}^1L_{i+1}\right)$ . Then, by Definition 4.4 and Definition 4.5,  $\neg\left({}_{xy}^1L_i^1 \approx {}_{xy}\beta_i\right)$ . This contradicts the inductive hypothesis that  ${}_{xy}^1L_i^1$  is *PredictValid*, so  ${}_{xy}^1L_{i+1}$  must be *PredictValid*.

**case 3** ( $1 < n < \delta_{xy}$ ): Assume that  ${}_{xy}^nL_{i+1}$  is not *PredictValid*. Therefore,  $\neg\left({}_{xy}^nL_{i+1} \approx {}_{xy}^{n+1}L_{i+1}^1\right)$ . Then, by Definition 4.5,  $\neg\left({}_{xy}^{n-1}L_i \approx {}_{xy}^nL_i^1\right)$ . This last

expression contradicts the inductive hypothesis that  ${}^{n-1}L_i$  is *PredictValid*, so  ${}^nL_{i+1}$  is *PredictValid*.

**case 4** ( $n = \delta_{xy}$ ): Assume that  ${}^\delta L_{i+1}$  is not *PredictValid*. This means that  $\neg({}^\delta L_{i+1} \approx {}_{xy}B_{i+1}^1)$ . This implies that  $\neg({}^{\delta-1}L_i \approx {}_{xy}L_i^1)$  (Definition 4.5), which contradicts the inductive hypothesis that  ${}^\delta L_i^1$  is *PredictValid*. Therefore,  ${}^\delta L_{i+1}$  must be *PredictValid*.

7.  ${}_{xy}B_{i+1}$ . Assume  ${}_{xy}B_{i+1}$  is not *PredictValid*.

This implies that  $\neg({}_{xy}B_{i+1} \approx {}_{xy}B_i^1)$ . By Definition 4.5,  ${}_{xy}B_{i+1} \equiv {}^\delta L_i$ , and, thus, by substitution,  $\neg({}^\delta L_i \approx {}_{xy}B_i^1)$ . This implies that  ${}^\delta L_i$  is not *PredictValid*, contradicting the inductive hypothesis. Thus,  ${}_{xy}B_{i+1}$  is *PredictValid*.

This completes the induction proof and hence Theorem 1 holds.



## 4.6 Relaxed PredictHorizon

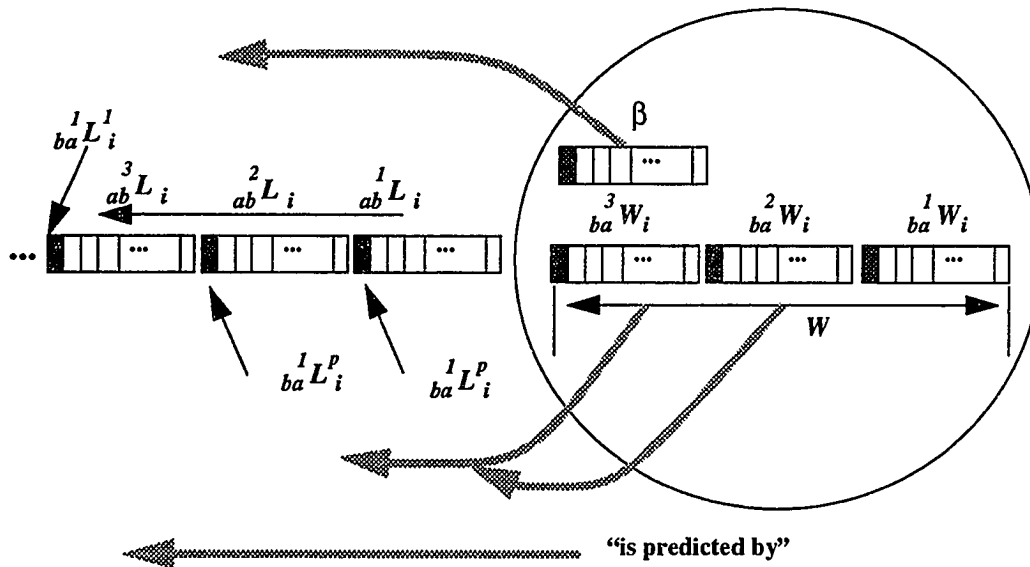
For simplicity, the proof of Theorem 1 assumed a *PredictHorizon* value of 1 (Assumption 4.5). In this section we discuss those modifications to the proof that are required when we relax this assumption. Figures 15 and 16 show how extending *PredictHorizon* to three extends the *W* and *S* buffers to three time cycles each. In general, a *PredictHorizon* of  $n$  requires an  $n$ -period warning shot buffer *W* and an  $n$ -period set *S* of committed schedules.

### 4.6.1 Impact on the Proof

From a high-level perspective, the generalized proof proceeds similarly to the restricted version. The complications arise from the fact that the *S* and *W* structures each consist of multiple instances of their earlier, simple versions. These instances are organized much like a shift register. Conceptually speaking, if we imagine that the simpler proof had a *loader* of *S* that reinitialized *S* each time cycle and a *carry out* from the contents of *S* each time cycle, the *loading* and *carry out* now occurs into and out of the first and last elements of the shift register. The shift register analogy is convenient as each element of *S* is shifted

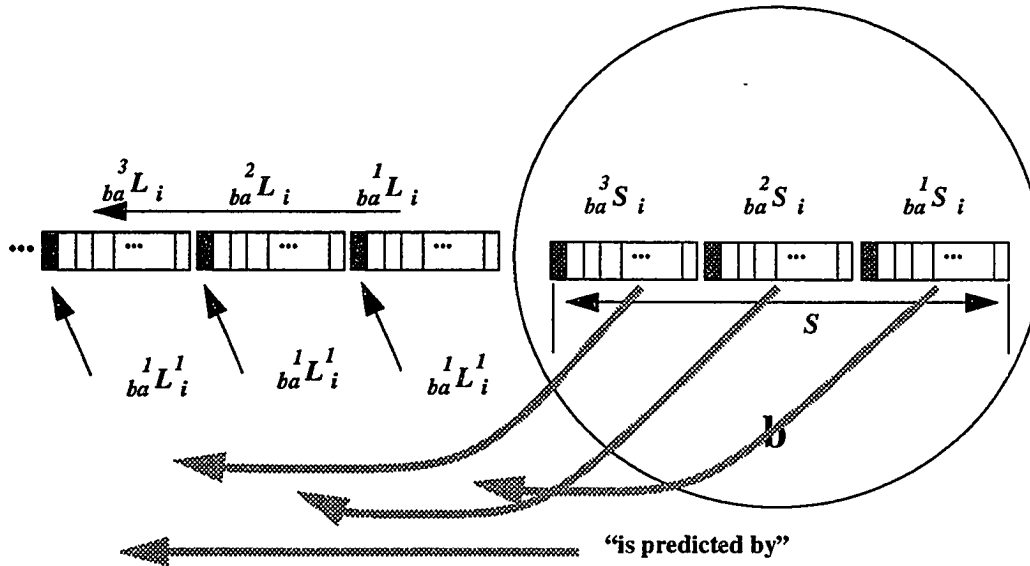
left one position with each time cycle. The shift register analogy holds for  $W$  as well. The definitions of *LinkScheduler* and *AALScheduler* are extended to account for the more complex  $S$  and  $W$ . The proof in Section 4.7 is conducted similarly to the earlier proof, but is based on these more complex definitions.

**FIGURE 15. PredictHorizon of 3 and Warning Shot (W) Buffers**



The other significant difference is the definition of *PredictValidity* for the  $^n L$  and  $B$  components. Since *PredictValidity* of an  $^n L$  component is defined in terms of its relationship to a component  $P$  slots distant, this complicates boundary cases. For example, *PredictValidity* of  $^{\delta}_{xy} L$  is determined by the prediction cell that entered the prediction database  $^D_{xy} P-1$  time cycles earlier. In order to avoid the complexities associated with formalizing this relationship, we alter our definition of *PredictValidity* of the  $^n L$  and  $B$  components so that it is defined in terms of its relationship to a cell spatially collocated, but temporally displaced by  $P$  time units. (See Definition 4.11, case 2.)

FIGURE 16. Generalized PredictHorizon and Committed Schedules (S)



#### 4.6.2 PredictHorizon re: LinkScheduler

The generalized *LinkScheduler* remains largely the same as the one described in Section . The important differences are isolated to steps 1 and 2 below which constitute a “shift left one position, carry into  $^1L$ ,” with respect to the set of committed schedules  $S$ .

##### Definition 4.13

For each output link  $xy$  *LinkScheduler* performs the following steps at  $t_i$  using the scratch-pad  $\sigma$ :

1.  ${}_{xy}S_{i+1} \leftarrow {}^P S_i$
2.  ${}^{n+1}{}_{xy}S_{i+1} \leftarrow {}^n{}_{xy}S_i, 1 < n < P$
3.  $\{ {}^1{}_{xy}S_{i+1}, {}_{xy}K_i \} = f({}_{xy}D_i)$
4. populates  ${}_{xy}S_{i+1}^1$  such that  ${}_{xy}S_{i+1}^1 \approx {}^1S_{i+1}$



5.  $D_{i+1}$  is derived from  $D_i$  by adding any pertinent new predictions ( ${}_{yx}A_{i+1}$ ) received from *PShunt* and by deleting any predictions that have just been scheduled ( ${}^1S_{i+1}$ ) as well as any explicit deletion requests ( $K_{i+1}$ ) from  $f$ . That is,

$${}_{xy}D_{i+1} \leftarrow {}_{xy}D_i + \{{}_{yx}A_{i+1}\} - \{{}^1S_{i+1} \cup {}_{xy}K_{i+1}\}$$

6.  ${}_{xy}L_{i+1} \leftarrow {}_{xy}S_{i+1}$ .

The effect of the  $A$ ,  $S$  and  $K$  terms above is to ensure that only valid predictions are added to the data base ( $A$ ) and that any predictions no longer eligible for scheduling ( $S$ ,  $K$ ) are removed.

#### 4.6.3 PredictHorizon re: AALSchedular

Like the *LinkScheduler*, the difference between the generalized *AALSchedular* and the one described in Section 4.3.3 is the set of Warning Shot Buffers ( $W$ ) which is “shifted one position to the left, carrying in to the  $\beta$  buffer.”

##### Definition 4.14

*AALSchedular* performs the following steps at  $t_i$  using the scratchpad  $\sigma$ :

1.  ${}_{xy}\sigma_{i+1} \leftarrow {}_{xy}\beta_i$
2.  ${}_{xy}\beta_{i+1} \leftarrow {}_{xy}^P W_i$
3. populates  ${}_{xy}\sigma_{i+1}^1$  such that  ${}_{xy}\sigma_{i+1}^1 \approx {}_{xy}W_i$
4.  ${}_{xy}^{n+1} W_{i+1} \leftarrow {}_{xy}^n W_i, 1 < n < P$
5.  ${}_{xy}L_{i+1} \leftarrow {}_{xy}\sigma_{i+1}$
6.  ${}_{xy}W_i \leftarrow \{c_1, c_2, \dots, c_P\}$ , where  $c_k$  is the user cell that arrives at AAL X during the  $k^{th}$  cell time slot of  $\{t_{i-1}, t_i\}$  and  $c_k$  is null if no cell arrived during that slot.

#### 4.6.4 PredictHorizon re: Passive Components and PredictValidity

In this section we define *PredictValidity* for each of the passive network PredictComponents. Each of these are introduced in Section 3.3. Section 4.3.3 explains how the contents of  ${}_{xy}W_i$  and  ${}_{xy}\beta_i$  are derived.

Note that certain caveats are required for the ranges expressed in the following definitions for the special cases of extremely short communications links or very large PredictHorizon (e.g.,  $\delta_{xy} < P$ ). The following definitions and arguments have not attempted to explicitly detail every possible combination of these. When we have not explicitly broken out a case for separate consideration we consider that it should be obvious from the context how that case should be treated. For example, in the case of Definition 4.11, when we stipulate two separate cases ( $1 \leq n < (\delta_{xy} - P)$ ) and ( $(\delta_{xy} - P) \leq n \leq \delta$ ) we expect that it is clear from the context that if  $\delta_{xy} < P$  then there is only one case to consider,  $1 \leq n < \delta_{xy}$ .

##### Definition 4.15

**case 1** ( $n = 1$ ):  ${}^1W_i$  is *PredictValid* if  ${}^1W_i \equiv \{c_1, c_2, \dots, c_P\}$ , where  $c_k$  is the user cell that arrives at AAL X during the  $k^{th}$  cell time slot of  $\{t_{i-1}, t_i\}$  and  $c_k$  is null if no cell arrived during that slot. (This case is essentially unchanged from Definition 4.6.)

**case 2** ( $1 < n \leq P$ ):  ${}^nW_i$  is *PredictValid* if  ${}^nW_i = {}^{n-1}W_{i-1}$ . That is, the warning shot buffer  $W$  functions as a shift register, with each element shifting one position to the left with each time cycle.

##### Definition 4.16

A delay buffer  ${}_{xy}\beta_i$  is *PredictValid* if  ${}_{xy}\beta_i \approx {}^P W_{i-1}$ . This definition closely resembles Definition 4.7, with the significant difference that since there are multiple elements of the  $W$  component, the definition stipulates that the relationship is only with the  $P^{th}$  component.

##### Definition 4.17

A link prediction database  ${}_{xy}D_{i+1}$  is *PredictValid* if

$${}_{xy}D_{i+1} = {}_{xy}D_i + \{-A_i\} - \{{}^1S_{i+1} \cup {}_{xy}K_i\}.$$

(This definition is essentially unchanged from Definition 4.8.)

**Definition 4.18**

**case 1** ( $n = 1$ ): The committed schedule  ${}^1S_{xy}$  is *PredictValid* if  ${}_{xy}D_{i-1}$  is *PredictValid*. A null  ${}^1S_{xy}$  is always *PredictValid*.

(This case is essentially unchanged from Definition 4.9.)

**case 2** ( $1 < n \leq P$ ): The committed schedule  ${}^nS_{xy}$  is *PredictValid* if

$${}^nS_{xy} = {}^{n-1}S_{xy}.$$

That is, the set  $S$  of schedule skeletons functions like a shift register, with each element shifting one position to the left with each time cycle.

**Definition 4.19**

**case 1** ( $1 \leq n \leq P$ ,  $X$  is a switch): The prediction cell  ${}^nL_{xy}^1$  is *PredictValid* if

$${}^nL_{xy}^1 \approx {}^nS_{xy}.$$

This means that the first  $P$  prediction cells on the communications link predict cells are currently scheduled in schedule skeletons still active in the switch.

**case 2** ( $1 \leq n < P$ ,  $X$  is an AAL): The prediction cell  ${}^nL_{xy}^1$  is *PredictValid* if

$${}^nL_{xy}^1 \approx {}^{n+1}W_{xy}.$$

This means that the first  $P-1$  prediction cells on the communications link predict cells currently scheduled in warning shot buffers still active in the AAL.

**case 3** ( $n = P$ ,  $X$  is an AAL): The prediction cell  ${}^PL_{xy}^1$  is *PredictValid* if

$${}^PL_{xy}^1 \approx {}_{xy}\beta.$$

That is, the  $P^{th}$  prediction cell predicts those cells currently scheduled in the AAL delay buffer  $\beta$ .

**case 4** ( $P < n \leq \delta_{xy}$ ): The prediction cell  ${}^nL_{xy}^1$  is *PredictValid* if  ${}^nL_{xy}^1 \approx {}^{n-P}L_{xy}$ .

A prediction cell that has propagated at least  $P$  cycles from the switch predicts cells that are currently on that same communications link, located  $P$  cell groups closer to the switch. (Note: If  $P > \delta_{.xy}$ , then cases 3 and 4 do not exist. Additionally, the range of cases 1 and 2 would then be restricted to  $1 \leq n \leq \delta_{.xy}$ .)

**Definition 4.20**

**case 1** ( $1 \leq n < (\delta_{.xy} - P)$ ): A cell group  ${}^nL_i$  is *PredictValid* if  ${}^nL_i \approx {}^{n+P}L_i^1$ .

A currently propagating cell group that is at least  $P$  cycles away from arrival at the next switch is predicted by the prediction cell located on the same communications link, located  $P$  cell groups closer to that remote switch.

**case 2** ( $(\delta_{.xy} - P) \leq n \leq \delta$ ): A cell group  ${}^nL_i$  is *PredictValid* if  ${}^nL_i \approx {}^nL_{i-P}^1$ .

A currently propagating cell group that is  $P$  or fewer cycles away from arrival at the next switch is predicted by the prediction cell in the cell group located in the same position on the communications link  $P$  cycles earlier.

**Definition 4.21**

A delay buffer  ${}_{xy}B_i$  is *PredictValid* if  ${}_{xy}B_i \approx {}_{xy}B_{i-P}^1$ .

That is, the contents of a switch delay buffer  $B$  are predicted by the prediction cell that entered the delay buffer  $P$  cycles earlier.

**4.7 The Relaxed Prediction Theorem**

In this section we prove Theorem 2, which we stated in Section 4.1.

**Base Case**

We prove that the composite network PredictState is *PredictValid* at  $t_0$  by demonstrating that the individual passive PredictComponents are all *PredictValid*:

1.  ${}_{xy}W_0$ .

**case 1** ( $n = 1$ ):  ${}^nW_0$  is empty at  $t_0$  (Assumption 4.1(iii)). Since no user cell can have arrived in  $\{t_{i-1}, t_i\}$ , this is *PredictValid* (Definition 4.15).

**case 2** ( $1 < n \leq P$ ):  ${}^nW_0 \equiv {}^{n-1}W_{-1} \equiv \phi$  (Assumption 4.1(iii, iv)). Thus,  ${}^nW_0$  is *PredictValid* (Definition 4.15).

2.  ${}_{xy}\beta_0$ .

Both  ${}_{xy}\beta_0$  and  ${}^P W_{-1}$  are empty at  $t_0$  (Assumption 4.1(ii) and Assumption 4.4, respectively). Thus, since  ${}_{xy}\beta_0 \equiv {}^P W_{-1} \equiv \phi$ ,  ${}_{xy}\beta_0$  is *PredictValid* (Definition 4.16).

3.  ${}_{xy}D_0$ . (argument unchanged from Section )

${}_{xy}D_0$  is empty at  $t_0$  (Assumption 4.1(iv)). Since  ${}_{xy}D_{-1}$ ,  ${}_{yx}A_{-1}$ ,  ${}^1S_0$ , and  ${}_{xy}K_{-1}$  are empty (Assumption 4.4),  ${}_{xy}D_0 \equiv {}_{xy}D_{-1} + \{{}_{yx}A_0\} - \{{}^1S_0 \cup {}_{xy}K_0\} \equiv \phi$ . Thus,  ${}_{xy}D_0$  is *PredictValid* (Definition 4.17).

4.  ${}_{xy}S_0$ .

**case 1** ( $n = 1$ ):  ${}^1S_0$  is empty at  $t_0$  (Assumption 4.1(v)). Thus,  ${}^1S_0$  is *PredictValid* (Definition 4.18).

**case 2** ( $1 < n \leq P$ ):  ${}^nS_0 \equiv {}^{n-1}S_{-1} \equiv \phi$  (Assumption 4.1(v), Assumption 4.4). Thus,  ${}^nS_0$  is *PredictValid* (Definition 4.18).

5.  ${}_{xy}L_0^1$ .

**case 1** ( $1 \leq n \leq P$ , X is a switch): Since  ${}_{xy}L_0^1 \approx {}^nS_0 \approx \phi$  (Assumption 4.1(i,v)),  ${}_{xy}L_0^1$  is *PredictValid* (Definition 4.19).

**case 2** ( $1 \leq n < P$ , X is an AAL): Since  ${}_{xy}L_0^1 \approx {}^{n+1}W_0 \approx \phi$  (Assumption 4.1(i, iii)),  ${}_{xy}L_0^1$  is *PredictValid* (Definition 4.19).

**case 3** ( $n = P$ ,  $X$  is an AAL): Since  ${}_{xy}L_0^P \approx {}_{xy}\beta_0 \approx \phi$  (Assumption 4.1(i,ii)),  ${}_{xy}L_0^P$  is *PredictValid* (Definition 4.19).

**case 4** ( $P < n \leq \delta_{xy}$ ): Since  ${}_{xy}L_0^n \approx {}_{xy}L_0^{n-P} \approx \phi$  (Assumption 4.1(i)),  ${}_{xy}L_0^n$  is *PredictValid* (Definition 4.19).

6.  ${}_{xy}L_0^n$ .

**case 1** ( $1 \leq n < (\delta_{xy} - P)$ ): Since  ${}_{xy}L_0^n \approx {}_{xy}L_0^{n+P} \approx \phi$  (Assumption 4.1(i)),  ${}_{xy}L_0^n$  is *PredictValid* (Definition 4.20).

**case 2** ( $(\delta_{xy} - P) \leq n \leq \delta$ ): Since  ${}_{xy}L_0^n \approx {}_{xy}L_{-P}^1 \approx \phi$  (Assumption 4.1(i), Assumption 4.4),  ${}_{xy}L_0^n$  is *PredictValid* (Definition 4.20).

7.  ${}_{xy}B_0$ .

Since  ${}_{xy}B_0 \approx {}_{xy}B_{-P}^1 \approx \phi$  (Assumption 4.1(ii) and Assumption 4.4),  ${}_{xy}B_0$  is *PredictValid* (Definition 4.21).

This concludes the base case.

### Inductive Hypothesis

The network *PredictState* is *PredictValid* at  $t_k$ ,  $0 \leq k \leq i$ .

At time  $t_{i+1}$ , the three active *PredictComponents*, *PShunt*, *LinkScheduler* and *AALSched-uler*, operate and perform the steps given in Definition 4.2, Definition 4.13 and Definition 4.14, respectively. We need to show that the composite network *PredictState* is *PredictValid* at  $t_{i+1}$  by demonstrating that the individual passive *PredictComponents* are all *PredictValid* at  $t_{i+1}$ .

### Inductive Step

1.  ${}_{xy}W_{i+1}^n$ .

**case 1** ( $n = 1$ ): By the action of the *AALSched-uler* (Definition 4.14, step 6),  ${}_{xy}W_{i+1}^1$  is *PredictValid* (Definition 4.15).

**case 2** (  $1 < n \leq P$  ): By the action of the *AALScheduler* (Definition 4.14, step 4),

${}_{xy}^n W_{i+1}$  is *PredictValid* (Definition 4.15).

2.  ${}_{xy} \beta_{i+1}$ .

By the action of the *AALScheduler* (Definition 4.4, step 2),  ${}_{xy} \beta_{i+1}$  is *PredictValid* (Definition 4.16).

3.  ${}_{xy}^n S_{i+1}$ .

**case 1** ( $n = 1$ ):  ${}_{xy} D_i$  is *PredictValid* by the inductive hypothesis. Therefore, by Definition 4.18,  ${}_{xy}^n S_{i+1}$  is *PredictValid*.

**case 2** (  $1 < n \leq P$  ): By the action of the *LinkScheduler* (Definition 4.13, step 2),

${}_{xy}^n S_{i+1}$  is *PredictValid* (Definition 4.18).

4.  ${}_{xy} D_{i+1}$ . (*argument unchanged from Section* )

$A_{i+1}$  and  $K_{i+1}$  are automatically *PredictValid* as part of our assumption that the *LinkScheduler* operates according to its Definition 4.13 at time  $t_i$ .  $D_i$  is *PredictValid* by assumption, and  $S_{i+1}$  is shown to be *PredictValid* in step 3 above. The action of *LinkScheduler* operating at time  $t_i$  (Definition 4.13, step 4) ensures that  ${}_{xy} D_{i+1} = {}_{xy} D_i + \{ {}_{yx} A_{i+1} \} - \{ {}_{xy}^1 S_{i+1} \cup {}_{xy} K_{i+1} \}$ . Thus,  ${}_{xy} D_{i+1}$  is *PredictValid*, by Definition 4.17.

5.  ${}_{xy}^n L_{i+1}^1$ . We prove this via eight cases.

**case 1** ( $n = 1$ , X is a switch): Assume  ${}_{xy}^1 L_{i+1}^1$  is not *PredictValid*. This means that  $\neg ({}_{xy}^1 L_{i+1}^1 \approx {}_{xy}^1 S_{i+1})$ . This explicitly contradicts the action of *LinkScheduler* (Definition 4.13, step 4,6) at time  $t_i$ . Therefore,  ${}_{xy}^1 L_{i+1}^1$  is *PredictValid*, by contradiction.

**case 2** ( $n = 1$ ,  $X$  is an AAL): Assume  ${}^1L_{xy}^1$  is not *PredictValid*. This means that  $\neg({}^1L_{xy}^1 \approx {}^2W_{xy})$ . By the action of *AALScheduler* at time  $t_i$ , however, we know both that  ${}^1L_{xy}^1 \approx {}^1W_{xy}$  (Definition 4.14, step 3) and that  ${}^2W_{xy} \equiv {}^1W_{xy}$  (Definition 4.14, step 4). Substituting, this yields  ${}^1L_{xy}^1 \approx {}^2W_{xy}$ , contradicting our assumption. Thus,  ${}^1L_{xy}^1$  is *PredictValid*.

**case 3** ( $1 < n \leq P$ ,  $X$  is switch): Assume  ${}^nL_{xy}^1$  is not *PredictValid*. By Definition 4.19 (case 1), this means that  $\neg({}^nL_{xy}^1 \approx {}^nS_{xy})$ . Propagation delay over  $\{t_p, t_{i+1}\}$  causes  ${}^nL_{xy}^1 \equiv {}^{n-1}L_{xy}^1$  (Definition 4.5) and, by the definition of the *LinkScheduler*,  ${}^nS_{xy} \equiv {}^{n-1}S_{xy}$  (Definition 4.13, step 2). Therefore, by substitution, we have  $\neg({}^{n-1}L_{xy}^1 \approx {}^{n-1}S_{xy})$ , which means that  ${}^{n-1}L_{xy}^1$  was not *PredictValid* (Definition 4.19). This contradicts the inductive hypothesis, so  ${}^nL_{xy}^1$  is *PredictValid*.

**case 4** ( $1 < n < P$ ,  $X$  is an AAL): Assume  ${}^nL_{xy}^1$  is not *PredictValid*. By Definition 4.19 (case 2), this means that  $\neg({}^nL_{xy}^1 \approx {}^{n+1}W_{xy})$ . Since  ${}^nL_{xy}^1 \equiv {}^{n-1}L_{xy}^1$  (Definition 4.5) and  ${}^{n+1}W_{xy} \equiv {}^nW_{xy}$  (Definition 4.14, step 4), by substitution we have  $\neg({}^{n-1}L_{xy}^1 \approx {}^nW_{xy})$ , which means that  ${}^{n-1}L_{xy}^1$  was not *PredictValid* (Definition 4.19). This contradicts the inductive hypothesis, so  ${}^nL_{xy}^1$  is *PredictValid*.

**case 5** ( $n = P$ ,  $X$  is an AAL): Assume  ${}^PL_{xy}^1$  is not *PredictValid*. By Definition 4.19 (case 3), this means that  $\neg({}^PL_{xy}^1 \approx {}_xy\beta_{i+1})$ . Propagation delay over  $\{t_p, t_{i+1}\}$  causes  ${}^PL_{xy}^1 \equiv {}^{P-1}L_{xy}^1$  (Definition 4.5) and the action of the *AALScheduler* causes  ${}_xy\beta_{i+1} \equiv {}^PW_{xy}$  (Definition 4.14, step 2). Therefore, by sub-



stitution we have  $\neg \left( \begin{smallmatrix} P-1 \\ xy \end{smallmatrix} L_i^1 \approx \begin{smallmatrix} P \\ xy \end{smallmatrix} W_i \right)$ , which means that  $\begin{smallmatrix} P-1 \\ xy \end{smallmatrix} L_i^1$  was not *PredictValid* (Definition 4.19, case 2). This contradicts the inductive hypothesis, so  $\begin{smallmatrix} P-1 \\ xy \end{smallmatrix} L_{i+1}^1$  is *PredictValid*.

**case 6** ( $n = P + 1$ , X is a Switch): Assume  $\begin{smallmatrix} P+1 \\ xy \end{smallmatrix} L_{i+1}^1$  is not *PredictValid*. By Definition 4.19 (case 4), this means that  $\neg \left( \begin{smallmatrix} P+1 \\ xy \end{smallmatrix} L_{i+1}^1 \approx \begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i+1} \right)$ . We know that the

action of the *LinkScheduler* (Definition 4.13, steps 1, 6) at time  $t_i$  causes

$\begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i+1} \approx \begin{smallmatrix} P \\ xy \end{smallmatrix} S_i$ . The repeated actions of the *LinkScheduler* (Definition 4.13, step 2)

over the period  $\{t_{i-p}, t_{i-1}\}$  cause  $\begin{smallmatrix} P \\ xy \end{smallmatrix} S_i \equiv \begin{smallmatrix} 1 \\ xy \end{smallmatrix} S_{i-p+1}$ . Propagation delay (Definition 4.5) over  $\{t_{i-p}, t_i\}$  causes

$\begin{smallmatrix} P+1 \\ xy \end{smallmatrix} L_{i+1}^1 \equiv \begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i-p+1}^1$ . Therefore, by substitution, we have  $\neg \left( \begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i-p+1}^1 \approx \begin{smallmatrix} 1 \\ xy \end{smallmatrix} S_{i-p+1} \right)$ . We know by the action of

*LinkScheduler* at  $t_{i-p}$  (Definition 4.13, steps 4, 6) that  $\begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i-p+1}^1 \approx \begin{smallmatrix} 1 \\ xy \end{smallmatrix} S_{i-p+1}$ , contradicting our assumption that  $\begin{smallmatrix} P+1 \\ xy \end{smallmatrix} L_{i+1}^1$  is not *PredictValid*.

**case 7** ( $n = P + 1$ , X is AAL): Assume  $\begin{smallmatrix} P+1 \\ xy \end{smallmatrix} L_{i+1}^1$  is not *PredictValid*. By Definition 4.19 (case 4), this means that  $\neg \left( \begin{smallmatrix} P+1 \\ xy \end{smallmatrix} L_{i+1}^1 \approx \begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i+1} \right)$ . Propagation delay

(Definition 4.5) over  $\{t_{i-p}, t_i\}$  causes  $\begin{smallmatrix} P+1 \\ xy \end{smallmatrix} L_{i+1}^1 \equiv \begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i-p+1}^1$ . We know that the action of the *AALScheduler* (Definition 4.14, steps 1, 5) at time  $t_i$  causes

$\begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i+1} \approx \begin{smallmatrix} 1 \\ xy \end{smallmatrix} \beta_i$ , and at time  $t_{i-1}$  causes  $\begin{smallmatrix} 1 \\ xy \end{smallmatrix} \beta_i \approx \begin{smallmatrix} P \\ xy \end{smallmatrix} W_{i-1}$ . The repeated actions of the *AALScheduler* (Definition 4.14, step 4) over the period  $\{t_{i-p}, t_{i-1}\}$  cause

$\begin{smallmatrix} P \\ xy \end{smallmatrix} W_{i-1} \equiv \begin{smallmatrix} 1 \\ xy \end{smallmatrix} W_{i-p}$ . Therefore, by substitution, we have  $\neg \left( \begin{smallmatrix} 1 \\ xy \end{smallmatrix} L_{i-p+1}^1 \approx \begin{smallmatrix} 1 \\ xy \end{smallmatrix} W_{i-p} \right)$ .

We know by the action of *AALScheduler* at  $t_{i-p}$  (Definition 4.14, steps 3, 5) that

${}^1L_{xy}^1{}_{i-P+1} \approx {}^1W_{xy}{}_{i-P}$ , contradicting our assumption that  ${}^{P+1}L_{xy}^1{}_{i+1}$  is not *PredictValid*.

**case 8** ( $P+1 < n \leq \delta_{xy}$ ): Assume  ${}^nL_{xy}^1{}_{i+1}$  is not *PredictValid*. By Definition 4.19

(case 4), this means that  $\neg\left({}^nL_{xy}^1{}_{i+1} \approx {}^{n-P}L_{xy}^1{}_{i+1}\right)$ . Propagation delay (Definition

4.5) over  $\{t_i, t_{i+1}\}$  implies that both  ${}^nL_{xy}^1{}_{i+1} \equiv {}^{n-1}L_{xy}^1{}_{i+1}$  and

${}^{n-P}L_{xy}^1{}_{i+1} \equiv {}^{n-P-1}L_{xy}^1{}_{i+1}$ . Therefore, by substitution we have

$\neg\left({}^{n-1}L_{xy}^1{}_{i+1} \approx {}^{n-P-1}L_{xy}^1{}_{i+1}\right)$ , which means that  ${}^{n-1}L_{xy}^1{}_{i+1}$  was not *PredictValid* (Definition

4.19, case 4). This contradicts the inductive hypothesis, so  ${}^nL_{xy}^1{}_{i+1}$  is *PredictValid*.

6.  ${}^nL_{xy}^1{}_{i+1}$ . We prove this via four cases.

**case 1** ( $n = 1$ , X is a switch): Assume that  ${}^1L_{xy}^1{}_{i+1}$  is not *PredictValid*. By Definition

4.20 (case 1), this means that  $\neg\left({}^{1+P}L_{xy}^1{}_{i+1} \approx {}^1L_{xy}^1{}_{i+1}\right)$ . We know that by Definition

4.5  ${}^{P+1}L_{xy}^1{}_{i+1} \equiv {}^P L_{xy}^1{}_{i+1}$ . By the action of the *LinkScheduler* (Definition

4.13, step 6),  ${}^1L_{xy}^1{}_{i+1} \approx {}^P S_{xy}^1{}_{i+1}$ . Thus, by substitution,  $\neg\left({}^P L_{xy}^1{}_{i+1} \approx {}^P S_{xy}^1{}_{i+1}\right)$ . This contra-

dicts the inductive hypothesis that  ${}^P L_{xy}^1{}_{i+1}$  is *PredictValid* (Definition 4.19), so

${}^1L_{xy}^1{}_{i+1}$  is *PredictValid* by contradiction.

**case 2** ( $n = 1$ , X is an AAL): Assume that  ${}^1L_{xy}^1{}_{i+1}$  is not *PredictValid*. By Definition

4.20 (case 1), this means that  $\neg\left({}^{1+P}L_{xy}^1{}_{i+1} \approx {}^1L_{xy}^1{}_{i+1}\right)$ . We know that by Definition

4.5  ${}^{P+1}L_{xy}^1{}_{i+1} \equiv {}^P L_{xy}^1{}_{i+1}$ . Also, by the action of the *AALScheduler*

(Definition 4.14, steps 3, 5),  ${}^1L_{xy}^1{}_{i+1} \approx {}_{xy}\beta_i$ . Thus, by substitution, this means that

$\neg\left({}^P L_{xy}^1{}_{i+1} \approx {}_{xy}\beta_i\right)$ . This contradicts the inductive hypothesis that  ${}^P L_{xy}^1{}_{i+1}$  is *Pre-*

*dictValid* (Definition 4.19), so  ${}^1L_{xy}^1{}_{i+1}$  must be *PredictValid*.

**case 3** ( $1 < n < (\delta_{xy} - P)$ ): Assume that  ${}^nL_{xy\ i+1}$  is not *PredictValid*. Therefore, by Definition 4.20 (case 1),  $\neg\left({}^nL_{xy\ i+1} \approx {}^{n+P}L_{xy\ i+1}^1\right)$ . Then, due to the effect of propagation delay over  $\{t_p, t_{i+1}\}$  (Definition 4.5),  $\neg\left({}^{n-1}L_{xy\ i} \approx {}^{n+P-1}L_{xy\ i}^1\right)$ . This last expression contradicts the inductive hypothesis that  ${}^{n-1}L_{xy\ i}$  is *PredictValid*, so  ${}^nL_{xy\ i+1}$  is *PredictValid*.

**case 4** ( $(\delta_{xy} - P) \leq n \leq \delta$ ): Assume that  ${}^nL_{xy\ i+1}$  is not *PredictValid*. Therefore, by Definition 4.20 (case 2),  $\neg\left({}^nL_{xy\ i+1} \approx {}^nL_{xy\ i+1-P}^1\right)$ . It then follows that propagation delay over  $\{t_p, t_{i+1}\}$  (Definition 4.5) means that  $\neg\left({}^{n-1}L_{xy\ i} \approx {}^{n-1}L_{xy\ i-P}^1\right)$ , which contradicts the inductive hypothesis that stated that  ${}^{n-1}L_{xy\ i}$  is *PredictValid*. Therefore,  ${}^nL_{xy\ i+1}$  must be *PredictValid*, by contradiction.

7.  ${}_{xy}B_{i+1}$ . Assume  ${}_{xy}B_{i+1}$  is not *PredictValid*.

Therefore, by Definition 4.21, this implies that  $\neg\left({}_{xy}B_{i+1} \approx {}_{xy}B_{i+1-P}^1\right)$ . Propagation delay over  $\{t_p, t_{i+1}\}$  (Definition 4.5) from link  $xy$  into the delay buffer  ${}_{xy}B$  results in  ${}_{xy}B_{i+1} = {}_{xy}L_i^\delta$ . Propagation delay over  $\{t_p, t_{i+1}\}$  results in  ${}_{xy}B_{i+1-P}^1 = {}_{xy}L_{i-P}^\delta$ . Substituting,  $\neg\left({}_{xy}L_i^\delta \approx {}_{xy}L_{i-P}^\delta\right)$ . This implies that  ${}_{xy}L_i^\delta$  is not *PredictValid* (Definition 4.20, case 2), contradicting the inductive hypothesis. Thus,  ${}_{xy}B_{i+1}$  is *PredictValid*.

This concludes the proof of Theorem 2.



# Chapter 5

## COMPLEXITY OF PREDICTION-BASED SCHEDULING

### 5.1 Introduction

Throughout this dissertation we have described scheduling as the problem of determining a feasible cell schedule that incurs the minimum  $QoSCost$ . The problem is formally stated as a decision problem in the next section. We call this general problem the *Complete QoS Scheduling Problem* (CQSP). Our goal in this chapter is to study the computational complexity of CQSP. We do so by first illustrating that the *Simple QoS Scheduling Problem* (SQSP) is NP-complete. We then reduce SQSP to CQSP. Since CQSP is also in NP, this shows CQSP is NP-complete. In summary, this chapter contains two main theorems. They are stated below.

#### Theorem 3

*The Simple QoS Scheduling Problem is NP-complete.*

#### Theorem 4

*The Complete QoS Scheduling Problem is NP-complete.*

The fact that the scheduling problem is intractable forces us to use heuristics that achieve good, but sub-optimal schedules. Several prediction-based scheduling heuristics that seem fruitful are introduced in Chapter 6 and tested in Chapter 7.

## 5.2 NP-completeness

One of the most common forms of NP-completeness proofs utilizes *local replacement*. A detailed discussion on NP-completeness proofs by local replacement can be found in Garey and Johnson [1]. The basic strategy of these proofs is to define a reduction from a known NP-complete problem to the target problem. The reduction demonstrates that the target problem is also intractable, that is, it is not in the class of polynomial time solvable problems (P). This is because one can solve the original problem using a subroutine, the reduction, for the target problem. This says that the target problem is at least as hard as the original one. Thus, if the original problem was NP-complete, the target problem would be NP-hard. Further, if the target problem were also in NP, it would be NP-complete.

Certain rules must be followed in this kind of proof. They are illustrated below.

1. The reduction must be from the general case of the known NP-complete problem.
2. The reduction must be performed in polynomial time (in the size of the input to the original problem).
3. While the transformation must be from the general case of the known NP-complete problem, the transformation may be to a sub-problem of the target problem.

The rationale behind this is that if a polynomial-time solution exists to the target problem, this solution must necessarily include the sub-problem that emerges from the translation. Therefore, a polynomial-time solution for the target problem implies a polynomial-time solution for the general case of the known NP-complete problem. Thus, unless  $P=NP$  the target problem is NP-hard and again, if it is in NP then it is NP-complete.

Another form of NP-completeness proof is *proof by restriction* [1]. In this style of proof one shows that the target problem contains an instance that itself is identical to a known NP-complete problem.

Many different scheduling problems have been proven NP-complete. Garey and Johnson provide a brief review of over 22 different NP-complete scheduling problems [1]. The problem we choose to transform to the SQSP problem is called *Sequencing Within Intervals* (SWI). It should become clear below that we have selected this problem since it displays many of the characteristics of our problem, making the transformation straightforward. As with many NP-completeness results, choosing the correct problem to reduce to is key to the overall proof.

### 5.2.1 Sequencing Within Intervals Problem (SWI)

Garey and Johnson [1] define SWI as follows:

#### Input

The input consists of 1) a finite set  $T$  of “tasks,” 2) for each  $t \in T$ , an integer “release time,”  $r(t) \geq 0$ , 3) a “deadline”  $d(t) \in Z^+$ , and 4) a “length”  $l(t) \in Z^+$ .

#### Question

Does there exist a *feasible schedule*  $\sigma$  for  $T$ , such that each task  $t$  is “executed” from time  $\sigma(t)$  to time  $\sigma(t) + l(t)$ , cannot begin execution until time  $r(t)$ , must be completed by time  $d(t)$ , and its execution cannot overlap the execution of any other task  $t'$ ?

That is, does there exist a function  $\sigma: T \rightarrow Z^+$ , such that:

1. for each  $t \in T$ ,  $\sigma(t) \geq r(t)$ ,  $\sigma(t) + l(t) \leq d(t)$
2. if  $t' \in T - \{t\}$ , then either  $\sigma(t') + l(t') \leq \sigma(t)$  or  $\sigma(t') \geq \sigma(t) + l(t)$ .

### 5.3 The Simple Quality of Service Scheduling Problem (SQSP)

We define SQSP formally as follows:

#### Input

The input consists of a finite set  $C$  of schedulable cells. These input cells are partitioned into a finite number  $P$  of subsets called “Connections”. For each connection  $i$ ,  $1 \leq i \leq P$ , there is a finite subset of schedulable cells  $C_i \subseteq C$  and, for each  $j, c_j \in C_i$ ,  $1 \leq j \leq |C_i|$ , an integer “release time,”  $r_{j,c_j} \geq 0$ , and a “deadline”  $d_{j,c_j} \in Z^+$ . For each connection  $i$ , there is also a maximum intercell time  $j(C_i) \geq 0$ .

#### Question

Does there exist a *feasible schedule*  $\sigma$  for  $C$ , such that the cell  $j, c_j$  is scheduled for transmission at time  $\sigma_{j,c_j}$ , cannot begin transmission until time  $r_{j,c_j}$ , and must be completed by time  $d_{j,c_j}$ ? Two cells may not be scheduled for the same slot and cells from a given connection must be scheduled in sequence. Each cell occupies exactly one time slot in the schedule. If a cell is not the first cell in a burst, then it must be scheduled within  $j(C_i)$  time slots of the earlier cell.

That is, does there exist a function  $\sigma: C \rightarrow Z^+$ , such that:

1. for each  ${}_j c_i \in C$ ,  $1 \leq j \leq |C_i|$ ,  $1 \leq i \leq P$ ,  $r({}_j c_i) \leq \sigma({}_j c_i) \leq d({}_j c_i)$ ;
2. if  $c' \in C - \{c\}$ , then either  $\sigma(c') < \sigma(c)$  or  $\sigma(c') > \sigma(c)$ ;
3. if  $m > n$ , then  $\sigma({}_m c_i) > \sigma({}_n c_i)$ ;
4. if  $r({}_j c_i) - r({}_{j-1} c_i) > j(C_i)$ ,  ${}_j c_i$  is designated the first cell of a burst; when  ${}_j c_i$  is *not* the first cell of a burst,  $\sigma({}_j c_i) \leq \sigma({}_{j-1} c_i) + j(C_i)$ ;  $j(C_i) = 0$  is equivalent to  $j(C_i) = \infty$ ?

To explicitly derive all possible schedules to accurately answer this question using a straightforward approach requires  $O(2^n)$  time where  $n = |C|$ . This computation is clearly not feasible as a real-time ATM cell scheduler. We now prove that it is unlikely that a polynomial time algorithm for SQSP will be found.

### 5.3.1 SQSP is NP-complete

We transform SWI to SQSP. For notational convenience, we refer to the original tasks of the SWI problem as  $t_i$ ,  $1 \leq i \leq |T|$ .

1. For each task  $t_i$  we create a corresponding set of cells for connection  $i$ . This set of cells is called  $C_i$ . We define  $C_i$  to act as a block of cells that must be scheduled in contiguous slots in order to resemble  $t_i$ . To this end, we define exactly  $l(t_i)$  cells for connection  $i$ . We refer to the individual input cells as  ${}_j c_i$ ,  $1 \leq i \leq |T|$ ,  $1 \leq j \leq l(t_i)$ .
2. We also require that the block  $C_i$  obey similar scheduling constraints to the original task  $t_i$ . Specifically, we wish for the block  $C_i$  to enjoy (suffer) precisely the same scheduling freedom (restrictions) as the task  $t_i$ . In other words, the first cell of  $C_i$  should be able to be scheduled no earlier than slot  $r(t_i)$ , the final cell of  $C_i$  must be scheduled no later than  $d(t_i)$ , and the cells must be scheduled contiguously. Specifically, this implies that  $r({}_j c_i) = r(t_i) + j - 1$ ,  $d({}_j c_i) = d(t_i) - (l(t_i) - j)$ , and that  $j(C_i) = 1$ .

This reduction can be performed in polynomial time in the input size  $|I|$  of the original problem. Suppose the original instance  $I$  of SWI had size  $a$ . Assuming  $M$  equal to  $\max(d(t_i) - r(t_i))$ ,  $1 \leq t \leq |T|$ , the reduction requires the creation of at most  $M$  SQSP input cells for each task  $T$  in  $I$ . These increase the size by no more than the constant factor  $M$ , so

the instance  $I'$  of SQSP has size  $O(a)$ , a polynomial in the size of  $I$ . There is a small constant computational cost  $K$  associated with the creation of each input to our instance of SQSP. The computational cost of the translation is bounded by  $|T| \times M \times K$ , which is clearly a polynomial in the size of the original problem.

All aspects of the original SWI problem are embodied in the translation. Therefore, a solution to the translated SQSP problem exists if and only if there exists a feasible schedule for the SWI problem. Since our translation encompasses the general case of the SWI problem, this suffices to show that SQSP is NP-hard. It is also easy to see that SQSP is in NP. One can guess a schedule in polynomial time and then verify that it is feasible using polynomial time. Thus, SQSP is NP-complete.

■

## 5.4 The Complete Quality of Service Scheduling Problem (CQSP)

The CQSP problem is defined to reflect the computational complexity of the scheduling problem presented to each *LinkScheduler* at the start of each network cycle. We define CQSP formally as follows:

### Input

The input consists of a finite set  $C$  of schedulable cells and a constant  $K$ . These input cells are partitioned into a finite number  $P$  of subsets called “Connections”. For each connection  $i$ ,  $1 \leq i \leq P$ , there is a finite subset of schedulable cells  $C_i \subseteq C$  and, for each  ${}_j c_i \in C_i$ ,  $1 \leq j \leq |C_i|$ , an integer “release time,”  $r({}_j c_i) \geq 0$ , and a “deadline”  $d({}_j c_i) \in Z^+$ . For each connection  $i$ , there is also a maximum intercell time  $j(C_i) \geq 0$ .

### Question

A *skeleton schedule*  $\sigma$  for  $C$  is an assignment of cells to time slots such that the cell  ${}_j c_i$  is scheduled for transmission at time  $\sigma({}_j c_i)$ , cannot begin transmission until time  $r({}_j c_i)$ , and must be completed by time  $d({}_j c_i)$ . Two cells may not be scheduled for the same slot and cells from a given connection must be scheduled in sequence. Each cell occupies exactly one time slot in the schedule. If a cell is not the first cell in a burst, then it must be scheduled within  $j(C_i)$  time slots of the earlier cell. Any cell not thus scheduled belongs to the set  $L$ . Does there exist a *skeleton schedule*  $\sigma$  for  $C$  such that  $|L| \leq K$  for some arbitrary  $K$ ?

That is, does there exist a function  $\sigma: C \rightarrow Z^+$ , such that:



1. for each  $c_i \in C$ ,  $1 \leq i \leq |C_i|$ ,  $1 \leq j \leq P$ ,  $r(c_i) \leq \sigma(c_i) \leq d(c_i)$ ;
2. if  $c' \in C - \{c\}$ , then either  $\sigma(c') < \sigma(c)$  or  $\sigma(c') > \sigma(c)$ ;
3. if  $m > n$ , then  $\sigma_m(c_i) > \sigma_n(c_i)$ ;
4. if  $r(c_i) - r_{j-1}(c_i) > j(C_i)$ ,  $c_i$  is designated the first cell of a burst; when  $c_i$  is *not* the first cell of a burst,  $\sigma(c_i) \leq \sigma_{j-1}(c_i) + j(C_i)$ ;  $j(C_i) = 0$  is equivalent to  $j(C_i) = \infty$ ;
5.  $|L| \leq K$  for arbitrary  $K$ ?

We now prove that it is unlikely that a polynomial time algorithm for this problem will be found.

#### 5.4.1 CQSP is NP-complete

We prove that CQSP is NP-complete via a proof by restriction. Specifically, we show that CQSP contains the NP-complete problem SQSP as a special case where  $K = 0$ .

The input to CQSP and SQSP are the same. Upon inspection CQSP differs from SQSP in only one fundamental way: rather than asking for the minimum cost schedule, we simply ask if there exists a schedule whose cost is zero (i.e.  $|L| = 0$ ). The fact that CQSP is *harder* is explained by the following argument: if we know the cost of a minimum cost feasible schedule, then we can easily (i.e. in polynomial time) know whether that minimum cost is zero. The fact that a polynomial-time solution to CQSP implies a polynomial-time solution to SQSP means that if SQSP is NP-complete, then CQSP is also NP-complete.

■

### 5.5 Summary

This chapter has demonstrated that it is unlikely that we will find an algorithm to efficiently solve a simplified version of the QoS scheduling problem, SQSP. This fact provides strong evidence of the unlikelihood of the existence of an efficient algorithm for the generation of a QoS-optimal cell schedule in the general case of CQSP. While the time available for calculation of cell transmission schedules is greater in our model than in the generic ATM model, the fact that schedule computation is still very time constrained renders the computational complexity of the problem even more significant. The difficulty of

the general problem and the limited computation time available lead us to restrict the problem in order to develop efficient heuristics that can compute good, but sub-optimal schedules.

In Section 3.11 we extended the basic model so that unschedulable cells need not be dropped but can be buffered in the prediction data base  $D$ . The proof of correctness in Chapter 4 and the proof of complexity argued here assume that *LinkScheduler* has the latitude to schedule any cell still in  $D$ . In the terminology of CQSP, this means that the set of schedulable cells  $C$  includes all cells present in  $D$ . The proofs also assume that *LinkScheduler* can consider all candidate slots for scheduling a cell, subject to that cell's particular constraints. One of the key elements in the heuristics discussed in Chapter 6 is that we restrict the scheduling problem so that the number of new cells that have to be scheduled is limited to the number of cells that can arrive in one cycle. While the actual number of cells available for scheduling *could* include all the cells in the prediction database  $D$ , we generally *freeze* cells into a tentative schedule after their initial scheduling. This restriction restricts the input and problem of CQSP in two fundamental ways: 1) the set of schedulable cells  $C$  is limited to the new arrivals, and 2) once a cell is placed in the tentative schedule it is not considered again for rescheduling (it may, however, be removed from the tentative schedule). Depending on the heuristics and a particular cell's scheduling constraints, the heuristics may also restrict the candidate slots considered to a small subset of the possible candidates. (Note that the heuristics work under the constraint that the thus-restricted scheduling problem needs to be solved within one cycle time.)

While this method does not necessarily discover the optimal schedule, it does greatly reduce the number of possible schedules considered by the algorithm. The simulation results in Chapter 7 demonstrate that while the different heuristics based on this key assumption do not guarantee a QoS-optimal schedule, they do provide QoS-sensitive schedules that out-perform a generic cell-scheduler.

## 5.6 References

- [1] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

# Chapter 6

## EXPOSITION OF THE SCHEDULING ALGORITHMS

### 6.1 Introduction

Thus far we have introduced and developed a general network model that provides accurate, distributed predictions of cell arrivals. This model creates the opportunity to investigate an entire family of cell scheduling techniques that exploit these predictions. In this chapter, we detail the specific scheduling heuristics that we formalized and evaluated as part of this research. Since these ideas are necessarily a small subset of the family of schedulers based on our model, it is our hope that other researchers may use the model as a springboard for investigation of other schedulers in this prediction-based family.

$LinkScheduler_{xy}(P, T_i)$  is a function operating on node  $X$  which takes a tentative cell transmission schedule  $P$  from the set of contributing  $PShunt_{v,x}$  for link  $xy$  targeted for time period  $T_i$  and returns a committed transmission schedule  $S$ . These committed schedules  $S$  are guaranteed to be enforceable on link  $xy$  for time period  $T_i$ . *Enforceable* means that all scheduled cells will be available for transmission when they are due and that no two cells are scheduled for the same cell time slot. In order to achieve this enforceability, some cells may have to be dropped. It is the task of *LinkScheduler* to select cells to be dropped from the arrival schedule so that the number of resulting QoS violations is minimized. *LinkScheduler* operates at cycle-time granularity, in parallel, for all links in the switch.

$Enforcer_{xy}$  operates at cell-time granularity, in parallel, for all links  $xy$  in the switch. At each cell time,  $Enforcer_{xy}$  transmits the cell indicated in  $S$  onto link  $xy$ . In the same cell time  $Enforcer_{xy}$  checks the cell that emerges from  $B_{yx}$  to see if it was marked for deletion. If so,  $Enforcer_{xy}$  discards the cell.

$LinkScheduler_{xy}(P, T_i)$  does not normally drop cells from  $P$  to cause  $xy$  to be starved during  $T_i$ . As  $LinkScheduler$  decides which cells to drop, each such decision is made by dropping a cell from the connection that has the maximum *DistanceFromQoSViolation* of the connections contributing cells to  $P$ . In this thesis, *DistanceFromQoSViolation* is an instantaneous measure of tolerance of cell loss, delay or jitter. For example, a connection that can definitely afford to lose a cell during  $T_i$  would have a large *DistanceFromQoSViolation* value whereas a connection that would incur a QoS violation if a single cell is lost during  $T_i$  would have a small *DistanceFromQoSViolation*. As explained earlier in Section 2.4, loss and delay should both receive consideration by the scheduling algorithm. Jitter must also be considered, despite the fact that much of the research has skirted jitter management as too difficult. Simultaneously providing for loss, delay and jitter guarantees while striving for high network utilization seems very complex. The quality of connection-specific loss, delay, and jitter guarantees as well as the computational complexity of providing them depends largely on how we define the *DistanceFromQoSViolation* metric. In Section 6.6 we propose different *DistanceFromQoSViolation* metrics that we believe can be feasibly implemented. Chapter 7 provides experimental results from simulations of a selection of these metrics.

In Section 6.3 we extend the simple definition of the *LinkScheduler* function so that it takes as an additional input a list of tentative schedules that will be buffered from time periods preceding  $T_i$  and may contend for link  $xy$  during  $T_i$ . These tentative schedules are called *schedule skeletons*. *LinkScheduler* would be permitted to delete cells from those skeletons either by scheduling the cell for time period  $T_i$  or by deciding that further buffering of the cell was pointless due to it having caused a QoS violation.

## 6.2 Baseline LinkScheduler

The baseline *LinkScheduler* is a simple greedy algorithm. It consists of the following steps:

1. Determine how many cells ( $n$ ) need to be pruned from  $P$  to render it congestion-free.

2. If any cells are found that belong to VC's without a guarantee of maximum loss, delete up to  $n$  of those cells. If further deletions are necessary, continue.
3. Sort all VC's of member cells by their current *DistanceFromQoSViolation* value. Delete a cell from the VC with the current maximum *DistanceFromQoSViolation* value, recomputing the affected VC's *DistanceFromQoSViolation* after the deletion, and resorting the list. Continue deleting from the VC at the head of the list until the required  $n$  cells have been deleted.

Step 3 above stipulates that waiting cells should be sorted. The heuristics we describe in this chapter (Section 6.4) do not actually *sort*. Instead, they use a dynamic programming approach where each step only modifies the existing solution (schedule) as is required to avoid QoS violations. When cell loss is inevitable, they search for a minimal cost cell to discard.

We can expand the information available to the baseline scheduler both by 1) extending the set of eligible cells to include some cells predicted in earlier cycles and also by 2) incorporating QoS information from other switches in the network. We discuss these two extensions to simple scheduling in the next section.

## 6.3 Expanded LinkScheduler

### 6.3.1 Allowing for Cell Queuing

In practical ATM networks, some cell buffering must be allowed if the network is to operate without high cell loss. This is due to the fact that when more arrivals converge on an output link than can be served in one time interval, those that are not immediately scheduled must wait or be dropped. Since ATM switches will have some amount of buffer space to hold these cells, they queue. We discuss two methods of cell queuing in this section. The first, *Future Time Slot Assignment (FTSA)* is intended for delay and jitter sensitive traffic (CBR and jitter-sensitive VBR). The second, *Deferred Queue (Defer\_Q)* is intended for delay and jitter tolerant traffic (ABR, UBR and jitter-tolerant VBR).

#### 6.3.1.1 Future Time Slot Assignment (FTSA)

*FTSA* is built upon the notion of *schedule skeletons* mentioned in the introduction to this chapter. *FTSA* uses the skeletons much like an arithmetic 'carry' in the sense that they are the result of an earlier calculation of *LinkScheduler* yet contribute to the current calculation.

With *FTSA*, a prediction is immediately scheduled by assigning it a time slot if possible in one of the available schedule skeletons. We denote the  $i^{\text{th}}$  skeleton as  $\kappa_i$ ,  $1 \leq i \leq \textit{SkeletonDepth}$ . The value of *SkeletonDepth* and the cell buffer space available jointly determine the maximum number of cells that may queue via this method. The value of *SkeletonDepth* should therefore be related to the buffer space available on the switch. If the scheduler takes care to not fully populate the skeletons, then *SkeletonDepth* is not directly limited by buffer space and is only limited by how far into the future it desires to schedule cells.

We thus extend the definition of *LinkScheduler* to be the function  $\textit{LinkScheduler}_{xy}(P, T_i, C)$ . This function is defined as in the introduction to this chapter with the additional argument  $C$  denoting a list of the preceding *SkeletonDepth* schedule skeletons for link  $xy$ . These skeletons are the result of previous invocations of  $\textit{LinkScheduler}_{xy}(P, T_i, C)$ . The modified *LinkScheduler* has the following steps:

1.  $P$  is merged into  $\kappa_i$ ,  $1 \leq i \leq \textit{SkeletonDepth}$ .
2. If the merge from the previous step required dropping any cells, schedule those cells to be deleted upon arrival. (*Enforcer* is responsible for this deletion.) The predictions of the dropped cells are removed from  $\kappa$ .
3.  $\kappa_1$  becomes a committed schedule ( $\mathcal{S}$ ).
4.  $\kappa_i \leftarrow \kappa_{i+1}$ ,  $1 \leq i < \textit{SkeletonDepth}$
5.  $\kappa_{\textit{SkeletonDepth}} \leftarrow \phi$

Allowing for some cell queueing should increase network utilization with no increase in QoS violations. This increase is a benefit of the statistical multiplexing performed within the framework of our QoS-based scheduler. The aggregate of the predictions in the skeletons can be thought of as a *hypothetical future queue* of cells. Decisions relating to buffer occupancy have to be based on this future queue. (This requires the simulation of the future queue that we describe in Section 6.5.2.2.)

The baseline scheduler can generate a schedule for time  $T_i$  that is guaranteed to be optimal based on the limited information available. This “optimality” is misleading, however, since there is a strong argument to allow more scheduling freedom than does the baseline scheduler. The baseline scheduler is simple precisely because, as we illustrated in Figure 8 on page 43, it allows no queueing across cycle boundaries and since it allows no such

queueing, overall cell loss is higher. That simplicity erodes when *FTSA* queueing is allowed, where both the scheduling options and the amount of available information can make computation of the truly optimal schedule computationally prohibitive (see Chapter 5). We believe that this added complexity is justified by the reduced cell loss.

In addition to adding complexity, allowing *FTSA* queueing exhibits a stochastic aspect not present in our baseline scheduler. For example, if the scheduler trades off transmission of a delay-tolerant cell for transmission of a delay-intolerant cell, there is a non-deterministic ‘guess’ about the likelihood of having a free cell slot for this cell within a few time periods. *It should be obvious such a decision involves simultaneous comparison of delay tolerance and loss tolerance, and represents a real challenge to the computation of our DistanceFromQoSViolation metric.*

If *LinkScheduler* never displaces an already-scheduled cell from a skeleton other than to discard it, we can establish an absolute upper bound on the queueing delay experienced by a cell in any switch as *SkeletonDepth* time units. (If *SkeletonDepth* is one, *FTSA* decomposes into the simple scheduling introduced in Section 3.8.3 on page 42. Since the strength of *FTSA* is control over cell delay, it is more appropriate for CBR and jitter-sensitive VBR traffic than for other classes.)

While the presence of the carried cells complicates the computation, the manner in which our proposed Prediction/Scheduling cycle decouples schedule generation from cell service can provide the freedom to perform these more complex computations. This has an advantage over other schedulers appearing in the literature (e.g., Push-Out, MARS) that make scheduling decisions only about currently buffered cells. Such schedulers can afford very little time for the scheduling decision without potentially starving the communications resource.

If only *FTSA* queueing is in effect, all predictions in the prediction database *D* reside exclusively in the schedule skeletons.

### **6.3.1.2 Deferred FIFO Queueing**

The reality of ABR traffic is that while it is nominally loss tolerant, the transport layer protocols that recover from cell loss exhibit very poor end-to-end throughput when cell loss occurs in the network [1] [6]. To address this, a whole new sub-field of ATM research has surfaced. This research generally presumes that ATM switches with very large cell buffers will be available. In this section we augment *FTSA* with *Defer\_Q* to allow exploitation of

large-buffer switches for jitter-tolerant traffic like ABR while still benefitting from the strengths of FTSA for jitter-sensitive traffic.

While *FTSA* is an effective means of maintaining tight control over queueing delay, it has drawbacks when applied to a traffic class that does not require delay guarantees. This is illustrated by the example of a large burst of jitter-tolerant cells arriving just before a burst of delay-sensitive cells. Strict *FTSA* will try to commit the jitter-tolerant cells to specific time slots upon receipt of their predictions. There is no way that this scheduling activity can anticipate whether it is better to schedule the jitter-tolerant cells as densely as possible or whether to leave ‘holes’ for possible future delay-sensitive arrivals. It is clearly wrong to leave scheduling ‘holes’ if there is no overlapping delay-sensitive traffic, since this produces useless network delay for the other cells. On the other hand, if the jitter-tolerant cells are prematurely and densely scheduled, and subsequent delay-sensitive arrivals force displacement of those jitter-tolerant cells, this displacement is computationally costly. For this reason we define *Defer\_Q* scheduling that applies only to delay-tolerant traffic.

When *Defer\_Q* queueing is used, yet-unscheduled predictions of delay-tolerant cells are scheduled in prediction slots that would otherwise go unused by *FTSA*. These *Defer\_Q*-scheduled cells are thus actually predicted and scheduled in fixed schedules as are the *FTSA*-scheduled cells and, thus, still fit nicely into our scheduling paradigm. *Defer\_Q*-style predictions will only be scheduled in a prediction cycle for which the *FTSA*-style scheduling has completed. Once predicted, their actual transmission is controlled by *Enforcer* just like *FTSA*-queued cells. *Defer\_Q* does not afford control of queueing delay as does *FTSA* and is thus inappropriate for jitter-sensitive traffic. (Note that augmenting *FTSA* with *Defer\_Q* queueing does not alter the computational complexity of *LinkScheduler*.)

When both *FTSA* and *Defer\_Q* are in effect, the prediction data base  $D$  is the composite of  $\kappa$  and those cells waiting on the *Defer\_Q* output link-based queue.

### 6.3.2 Global Knowledge of QoS State

None of the available research on QoS guarantees appears to address the fact that the quality of service provided by the network is really only perceived at the network traffic boundary. Work has focussed on providing the desired loss and delay characteristics at a single link on the Virtual Path of the connection. In this section we argue that it is unreasonable to base decisions about minimizing *QoS Cost* on local information alone.



When an upstream node drops a cell, downstream nodes must be aware of this loss in order to maintain their local *DistanceFromQoSViolation* measure for the affected virtual connection. If this is not done, a downstream node minimizes *QoS Cost* in ignorance of upstream clipping which clearly leads to inaccuracies. For this reason, the local knowledge about *DistanceFromQoSViolation* needs to be *downstream-global*. We define downstream-global as the following: any knowledge about a QoS violation for  $VC_i$  at time  $t_k$  at node  $U$  must also reside in all downstream (with respect to  $U$ ) hops of  $VC_i$  *by the time any predictions of subsequent cells of  $VC_i$  arrive at those hops*. ‘Subsequent cells’ means any cells belonging to  $VC_i$  that arrive at  $U$  after  $t_k$ . We now explain an efficient means of maintaining downstream globality with respect to knowledge about cell loss.

A VC is not a candidate for the *QoS Cost* trade-offs that *LinkScheduler* makes unless at least one cell in the predictions being analyzed belongs to that VC. In general, the candidate VC’s for the scheduler represent a small subset of the set of active VC’s over the  $O_X^l$  being scheduled. Thus, an upstream QoS violation on  $VC_i$  cannot possibly influence a downstream *LinkScheduler* for  $O_X^l$  until  $VC_i$  becomes a candidate at  $X$ , and that does not occur until the prediction for some cell of  $O_X^l$  arrives at  $X$ .

Each  $VC_i$  on a node  $U$  tallies any outstanding QoS violations on that VC as those violations occur. This number is propagated downstream in the next prediction that includes a cell of  $VC_i$ . Thus, assuming that a QoS violation occurs in  $O_U^l$  at  $t_k$ , if the next cell prediction from  $O_U^l$  on  $VC_i$  occurs at  $t_{k+c}$ , no downstream *LinkScheduler* will be informed of that QoS violation until the prediction for that cell arrives at that scheduler. While the downstream *LinkScheduler* could potentially benefit from earlier knowledge of the QoS violation(s), we deem that benefit to be slight in comparison to the implementation elegance of piggybacking the upstream QoS knowledge in the downstream-flowing predictions. We feel that it is therefore reasonable to withhold notification of QoS violation at  $O_U^l$  on  $VC_i$  until  $VC_i$  is predicted to forward a cell on  $O_U^l$ .

*Upstream globality* cannot be defined and implemented symmetrically to downstream globality. Since we cannot rely on reverse direction traffic to permit propagation of outstanding QoS violations upstream, we define upstream globality as a best-effort globality; if there is an unused prediction slot in a prediction cell, it may be used to propagate loss information. We acknowledge that in a simplex VCI (i.e. a VCI where the user cells flow in one direction only), waiting for such unused slots to propagate information upstream may well be the only way to distribute global information upstream, and that we cannot place any bound on how long it may take for this information to percolate upstream. We

discuss specifics of our downstream/upstream globality implementation in Section 6.5.3. Propagation of QoS violation information in upstream-bound prediction cells presumes that prediction cells employ unique VC identifiers between any neighboring pair of switches, regardless of the direction of flow of the virtual circuit. (The notion of Global QoS State described above does not contemplate global knowledge of a cell's cumulative delay. This seems to be a very difficult problem and we currently offer no ideas in this area.)

Since our Global QoS State knowledge implementation may compete with *Defer\_Q* scheduling for the use of empty prediction slots to propagate downstream or upstream loss information, the scheduling policy needs to determine whether an unused prediction slot should be used for loss propagation or waiting ABR traffic. It is not obvious which of these should receive priority. (The schedulers implemented in this research give priority to waiting traffic when such contention exists.)

## 6.4 Heuristic LinkSchedulers

Table 2 provides a summary of the LinkScheduler heuristics that we studied in the course of this research. We describe each of these individually in the following sections. Table 2 includes two non-prediction-based schedulers, *NORMFIFO* and *HOLDISP*, that are used later for purposes of comparison. *NORMFIFO* is a simple single-FIFO scheduler. The *HOLDISP* scheduler implements HOL priority queue service as described in Section 2.4.2, and additionally allows an arriving high priority cell to *displace* a waiting low priority cell in the case of a buffer full condition. In our application of *HOLDISP*, jitter sensitive cells are high priority cells and all other cells are low priority cells.

### 6.4.1 Predictive FIFO (PFIFO)

*PFIFO* is a prediction-based FIFO scheduler based exclusively on *FTSA* queueing. Predictions are processed and the corresponding cells are transmitted in strict first-in-first-out order. When an arrival encounters a buffer full condition, that arrival is dropped without regard to *QoSCost*.

**TABLE 2. Scheduler Summary**

Heuristic Name	Prediction-Based	Loss-Sensitive	Jitter-Sensitive	Global Knowledge	Intelligent Displacement
<i>PFIFO</i>	yes	no	no	no	no
<i>PFIFO_D_NG</i>	yes	yes	no	no	yes
<i>PFIFO_D</i>	yes	yes	no	yes	yes
<i>PTDM</i>	yes	no	yes	no	no
<i>PTDM_D</i>	yes	yes	yes	yes	yes
<i>NORMFIFO</i>	no	no	no	no	no
<i>HOLDISP</i>	no	yes	yes	no	no

#### 6.4.2 Predictive FIFO with Displacement (*PFIFO\_D*)

*PFIFO\_D* is a prediction-based FIFO scheduler based exclusively on *FTSA* queueing. *Displacement* here implies that when a cell cannot be scheduled due to a buffer full condition, a *QoS*Cost minimization function is used to determine which, if any, existing cell may be dropped at lower *QoS*Cost than the cell in question. If a lower cost candidate is identified, the predicted cell is scheduled in its stead, and that lower cost cell is deleted when it arrives at the switch. If no lower cost cell is found, the predicted cell is scheduled for deletion.

*PFIFO\_D\_NG* is identical to *PFIFO\_D* with the exception that no use is made of global QoS knowledge.

#### 6.4.3 Pseudo-Time Division Multiplexing (*PTDM*)

In *Pseudo-Time Division Multiplexing (PTDM)* scheduling, cells are scheduled on a best-effort basis as closely as possible to their nominal interarrival time. That is, it attempts to schedule arriving predictions such that each cell is transmitted at precisely the nominal interarrival interval *after* the previous cell on that VCI. If that ideal slot is occupied, the search expands on either side of that ideal time until either a free slot is found or the cell cannot be scheduled within the defined jitter tolerance. We refer to this search as *jitter-bracketing*.

Successfully scheduled cells will queue according to the *FTSA* model. When jitter-sensitive traffic cannot be scheduled within its defined jitter tolerance, it is dropped. Unsuccessfully scheduled jitter-tolerant traffic, on the other hand, will be queued by *PTDM* in

accordance with the *Defer\_Q* model. When any arriving cell cannot be buffered due to lack of space, it is summarily dropped without considering displacement of an existing cell.

The manner in which *PTDM* can artificially delay a cell is called *cell-spacing*. Results reported in [4] indicate that rate-based policing alone is inadequate to guarantee no network buffer overflow and that cell spacing can significantly mitigate the overflow problem. Both [4] and [2] emphasize that cell spacing is essential for precise jitter control. *PTDM* implements a simpler cell spacing algorithm than the one offered in [4].

*PTDM* presumes that all connections will be characterized by a nominal interarrival time. Nominal interarrival time may assume values between the minimum intercell spacing of a VBR burst to the average cell rate averaged over the long term. It is obvious that for CBR traffic these two extremes are one in the same.

#### **6.4.4 Pseudo-Time Division Multiplexing with Displacement (PTDM\_D)**

*Pseudo-Time Division Multiplexing with Displacement (PTDM\_D)*, which is based on *PTDM*, attempts to combine the loss-robustness of *PFIFO\_D* and the jitter robustness of *PTDM*. It differs from *PTDM* in that when an arrival cannot be buffered due to lack of space, *PTDM\_D* will displace an existing cell whose loss incurs lower *QoS*Cost. Unlike *PTDM*, only the initial attempt at *FTSA* scheduling is made. If this attempt fails, there is no *Defer\_Q* fall-back as in the case of *PTDM*. The displacement strategy is explained below.

During the jitter-bracketing portion of the scan, two conditions can require cell displacement. The first is that no free slot is found at all within the jitter bracket. The second is that, although a free time slot is identified, scheduling the cell would result in one more cell being buffered than the switch buffer can hold. Thus, even in the latter case, the jitter-bracket scan proceeds to the jitter-bracket boundary. In this scan, the minimum-cost cell is identified and displaced (dropped). *PTDM\_D* uses the same cost function as *PFIFO\_D*. Note that *PTDM\_D* assumes that if any connection in the jitter bracket can tolerate one more loss without incurring a loss violation, it is preferable to drop that cell than to cause a jitter violation.

We also considered the idea that if no cell within the bracket range can be dropped without a violation, that we ‘grow’ the bracket size dynamically just for this one prediction. The motivation is that we can exceed jitter violations upstream at a highly congested node and

still have the opportunity to bring them back within their jitter tolerance limits at a less congested downstream scheduler. In our simulator implementation of *PTDM\_D*, we opted for a compromise approach where the actual bracket size used was a fixed ‘forgiveness’ constant added to the connection’s tolerance. This was much easier to implement and achieved the goal of surviving the transit of the highly congested node.

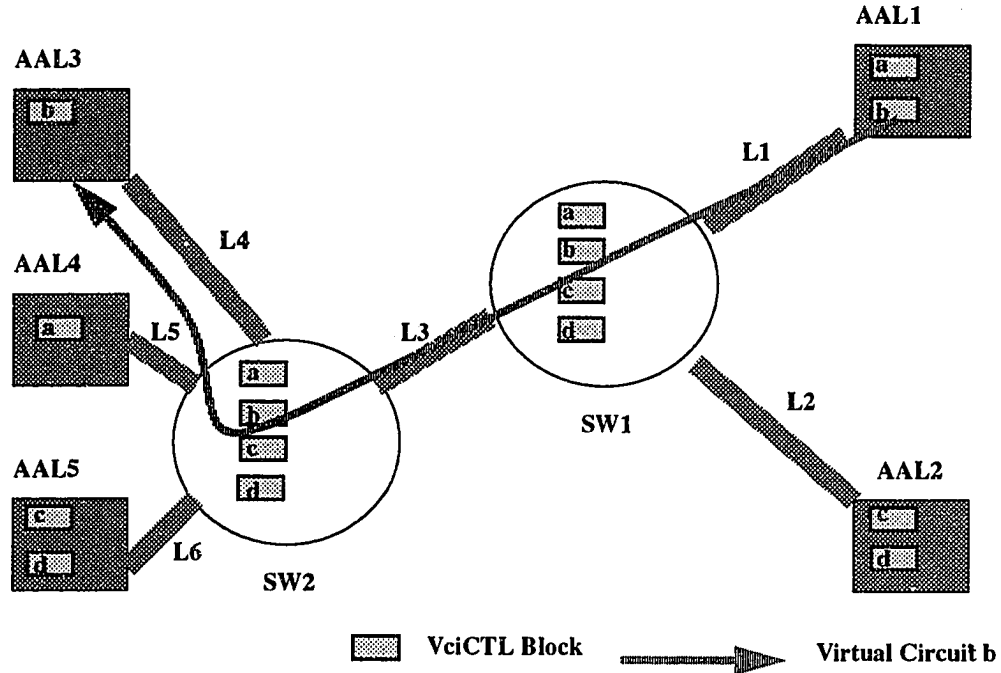
## 6.5 Implementation of the Heuristics

In this section we present a high-level review of the composition and use of the principal data structures we used to implement prediction-based cell scheduling. These structures include the Virtual Circuit Control Block (VciCTL), Output Link Control Blocks (Neighbor Blocks), committed schedules ( $S$ ), schedule skeletons ( $\kappa$ ) and Prediction Cells. The VciCTL structures provide an anchor for the connection-specific control that we have argued is essential for good QoS performance. The Neighbor Blocks are the repositories of the prediction data bases  ${}_{xy}D_i$  used by the *LinkSchedulers*. Neighbor Blocks also contain  $S$  and  $\kappa$ . Prediction cells, of course, define the format of the prediction information transmitted between neighbors on which our entire model is based.

### 6.5.1 Virtual Circuit Control Block

The Virtual Circuit Control Block (VciCTL) as we propose it here is actually not a new requirement of prediction-based scheduling, but rather an extension to a data structure that is arguably an integral part of any functional ATM switch’s data base. Such a structure is necessary for most call admission schemes. (*It is possible, however, for call admission to be done on a VPI basis rather than a VCI basis.*) These structures form the ‘glue’ that joins the separate pieces of a virtual circuit. For example, in Figure 17 the VciCTL blocks for virtual circuit ‘b’ present in SW1 and SW2 embody the virtual circuit’s path from L1 to L3 to L4.

FIGURE 17. VciCTL Blocks in the Network



While there is definitely a generic need for a virtual circuit control structure, prediction-based cell scheduling imposes special requirements. Figure 18 provides a high-level, graphical description of the VciCTL block we use, emphasizing those components that are specific to prediction scheduling itself, or to the QoS guarantees that the prediction-based cell scheduler attempts to provide. We discuss the individual components of the VciCTL structure below.

1. *cellq*. Upon arrival at the switch, cells enter the delay buffer  $B$  described earlier. Upon emerging from the delay buffer, cells wait on their VCI-specific *cellq* until their scheduled time of transmission arrives. The scheduled time is maintained in the schedule skeletons or in a committed schedule  $S$ , if the cell can be transmitted without queueing. (See Figures 20-22.) When no queueing is occurring on the output link, the cell is removed from *cellq* immediately. This is accomplished by the *Enforcer* component of LinkScheduler enforcing a schedule  $S$ .
2. *await\_schedq*. (Used only by LinkScheduler for *Defer\_Q* scheduling.) Under *Defer\_Q* scheduling, predictions that cannot be scheduled immediately are placed on the output link *await\_schedq*. When a prediction cell is about to be sent with unused time slots, *Defer\_Q* scheduling fills these slots with cells waiting on these

VCI-specific *await\_schedqs*. This type of scheduling is particularly suited to ABR and UBR traffic classes. The simulations conducted in this research serve in a fixed priority order the many eligible VCI *await\_schedqs* which are possible.

3. *await\_deletq*. When the scheduler determines that a cell has to be dropped, it is, of course, making that decision on a prediction of that cell's arrival. Thus, it actually has to schedule the deletion. This deletion scheduling is accomplished by entering a deletion request on that VCI's *await\_deletq*, noting the predicted arrival time of the cell. *This queue must be maintained in sorted order, using arrival time as the sort key.* When cells emerge from a switch delay buffer  $B_{yx}$ ,  $Enforcer_{xy}$  checks that cell's *await\_deletq*. If the current time is equal to or greater than the time in the head of the *await\_deletq*, then that cell is dropped and the queue entry deleted.
4. *cell\_losses\_this\_interval*. This statistic measures how many cell drops have occurred in the current measurement interval. The current measurement interval is defined to begin at the current value of *last\_mi\_start* and continues for *qos\_loss\_par2* microseconds. At the end of each such interval, *last\_mi\_start* is set to the current time. This statistic is part of the QoS maintenance function.
5. *qos\_loss\_par1*. This parameter determines the loss tolerance of the connection. Specifically, if more than *qos\_loss\_par1* cell drops occur per *qos\_loss\_par2* microseconds, the switch considers this a loss violation and increments *tot\_loss\_violations* for that VCI.
6. *qos\_loss\_par2*. This statistic is part of the QoS maintenance function. (See *cell\_losses\_this\_interval* and *qos\_loss\_par1* for details.) A value of zero indicates that no loss guarantee is requested.
7. *qos\_dly\_par1*. This statistic represents the nominal cell interarrival time for this connection. For jitter sensitive CBR and VBR traffic, this statistic is assumed to be the desired interarrival time. Such jitter sensitive traffic expresses this by providing a non-zero jitter-tolerance parameter, *qos\_dly\_par2*. For the *PTDM*-family *LinkSchedulers*, any cell whose interarrival time differs from *qos\_dly\_par1* by more than *qos\_dly\_par2* microseconds is considered a jitter violation and *tot\_dly\_violations* is incremented for that VCI.
8. *qos\_dly\_par2*. This statistic is part of the QoS maintenance function. (See *qos\_dly\_par1* for details.) A value of zero indicates that no jitter guarantee is requested.
9. *tot\_loss\_violations*. (See *cell\_losses\_this\_interval* and *qos\_loss\_par1* for details.) Note that each switch and AAL function maintains the *tot\_loss\_violation* and *tot\_dly\_violation* statistics locally. Since we are primarily concerned with end-to-

end QoS, *tot\_dly\_violation* results provided in this thesis are taken from the destination AAL function (unless stated otherwise). These statistics provide a measure of end-to-end QoS performance.

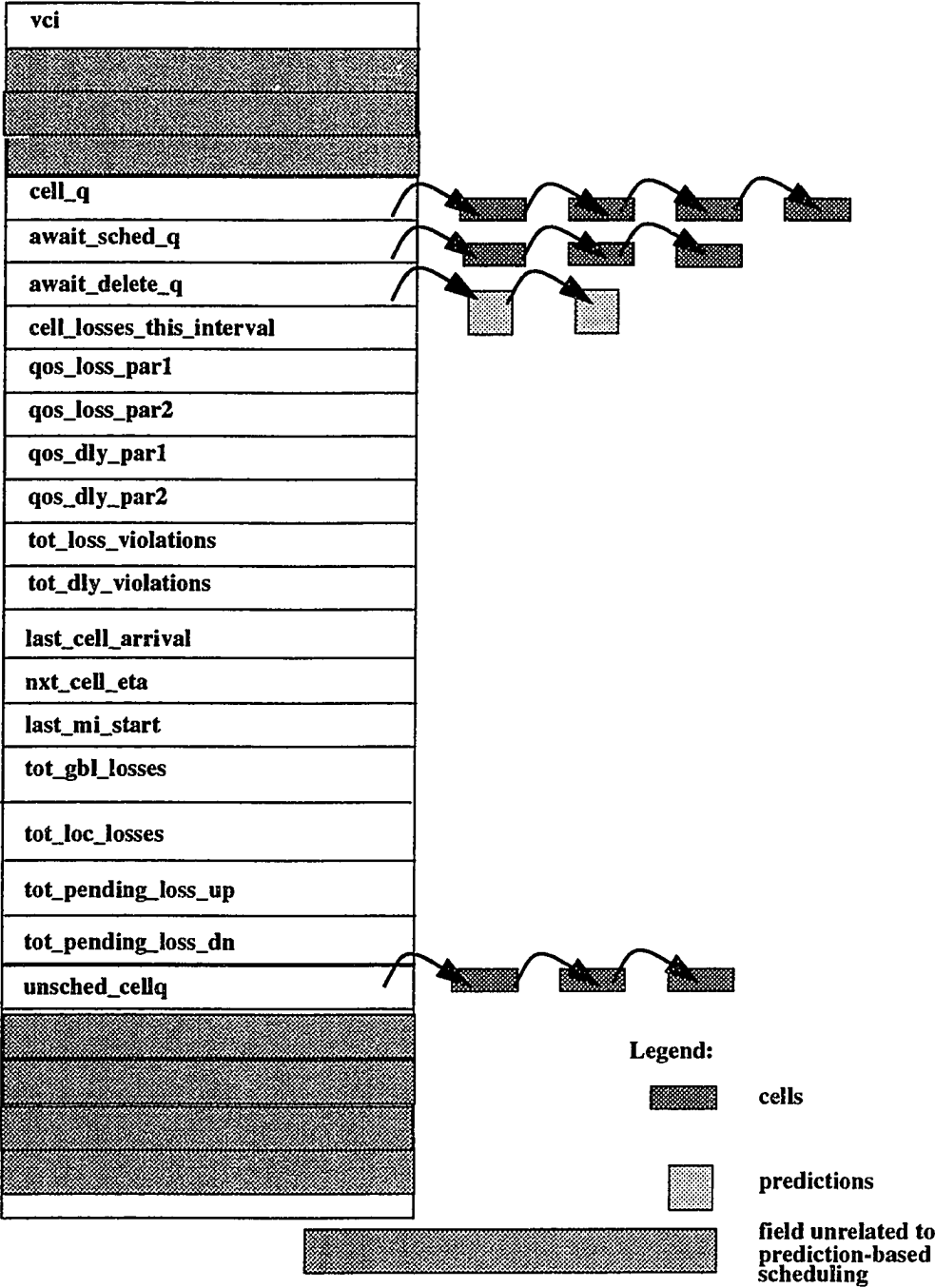
10. *tot\_dly\_violations*. This statistic is part of the QoS maintenance function. (See *qos\_dly\_par1* for details.)
11. *last\_cell\_arrival*. This records the arrival time of the most recent arrival on this connection. It is used to calculate cell interarrival time.
12. *next\_cell\_eta*. This represents the estimated arrival time of the next cell on this connection. It is derived from *last\_cell\_arrival* and *qos\_dly\_par2*.
13. *last\_mi\_start*. This statistic is part of the QoS maintenance function. (See *cell\_losses\_this\_interval* for details.)
14. *tot\_gbl\_losses*. This statistics counts all cell drops for this connection, including those losses obtained from upstream and downstream neighbors.
15. *tot\_loc\_losses*. This statistic counts only those cell drops performed locally.
16. *tot\_pending\_loss\_up*. This statistic is used to maintain Global Knowledge of QoS State. This represents the number of local and downstream-learned cell drops that have yet to be propagated upstream.
17. *tot\_pending\_loss\_dn*. This statistic is used to maintain Global Knowledge of QoS State. This represents the number of local and upstream-learned cell drops that have yet to be propagated downstream.

The following VciCTL block components are used only by *AALScheduler* and have no relevance for the switch:

1. *unsched\_cellq*. This queue is used to schedule cells that may be delayed arbitrarily long in AAL buffers. The principle applied here is that if a burst of ABR cells arrives at the transmitter network boundary we do not want to schedule them immediately since this would occupy prediction slots that may be better used by yet unknown VBR arrivals. (CBR arrivals could theoretically be accurately forecast, but our simulator currently does not distinguish between VBR and CBR classes in generating predictions). Thus, rather than occupy prediction slots prematurely, ABR cells are only scheduled by the AAL when a “sparse” prediction cell is about to be sent. Any unused time slots in such a prediction may be used for ABR cells waiting on VciCTL *unsched\_cellqs*. The simulations conducted in this research serve the many possible VCI *unsched\_cellqs* in a fixed priority order.



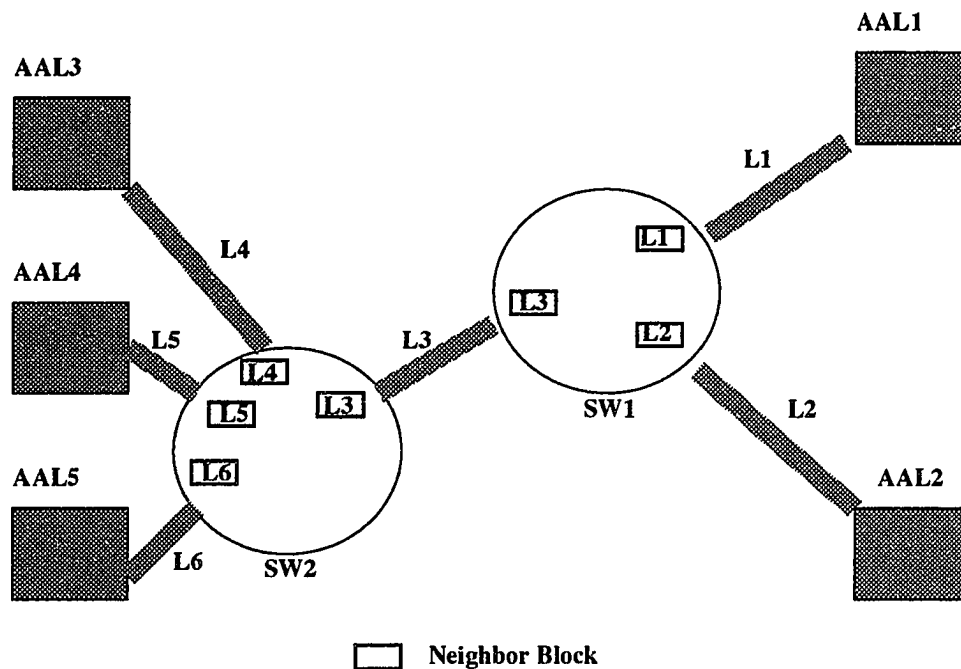
**FIGURE 18. The VciCTL Block**



## 6.5.2 Output Link Control Blocks (Neighbor Blocks)

Just as some kind of VCI control structure is a likely part of any ATM switch database, the switch architecture is likely to include some control structure for each communications link to a neighbor. A generic switch would use this structure to encode such link-specific characteristics as bit rate, neighbor switch identification, etc. Subsequent references to this structure use the term *NeighborBlock*. Figure 19 depicts how these structures would map to the links connecting SW1 and SW2 to their neighbors.

FIGURE 19. Neighbor Blocks in the Network



### 6.5.2.1 Schedules

The prediction-based cell scheduling switch augments *NeighborBlock* to include information about the predictions and schedules specific to the link described by that structure. In our implementation of the prediction-based cell scheduler, a schedule for a cycle is implemented as an array of VCI identifiers. We depict such a schedule in Figure 20. The index of each element in the array determines a specific cell slot time within the cycle being scheduled. Each *NeighborBlock* (Figure 21) contains a collection of these schedules ranging from the schedule for the time cycle currently being transmitted up to *PredictHorizon* + *SkeletonDepth* cycles into the future. Figures 22 and 23 show that we have chosen to implement this collection of cycles as a ring of schedules indexed by cycle number. In par-

particular, the difference between Figures 22 and 23 illustrates how the portion of that ring that represents fixed schedules advances as the current time cycle advances from  $T_i$  to  $T_{i+1}$ . The portions of the ring that are *not* fixed are comprised of those schedules that are still candidates for scheduling arrival predictions.

**FIGURE 20. A Schedule for a Cycle**

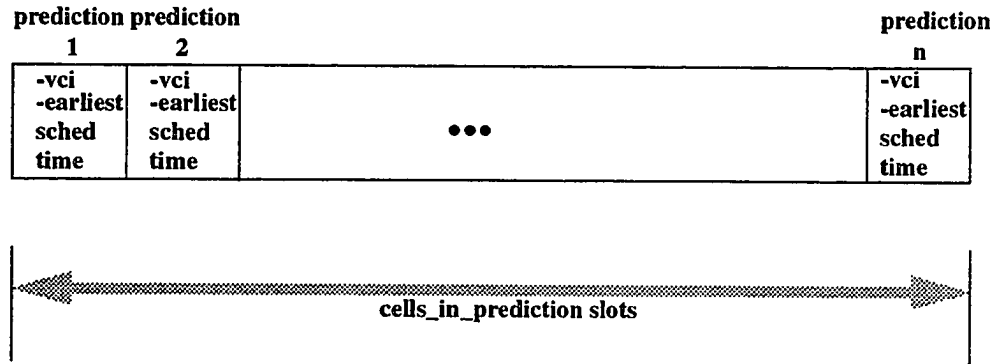


Figure 24, "Interrelationship of Structures and Scheduling Components (perspective: link  $xz$ )," depicts the relationship between a Neighbor Block for a link  $xy$ , its schedule skeletons  $\kappa$ , its committed schedule  $S$ ,  $LinkScheduler_{xy}$  and  $PShunt_{xy}$ .

FIGURE 21. Neighbor Blocks Detailed

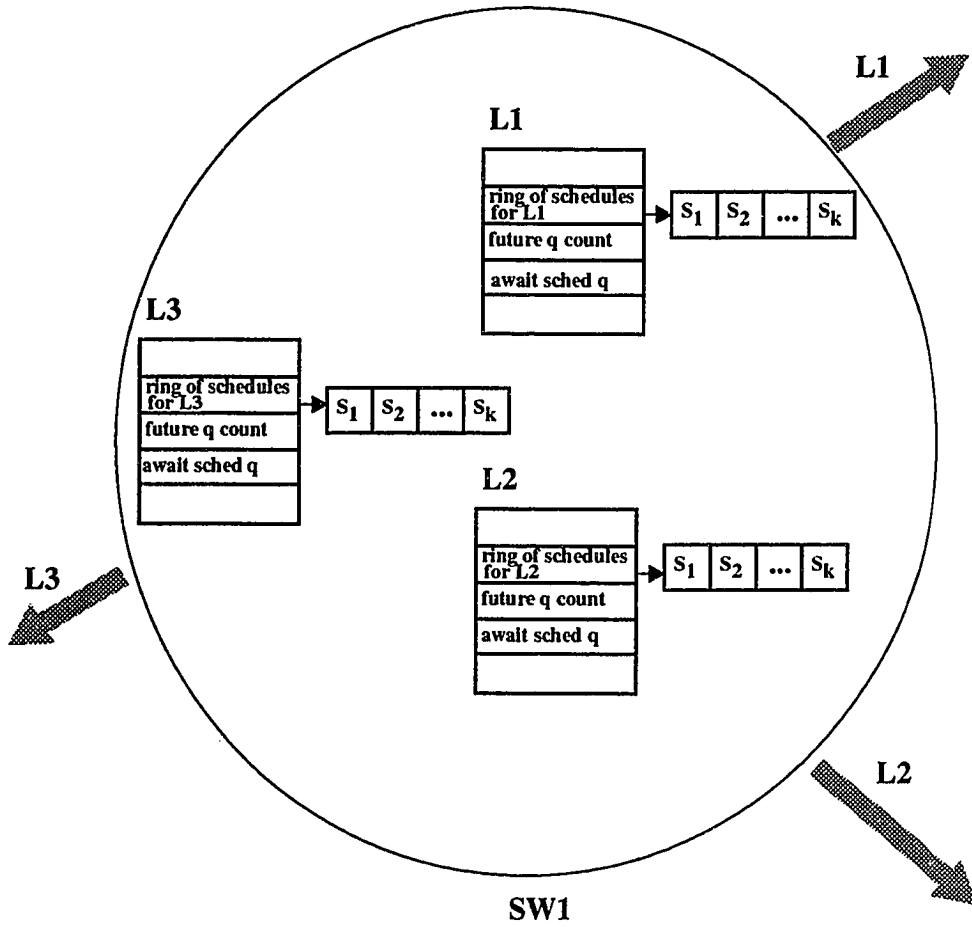


FIGURE 22. Ring of Schedules at Time  $i$  for Link  $K$  in Node  $Z$ .

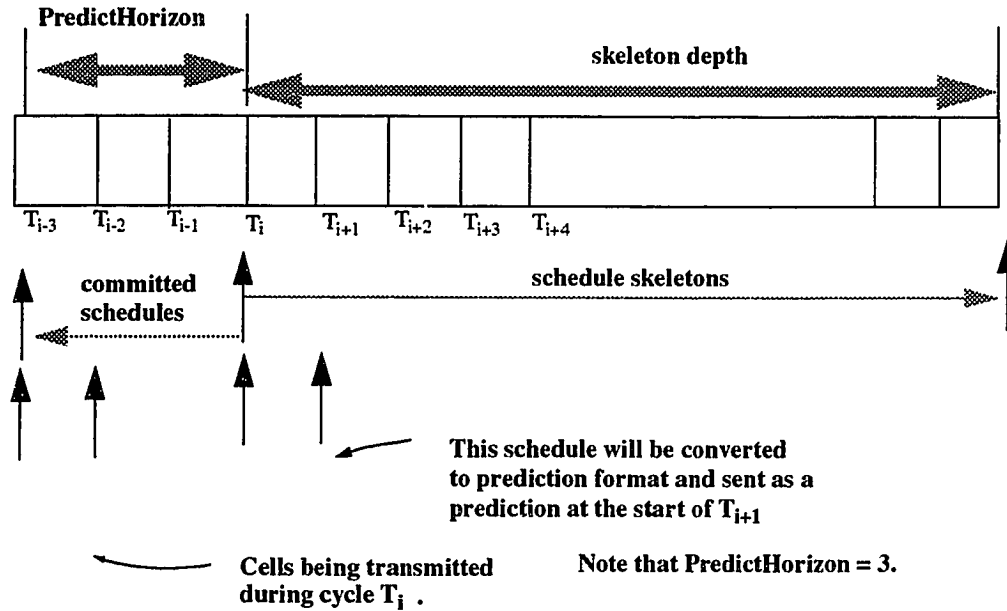
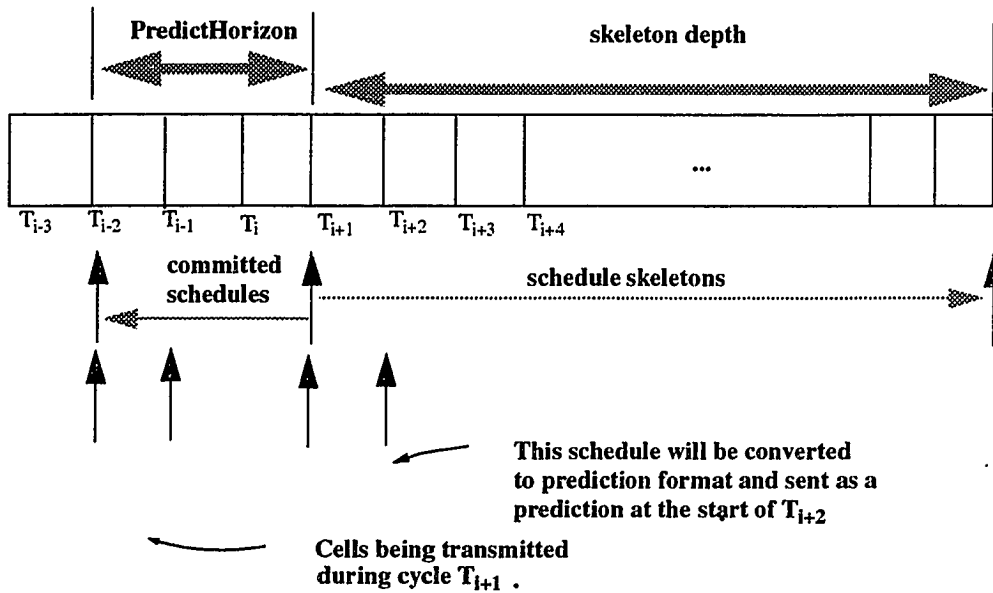


FIGURE 23. Ring of Schedules at Time  $i+1$  for Link  $K$  at Node  $Z$ .



### 6.5.2.2 Future Queue Count

The *NeighborBlock* is also expanded to include a future queue count. When the prediction-based scheduler decides to drop a cell, it is actually making a decision to drop a *future* arrival. The decision to drop the cell is usually motivated by having exhausted the buffer space available for queuing cells. *We assume that this buffer space is allocated on a per-link basis and that no sharing of buffer space between output links is possible.* When *Link-Scheduler* determines whether or not there is room to buffer an arriving cell, that decision must be based on a future queue count, not on the current number of queued cells. The maintenance of this *NeighborBlock*-based future queue count requires emulation of the arrivals and service for this future queue and is performed as part of scheduling the future cell arrivals described by arriving predictions. This emulation can be performed at low computational cost by embedding it directly into the processing of the set of prediction cells received for each cycle.

### 6.5.2.3 Cell Deletion

When faced with the need to drop a cell, the challenging problem of determining the optimal cell to drop is one of the areas explored in this research. We present specific ideas in this area in Section 6.4. In this section we discuss the implementation challenges faced by our prediction-based system when confronted with the need to drop a cell. We commence with a brief review of why it may be necessary to drop a cell in the first place.

#### Why drop a cell at all?

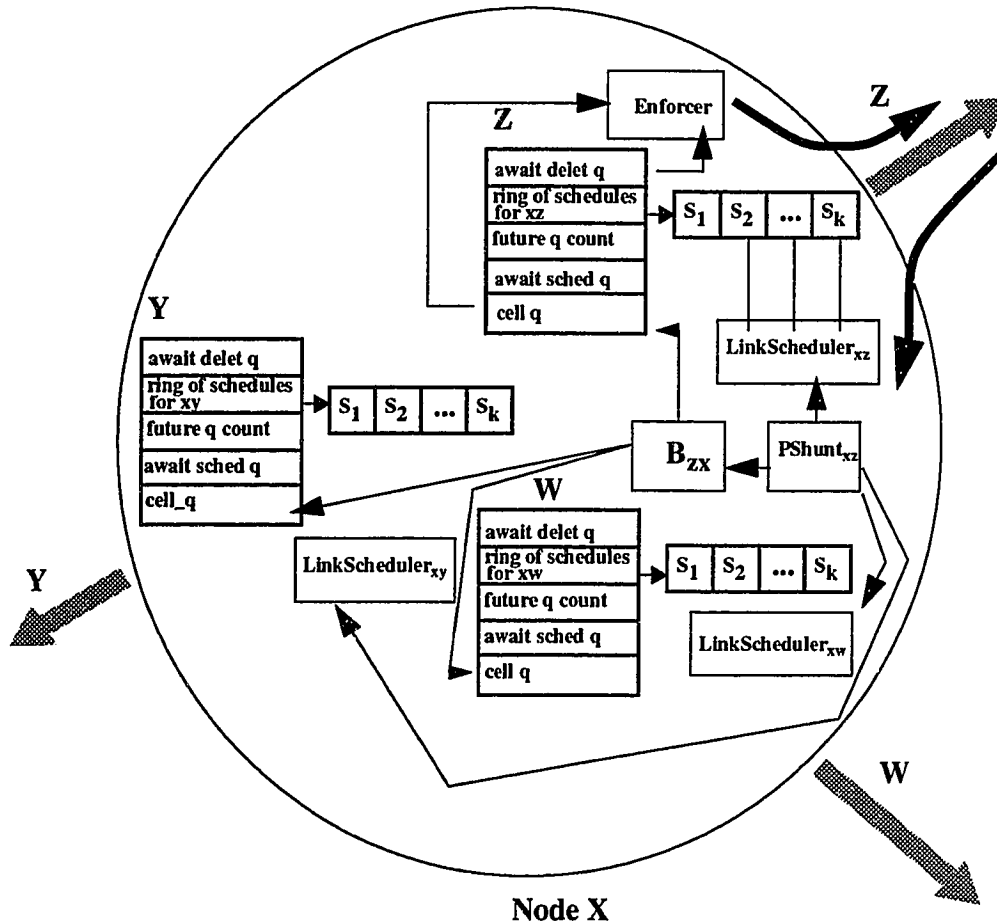
If a jitter-sensitive cell would violate its QoS delay variation parameter or force another VCI to violate its QoS contract, that cell should be dropped. If any cell arrives and there is (or will be) insufficient buffer space to hold the cell, either that cell or another “less costly” cell should be dropped. Note that a CBR cell should never need to be dropped due to lack of buffer space if the call admission function works properly.

#### If a cell must be dropped, what is involved?

Our prediction-based scheduling paradigm is complicated by the fact that the decision to drop the cell takes place when the prediction for that cell arrives, and, hence, before the actual cell arrival. In order to preserve the deletion request, we implement the *await\_delete\_q* (see Figure 18). When a predicted cell is to be deleted, the prediction of that cell is queued to the *await\_delete\_q*. When each cell arrives at the switch, the corresponding *VciCTL* is inspected by *Enforcer* to see if there is a pending deletion request that can

match that cell. As long as the actual arrival time of the cell is greater than the earliest arrival time in the prediction at the head of the *await\_delete\_q*, the cell is dropped and the deletion request is removed.

FIGURE 24. Interrelationship of Structures and Scheduling Components (perspective: link xz)



### 6.5.3 Prediction Cells

Figure 25 describes the format of a prediction cell in our implementation. A prediction cell contains the same number of cell descriptors as the schedules shown earlier in Figure 20. As with schedules, the position of the cell descriptor (see Figure 26) within the prediction denotes the specific cell time slot being predicted for that VCI. That cell descriptor, however, differs somewhat from the cell descriptor in a schedule. We emphasize that while the two are similar, *schedules* are data structures internal to a switch and *predictions* are con-

trol cells that flow between neighboring switches. First, the prediction cell is constrained by the ATM cell payload size of 48 octets, so  $CellsInPrediction \times |CellDescriptor|$  must be less than or equal to that payload size. If, for example, we assume that *cells\_in\_prediction* equals 16, then the *CellDescriptor* can occupy no more than 24 bits. As shown in Figure 25, the prediction cell contains both a VCI identifier and global QoS information. We assume the existence of a hash function that reduces the 32 bit VPI-VCI cell identifier to some value less than 24 bits. To the extent that the hash function can reduce the VPI-VCI cell identifier to less than 24 bits, the amount of global QoS information that may be piggybacked with each predicted cell increases. In our implementation, the Global QoS information field reports new cell losses on that VCI that have occurred upstream of the direction of flow of the prediction cell. In order to allow use of an otherwise empty prediction cell to report Global QoS information, we use an *Information-only Flag* (see Figure 26) so that the scheduling logic can distinguish such a prediction cell descriptor from one actually predicting a future arrival on that VC. If the *Information-only Flag* is clear and there is Global QoS information piggybacked with the prediction, then that prediction cell slot is serving the dual purpose of communicating a prediction and Global QoS information. Continuing with the example of the 16-cell prediction, if the hash function encodes the VPI-VCI identifier with 20 bits, the information flag bit would leave us with 3 bits permitting communication of up to 7 cell drops for a single VCI in one descriptor. When more than 7 cell drops need to be communicated at one instant, the *tot\_pending\_loss\_up* (*tot\_pending\_loss\_dn*) shown earlier in Figure 18 can store unreported downstream (upstream) losses until a cell descriptor for that VC can be sent.

We cannot place an upper bound on how long it can take to propagate Global QoS Knowledge throughout a network. If a network is running at 100% capacity, there may be no unused cells available to propagate cell drop information in the reverse direction of the cell flow on a VC. *For this reason, downstream-globality may be easier to achieve in practice than upstream-globality. (See Section 6.3.2.)* We believe that under realistic network loading levels, this implementation of Global QoS knowledge will serve to improve end-to-end QoS guarantees. The results shown in Section 7.7, "CBR Loss Guarantees Only (Global QoS Knowledge)" support this thesis.



FIGURE 25. A Prediction Cell

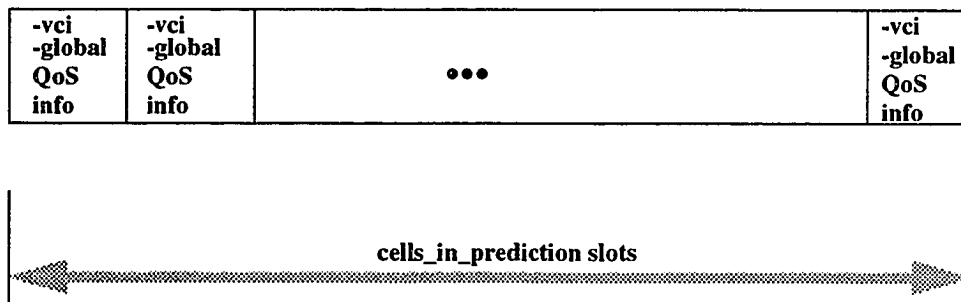
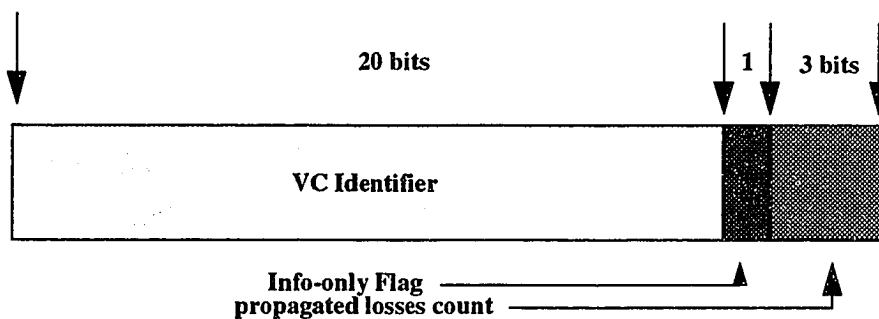


FIGURE 26. A 3-octet Prediction Cell Descriptor



## 6.6 Distance from Quality of Service Violation Metrics

In this section we propose four different criteria that could either stand alone or be used in conjunction with one another to form *DistanceFromQoSViolation* metrics. Each of these can be computed with at most a few arithmetic operations. The results published in this thesis that are derived from the application of a *DistanceFromQoSViolation* metric are obtained using *Loss Cushion Depth* only.

### 6.6.1 Loss Cushion Depth

A simple *DistanceFromQoSViolation* value for any VCI may be computed as the difference:

### *QoS LossPar1 - Cell Losses This Interval*

This is perhaps the most direct measure of *DistanceFromQoS Violation* in that it directly reflects how many additional losses may be tolerated in the current interval before a loss violation occurs.

#### **6.6.2 Time Since Last Cell Discard**

A straightforward extension of this simple metric is to compute the weighted quotient of the number of time units since the last cell was dropped times the loss cushion depth described above. That is,

$$((QoS LossPar1 - Cell Losses This Interval) \times K) / (Time Since Last Cell Loss)$$

If the constant  $K$  is chosen properly, this metric might provide early detection of the start of a burst of cell loss by revealing that the losses are occurring in a comparatively shorter time interval compared to another connection.

#### **6.6.3 Depth of Network Penetration**

Another extension to the metric described in Section 6.6.1 is to weight the *DistanceFromQoS Violation* metric according to the cells' penetration into the network. This idea stems from the intuition that it is more costly in terms of network bandwidth loss to drop a cell that has already traversed 10 network hops than to drop a cell that is comparatively close to the transmitter network boundary. Preliminary results on the benefits of this metric in a traditional (non-prediction) ATM switch environment are available in [7].

#### **6.6.4 Multi-cell Packet Preservation**

When user traffic is presented to the transmitter traffic boundary, it is usually presented as packets or frames of data that are much larger than the ATM cell. The ATM segmentation and reassembly function (SAR) segments these user frames into ATM cells and reassembles these cells into frames at the receiver traffic boundary. Generally, if one cell of the packet or frame is dropped, the entire frame is discarded in the reassembly function. In the ATM switches, these frames appear as multi-cell bursts. Thus, once *LinkScheduler* drops one cell from such a burst, all remaining cells of that burst are doomed and may be dropped without incurring additional QoS violations. Implementation of this strategy requires that *LinkScheduler* violate the convention that the SAR header only be

inspected at the network edge. Romanow and Floyd study the application of this metric in a standard ATM switch environment in [8].

## 6.7 References

- [1] C. Fang, H. Chen, and J. Hutchins. Simulation analysis of TCP performance in congested ATM LAN using DEC's flow control scheme and two selective cell-drop schemes. ATM Forum 94-0119, January 1994.
- [2] A. Fichou and P. Foriel. Spacing/policing function on a VP/VC basis. ATM Forum 94-1075, January 1994.
- [3] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.
- [4] P. Guillemin, P. Boyer, A. Dupuis, and L. Romoeuf. Peak rate enforcement in ATM networks. In *IEEE INFOCOMM 92*, volume 2, pages 753–758, June 1992.
- [5] Udi Manber. *Introduction to Algorithms*. Addison-Wesley, 1989.
- [6] Keung S., H. Tzeng, K. Siu, C. Fang, H. Chen, and J. Hutchins. IPX performance with flow control in congested ATM LANS. ATM Forum 94-0272, March 1994.
- [7] R. Chauhan and R. Russell. Simulation of ATM networks. Technical Report TR-94-19, University of New Hampshire, December 1994.
- [8] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. In *ACM SIGCOMM 94*, pages 79-88, October 1994.

# Chapter 7

## RESULTS

### 7.1 Introduction

We have theorized that the model presented in Chapter 3 will allow us to use more sophisticated scheduling methods and that such methods will lead to improved QoS performance. In order to validate this theory, we have conducted numerous simulations of a prediction-based ATM network. Note that we have attempted neither analytical work nor implementations of this model. We deem the former infeasible due to a lack of established methodology for representing such a complex network mathematically. The implementation is only infeasible due to time and financial constraints.

We obtained simulation results varying the following parameters:

1. *network topology*. Specifically, our simulations varied the number of switches and links, and the length of the communications links.
2. *switch class*. We uniformly simulated both small-buffered switches and large-buffered switches. We did not simulate input-buffered switches as our model was restricted to output buffered switches.
3. *traffic characteristics*. We simulated both CBR and VBR traffic with different cell generation rates. We simulated combinations of different numbers of traffic sources and sinks with different traffic patterns between the sources and sinks. The traffic exhibited varied QoS tolerances for loss and jitter.

4. *scheduling policy*. Our simulations were conducted for the scheduling policies listed in Table 2, “Scheduler Summary,” on page 102. This included two non-prediction-based schedulers used for baseline comparisons.

The most impressive simulation results are revealed in the traffic scenarios where there is a demand for relatively high network utilization *and* there are mixed QoS requirements. By *mixed QoS requirements* we mean that the loss-tolerance and jitter-tolerance as well as the *degrees* of those tolerances vary between the connections in the simulation. The simulations demonstrate that the application of a scheduling policy (*PTDM\_D*) that combines both displacement and cell-spacing can indeed address both loss and delay, and can yield overall QoS performance significantly better than the generic schedulers. This performance improvement was most striking with respect to jitter.

We begin this chapter in Section 7.2 with a brief discussion of the simulator used to produce the results and provide an overview of the configurations used for the simulations in Section 7.3. Section 7.4 describes our definition of how QoS performance should be evaluated. The following three Sections 7.5, 7.6 and 7.7 provide results focussing on very specific QoS performance characteristics of our heuristics to illustrate how the fundamental scheduling trait of QoS-sensitive cell displacement improves loss performance while pseudo-TDM scheduling enhances jitter performance. Section 7.8 illustrates how the *PTDM\_D* scheduler which combines these two fundamental traits, can produce results that are superior overall to all other scheduling methods studied here. In Section 7.9 we extend the simulations to measure QoS performance of both CBR and VBR connections. The connections studied in these extended results are characterized by more diverse QoS requirements than those used in the earlier tests. We then examine how these schedulers fare when evaluated by the more traditional network metrics of link utilization, throughput and delay in Section 7.10, 7.11, and 7.12, respectively. We conclude the chapter with an analysis of the observed performance of all the schedulers studied.

## 7.2 Simulator

Our simulation tool is based on the MIT network simulator (Netsim), and includes ATM enhancements implemented by Sandia National Laboratory and independent work conducted at the University of New Hampshire (UNH). The Sandia additions include an ATM switch module, an AAL module for segmentation and reassembly, and optional ABR credit-based flow control [2] (which was not used in these tests). The UNH additions include implementation of CBR and on-off VBR traffic sources [3], and on-line calcula-

tion of delay distributions [1], thereby eliminating the need for time-consuming post-processing of Netsim log files.

To this basic tool we have added the optional capability of running in *prediction* mode. In prediction mode we actually simulate the operation of the network's active predict components including *AALScheduler*, *LinkScheduler*, *PShunt*, and *Enforcer*, as well as the passive components including *W*,  $\beta$ , *B*, *D*, and *S*. Prediction cells are emitted on all network links in accordance with the descriptions provided in Chapter 6. Results labeled *PTDM*, *PTDM\_D*, *PFIFO*, *PFIFO\_D*, and *PFIFO\_D\_NG* are produced with our simulator running in prediction mode. Results labeled *NORMFIFO* and *HOLDISP* are from the same simulator running in "generic ATM" mode.

The simulations were run on a Sun Sparc IPX processor. Since the execution times of the simulations are quite long, the simulation runs were for 50 milliseconds of simulated time. In all tests the observations were drawn from the 50 milliseconds starting after 10 milliseconds of simulated network operation. The period between 10 and 60 milliseconds was preferred since transient conditions present during network start-up should have stabilized after 10 milliseconds of operation.

## 7.2.1 Traffic Source Types Supported

Our simulator has been developed with the capability to simulate CBR, VBR (both jitter-sensitive and jitter-tolerant), ABR, and UBR traffic classes. We provide details below on the simulated CBR and VBR sources. As no results for ABR or UBR traffic classes are provided here, we do not describe these further.

### 7.2.1.1 Constant Bit Rate (CBR)

The Constant Bit Rate (CBR) source is a cell-based source that emits cells to a Host AAL function at a constant rate. The source specifies loss tolerance in terms of a measurement interval and the number of cell losses that may be tolerated within that measurement interval before a loss is considered a QoS violation. Similarly, the source specifies jitter tolerance in terms of a nominal cell interarrival time and a variance from that nominal time that may be tolerated before the variance constitutes a QoS jitter violation. The loss tolerance used in our tests ranged from 0 to 100 losses per measurement interval. For jitter-sensitive traffic, the jitter tolerance used in our tests ranged from 2.11 cell times (6000 ns) to 11.3 cell times (32000 ns). Measurement intervals ranged from 1766 cell times (5000  $\mu$ s) to 8830 cell times (25000  $\mu$ s).

### 7.2.1.2 Variable Bit Rate (VBR)

The Variable Bit Rate (VBR) source is a cell-based source that emits cells at a rate that may vary according to a number of parameters. First the source is defined to have a busy period and a silent period. The length of each of these periods is defined to have a mean value specified by an independent parameter. The actual lengths of the interval may be configured as constant, equal to the specified mean, or exponentially distributed around that mean. The length of the silent period may be ignored, leaving the source in a permanently busy state. While in this busy state, cells are emitted at a configured average rate. The interarrival time between cells can be configured to be constant or exponentially distributed. (Selection of the latter turns the VBR source into a Poisson source.) Loss tolerances for the VBR traffic used in our tests ranged over values similar to those specified above for CBR.

### 7.2.2 Link Types

All communications links in our simulations are OC-3 SONET [4] links. The effective bandwidth available for cell transmission is 149.760 Megabits/second. We have assumed a propagation delay of 5085 ns/km, which is typical of optical fiber. Link lengths vary in our tests. For further details on the communication links used in these simulations, see Appendix A.

### 7.2.3 Switch Characteristics

The following switch characteristics apply to all the tests conducted here:

1. No switch fabric delay.
2. There are two different switch buffer sizes used in these simulations. The small buffer switches can buffer up to 90 cells per output link. The large buffer switches may buffer up to 9999 cells per output link. For the source parameters used in these tests, queue depths never approach 9999. Thus, for all practical purposes, the large buffer switches function as infinite buffer switches here.
3. The cell scheduling policy is test-dependent. The policies tested include prediction scheduling policies *PFIFO*, *PFIFO\_D\_NG*, *PFIFO\_D*, *PTDM*, and *PTDM\_D*. The single FIFO default policy is universally tested to provide a basis for comparison. It is called the *NORMFIFO* policy in the results. In the tests in

Section 7.9 we introduce a second generic scheduler, *HOLDISP*, for comparison purposes. This dual priority queue scheduler was introduced in Section 6.4 on page 101.

4. *PredictHorizon* = 3.
5. A basic prediction cycle of 16 cell times is used, which meets the requirement stated in Section 3.10.1 that the link propagation delay be an integral number of basic time cycles.

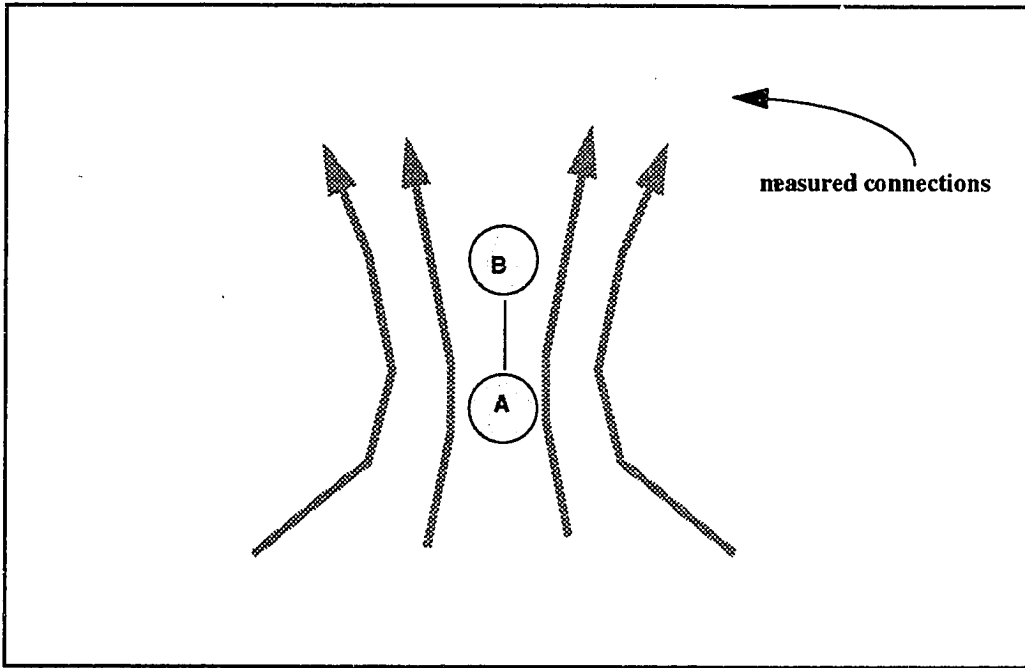
### 7.3 Simulated Network Configurations

The *Bottleneck* traffic flow pattern shown in Figure 27 is the basis for both *Bottleneck\_Jitter* and *Bottleneck\_Loss* configurations. We utilize these configurations in Section 7.5 and Section 7.6 to provide a simple scenario for studying loss or jitter QoS guarantees. These configurations intentionally exclude cross traffic and multiple network hops as part of the simplification. The *Bottleneck\_Jitter* and *Bottleneck\_Loss* configurations are described in detail in Appendix A.

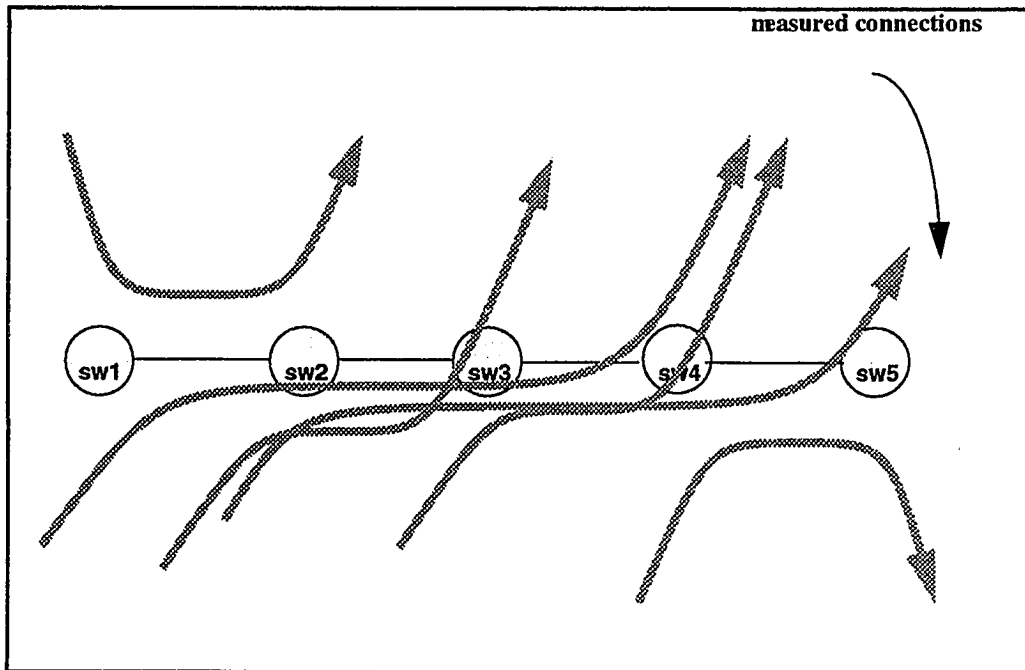
The *Cross\_Traffic* traffic flow pattern depicted in Figure 28 allows cross traffic, multiple hops and varied-length communication links. This topology is the standard GFC2 topology used for simulations by the ATM Forum, and is used here as the basis for the *Cross\_Traffic\_HomLink*, *Cross\_Traffic\_HetLink* and *Cross\_Traffic\_HetTraffic* configurations. These configurations are used in Section 7.7, Section 7.8 and Section 7.9 to evaluate scheduling methods in a more realistic network scenario than that provided by the *Bottleneck* topology. The details of the differences between the *Cross\_Traffic\_HomLink*, *Cross\_Traffic\_HetLink* and *Cross\_Traffic\_HetTraffic* configurations are provided in Appendix A.



**FIGURE 27. Traffic Flow Patterns for *Bottleneck***



**FIGURE 28. Traffic Flow Patterns for *Cross\_Traffic***



## 7.4 Measuring Quality of Service Performance

Since the focus of this research is end-to-end QoS performance, unless otherwise stated, we only report loss results from the point of view of the receiver AAL function. We generally report loss statistics expressed as loss violations. Loss violations are distinguished from straightforward cell loss in that sources in our study can express their QoS tolerance to cell loss as *a number of cell losses tolerable in a specified amount of time*. Only losses greater than the specified tolerances are considered loss violations.

We provide a somewhat more diverse set of measures of jitter performance. We show “raw” jitter in cell delay distributions measured at a variety of links through the network topologies. We also provide distributions of jitter as perceived by the receiver AAL function (the receiver network boundary). Jitter is measured as the difference between a cell’s expected arrival time and its actual arrival time. The expected arrival time of a cell is established from the previous cell’s arrival time plus a connection-specific nominal inter-arrival time. Like cell loss, though, the most telling measure of QoS jitter performance is a count of end-to-end jitter violations perceived at the receiver AAL function. A jitter violation occurs whenever a cell’s jitter is greater than a connection-specific parameter (*qos\_delay\_par2*). In order to contemplate the possibility of jitter-sensitive VBR traffic, if the cell’s jitter is greater than 5 times the connection’s nominal interarrival time, this excessive variance from the expected arrival time is interpreted as the start of a new VBR burst rather than inter-cell jitter.

Our reporting of end-to-end delay is based on the delay distributions shown in Figures 38 through 43 in this chapter, and 75 through 80 in Appendix B. For example, the difference in the x-axis origin between Figures 38 and 39 underscores the end-to-end delay penalty incurred by using prediction-based cell scheduling. We discuss this delay further in Sections 7.5.2 and 7.12.

### 7.4.1 Comparing the Heuristics: Ground Rules

If we are to compare one scheduling method to another, we need a quantitative means of measuring schedulers. We introduce the following ground rules for comparison of the *LinkScheduler* heuristics.

1. A formal comparison between two schedulers must be made for a well defined set of input. The comparison is defined over a particular time period  $\{T_i, T_{i+k}\}$  of operation. The input description defines traffic present in the network at the start

of the time period and traffic introduced into the network during the time period used for the evaluation. The input describes the QoS guarantees requested by each of the connections defined in the input.

2. We count end-to-end QoS violations only. Since individual connections may request cell loss QoS guarantees, cell jitter guarantees, both loss and jitter guarantees, or neither, the definition of what constitutes the QoS violations for a given comparison is a function of the input. QoS violations are counted as the sum of all the end-to-end QoS violations incurred in the interval  $\{T_i, T_{i+k}\}$  for all of the connections defined in the input.
3. Scheduler S is considered better than Scheduler T if the number of QoS violations produced by S's schedule is less than that produced by T's schedule during the period  $\{T_i, T_{i+k}\}$ . While this definition does not apparently account for differences in throughput, it does indeed consider throughput if all connections require both loss and delay guarantees. Provided that the duration of the test is very long compared to the delay allowed, lower throughput would surface as either loss or delay QoS violations.
4. We have frequently referred to jitter as a measure of deviation from the nominal or *expected time of arrival* (ETA) of a cell. In Appendix A we stipulate a nominal cell spacing for each connection. We derive the ETA for the next arrival by adding the nominal cell spacing to the time of the last arrival.

## 7.5 CBR Jitter Guarantees Only

### 7.5.1 Simulation Description and Motivation

The simulation results reported in this section were performed using the *Bottleneck\_Jitter* configuration. As we explained in Section 7.3, *Bottleneck\_Jitter* is designed to allow evaluation of QoS jitter performance of different scheduling algorithms without considering the complications of cross-traffic and the cumulative effects of multiple network hops.

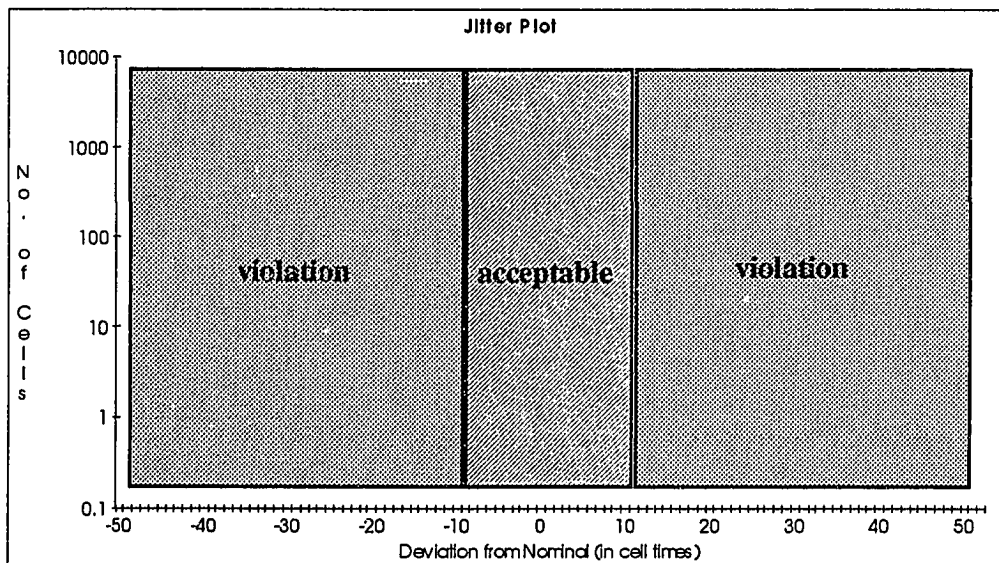
In this test we are interested in observing the effects that a set of bursty sources can have on the jitter characteristics of CBR streams. These tests were conducted for large buffer switches only since we do not wish to induce cell loss in these tests.

We only observe jitter at Host 2a and Host 2b (Figure 73 on page 181), as all the CBR streams terminate in these AAL functions. We measure cell delay distributions both at link *pp3* and the final link before the destination function to observe whether upstream-induced jitter improves, remains unchanged, or worsens as it approaches the destination network boundary.

The tests were conducted for the generic no-prediction scheduler and for the prediction based scheduler using *PTDM*. *PTDM* (Section 6.4.3) attempts to provide jitter guarantees while ignoring any QoS requirements related to cell loss. Considering that the *Bottle-neck\_Jitter* configuration is contrived to avoid cell loss, *PFIFO* policies (Section 6.4.2) and *PTDM\_D* (Section 6.4.4) are not considered in this test.

We varied three parameters in these tests: 1) the number of active CBR connections, 2) whether the total load on the bottleneck increases with the number of active CBR connections or is held constant, and 3) the amount a cell may deviate from its nominal arrival time before that deviation makes that cell useless at the receiver traffic boundary. When a cell becomes useless in this manner we call this a *jitter violation*. The amount of deviation that is tolerable without incurring a jitter violation is called *jitter tolerance* and is expressed here in units of cell times. Figure 29 graphically shows the acceptable arrival interval for a cell from a connection with a 10-cell jitter tolerance. The zero on the X-axis indicates the *nominal* arrival time (i.e. the ETA).

**FIGURE 29. Jitter Tolerance**



Assuming other factors remain constant, increasing jitter tolerance reduces the number of jitter violations.

The first parameter, the number of CBR network connections, controls how many CBR sources of the six configured in *Bottleneck\_Loss* are enabled. Again referring to Figure 73, the sources are enabled in the following order: 1) *gsource1bb*, 2) *gsource1ba*, 3) *gsource1b*, 4) *gsource1ab*, 5) *gsource1aa*, and 6) *gsource1a*. Thus, a 4-connection test would use *gsource1bb*, *gsource1ba*, *gsource1b*, and *gsource1ab*.

We expect that as we increase the total network load (through parameters 1 and 2), we will increase the maximum queue length at the bottleneck link *pp3a*. This directly correlates with an increase in the maximum queue waiting time. We expect that an increase in the maximum waiting time will exacerbate jitter and lead to more jitter violations.

## 7.5.2 Jitter Violations

Figures 30 and 31 show the number of jitter violations for prediction-based *PTDM* and the *NORMFIFO* cell scheduler, respectively. In these graphs the total CBR traffic load is held constant while the number of active CBR connections varies. (Thus, with  $n$  connections, each connection contributes  $1/n^{\text{th}}$  of the total.) Figures 32 and 33 depict results for the case where the CBR traffic load increases linearly with the number of active CBR connections. The expected correlation between jitter tolerance and jitter violations is evidenced in all four figures. The expectation that increasing the number of active connections will exacerbate jitter is also manifested in all four figures, though some interesting results surface, which we discuss below.

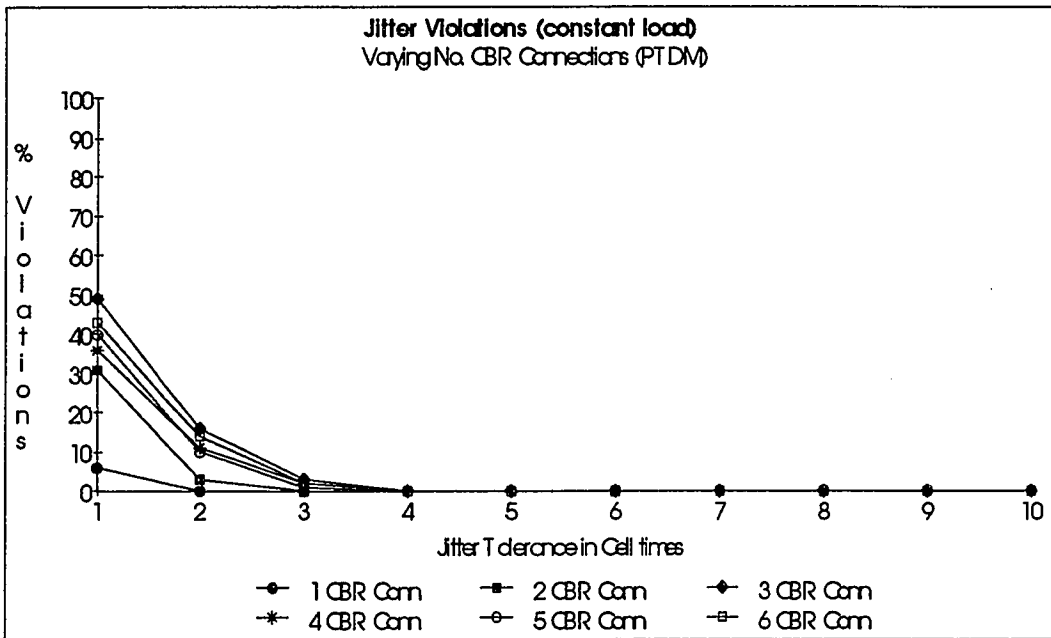
When we compare the number of jitter violations shown in Figures 30 and 32 for *PTDM* to those shown in Figures 31 and 33 for *NORMFIFO*, the performance for the prediction cases is significantly better. In Figures 31 and 33 *NORMFIFO* jitter violations remain at high levels even as jitter tolerance increases. As shown in Figures 30 and 32, the jitter control capabilities of *PTDM* seem relatively consistent whether the offered load is constant or increasing. Such consistency is not observed for *NORMFIFO* where jitter control is considerably worse in Figure 33 (increasing load) than in Figure 31 (constant load). It is significant that in Figure 33 the *NORMFIFO* scheduler shows a pronounced weakness in the area of jitter control even at low traffic load (1-2 connections). These results underscore the fundamental weakness of default scheduling in supporting jitter guarantees, even at low utilization levels.

In Appendix B, Tables 6 and 7 provide the detailed results illustrating this improved jitter performance of *PTDM* scheduling for the constant load case. Tables 8 and 9 provide the detailed results illustrating this improved jitter performance of *PTDM* scheduling for the increasing load case.

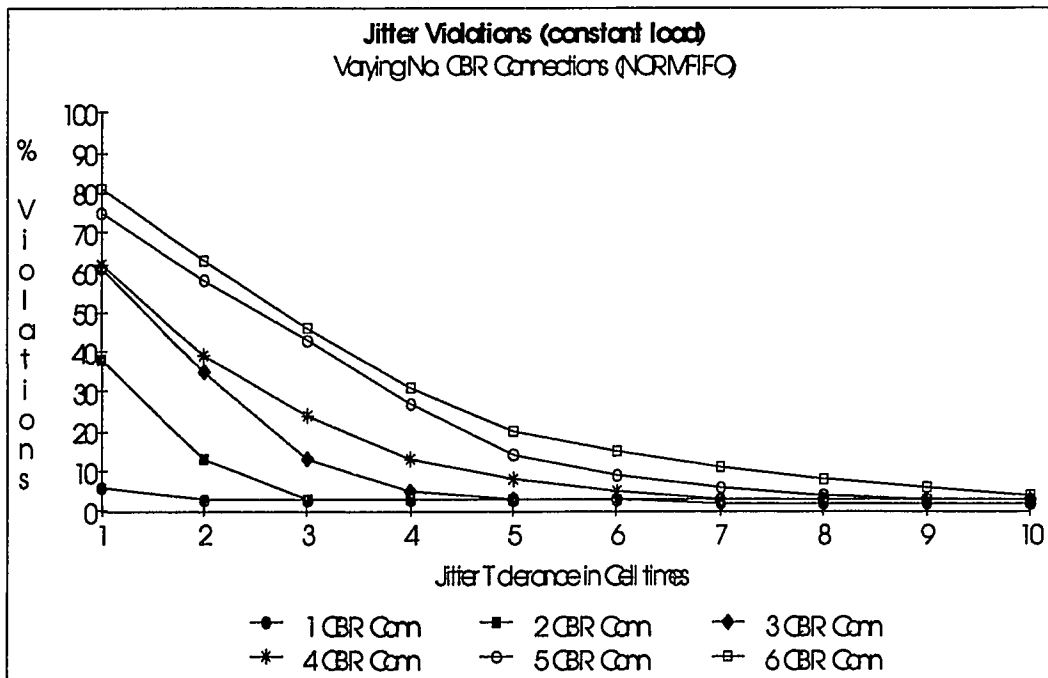
Figures 34 through 37 show the CBR cell delay distribution for a 4-connection load with a 4-cell time jitter tolerance. The VBR loading level was the same as explained in the introduction to this test above. These figures illustrate that *PTDM* provides significantly better jitter performance than the generic FIFO scheduler. Note that there are two network traffic boundaries depicted in these figures, Host 2a (Figures 34 and 35) and Host 2b (Figures 36 and 37). The four active connections are *gsource1ab*, *gsource1b*, *gsource1ba* and *gsource1bb*. Note that this produces an asymmetrical load on Host 2a and Host 2b, with Host 2b receiving the cells of 3 connections and Host 2a of only one. Specifically, of the 4 CBR connections loading the network in this test, three of them are routed to Host 2b (*i.e.* *sink2b*, *sink2ba* and *sink2bb*). This produces three times the load on Host 2b as on Host 2a which only carries a single VC routed to *sink2ab*. This is manifested by the fact that three times the number of cell samples appear in Figures 36 and 37 compared with Figures 34 and 35.

The bimodal characteristic of the *NORMFIFO* scheduler shown in Figures 35 and 37 is explained by the cyclical nature of the VBR bursts. During the VBR active periods in *Bottleneck\_Jitter* there are 6 VBR sources emitting cells at approximately 10 times the rate of the CBR sources. The time between 2 arrivals on the same CBR connection is 12 cell times. During this time there will be around 60 VBR arrivals of which at most 12 will be served. Thus, when the second CBR cell arrives after 12 cell times it must queue behind 48 waiting VBR cells. This second cell will appear to arrive 48 cell-times later than its ETA. Before the third CBR cell arrives, another 60 VBR cells will have arrived, once again increasing the inter-cell spacing between the second and third cell by about 40 cell times. It is the repeated occurrence of this phenomenon during VBR active periods that results in the small modes around +50 cell times in Figures 35 and 37. *PTDM* avoids this by advancing the CBR cells in front of the VBR cells in order to schedule those jitter sensitive CBR cells as closely as possible to their nominal transmission time.

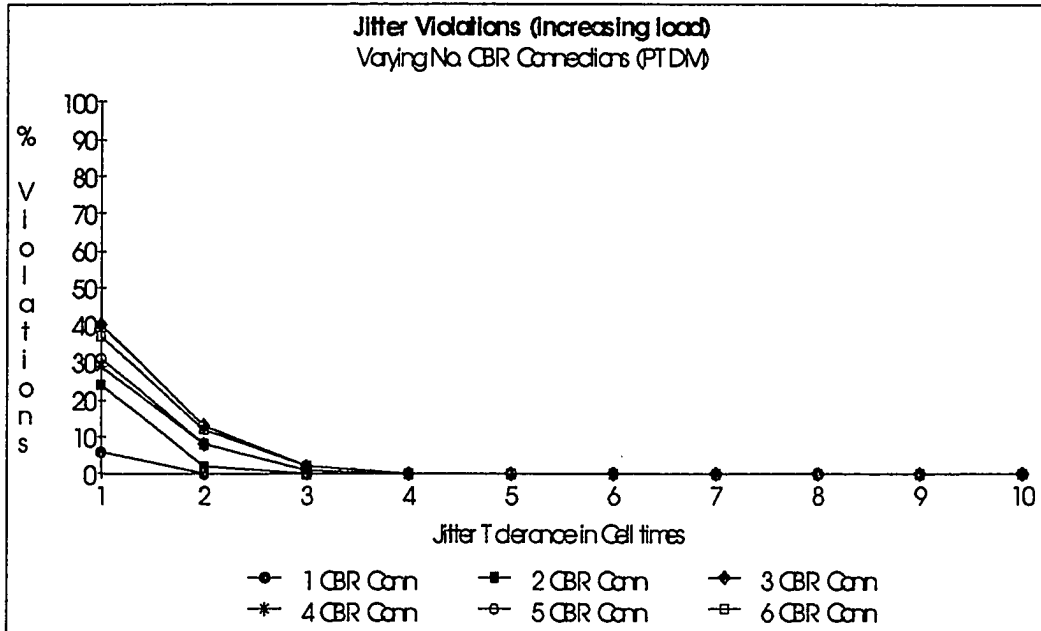
**FIGURE 30. CBR Connections Supported vs. Jitter Tolerance (PTDM, constant load)**



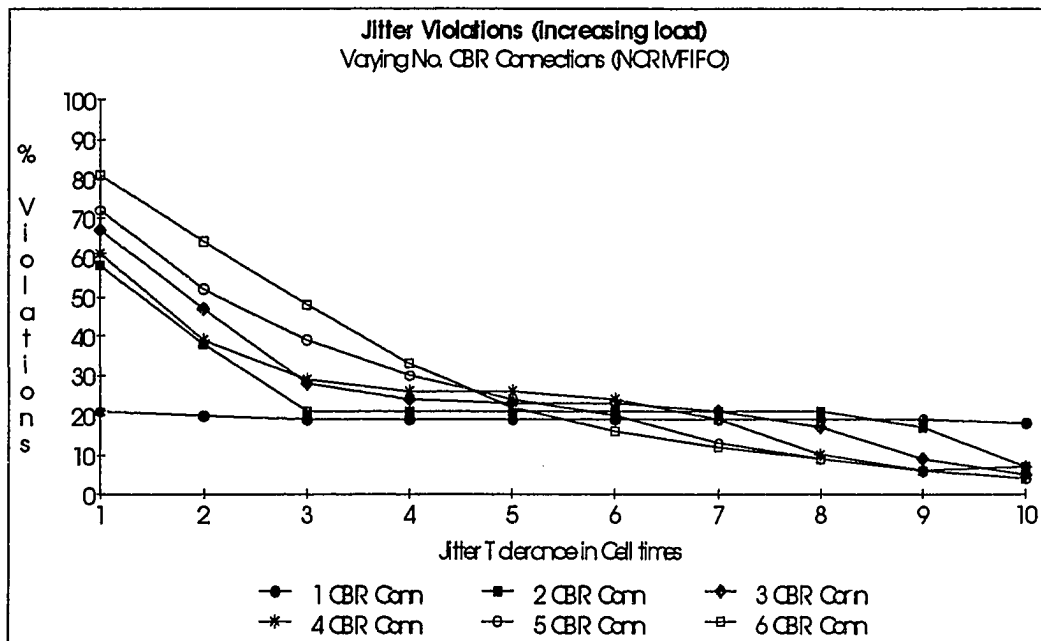
**FIGURE 31. CBR Connections Supported vs. Jitter Tolerance (NORMFIFO, constant load)**



**FIGURE 32. CBR Connections Supported vs. Jitter Tolerance (PTDM, increasing load)**

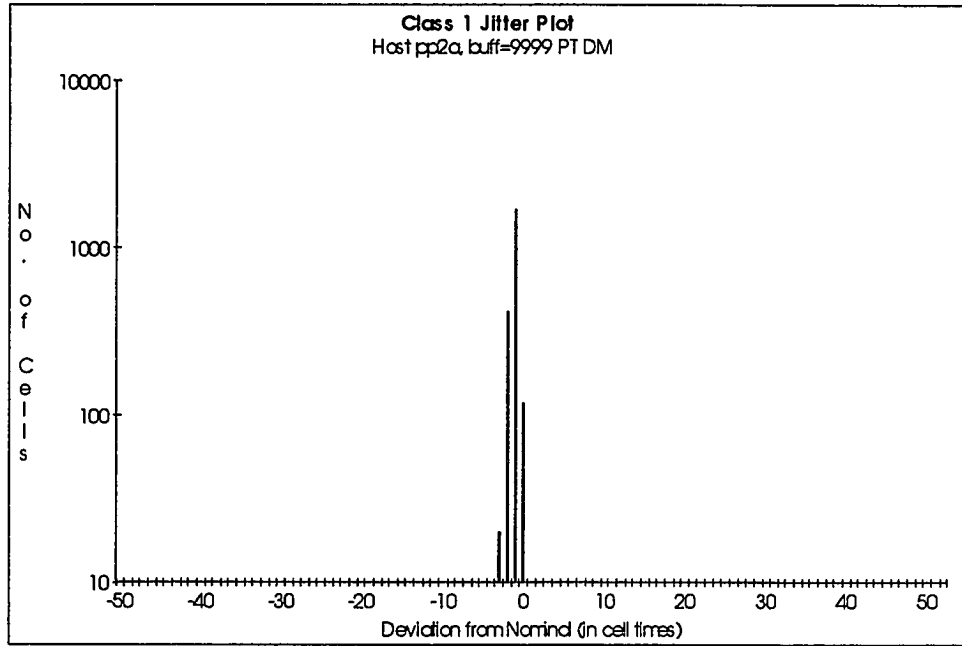


**FIGURE 33. CBR Connections Supported vs. Jitter Tolerance (NORMFIFO, increasing load)**

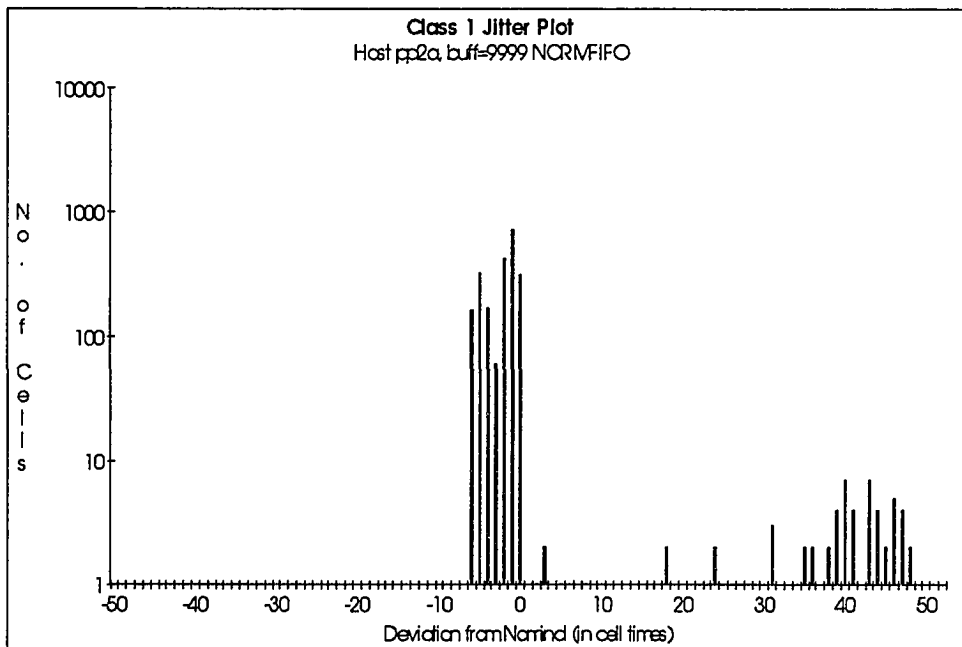




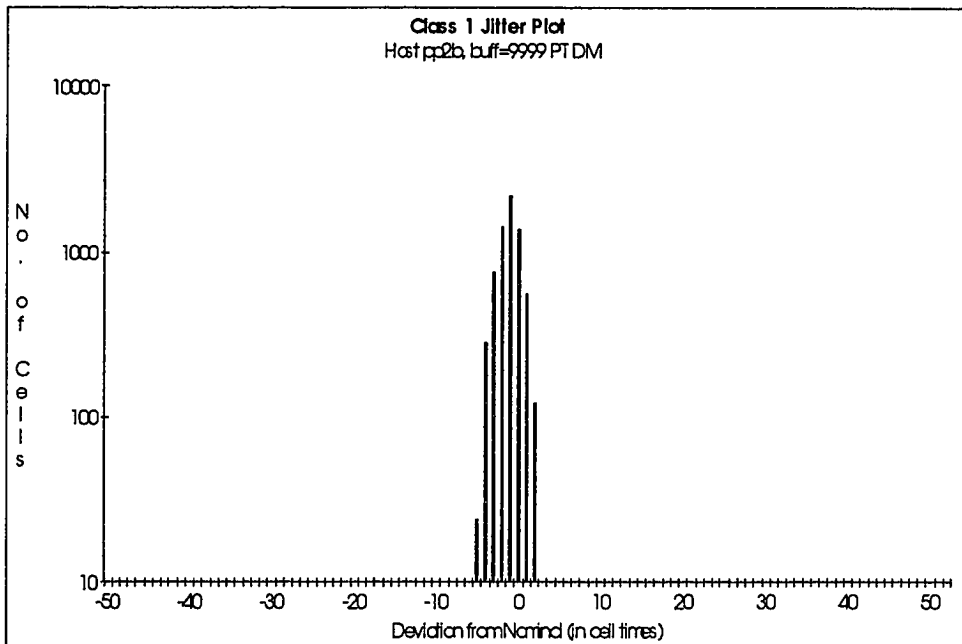
**FIGURE 34. Jitter at Host pp2a (PTDM)**



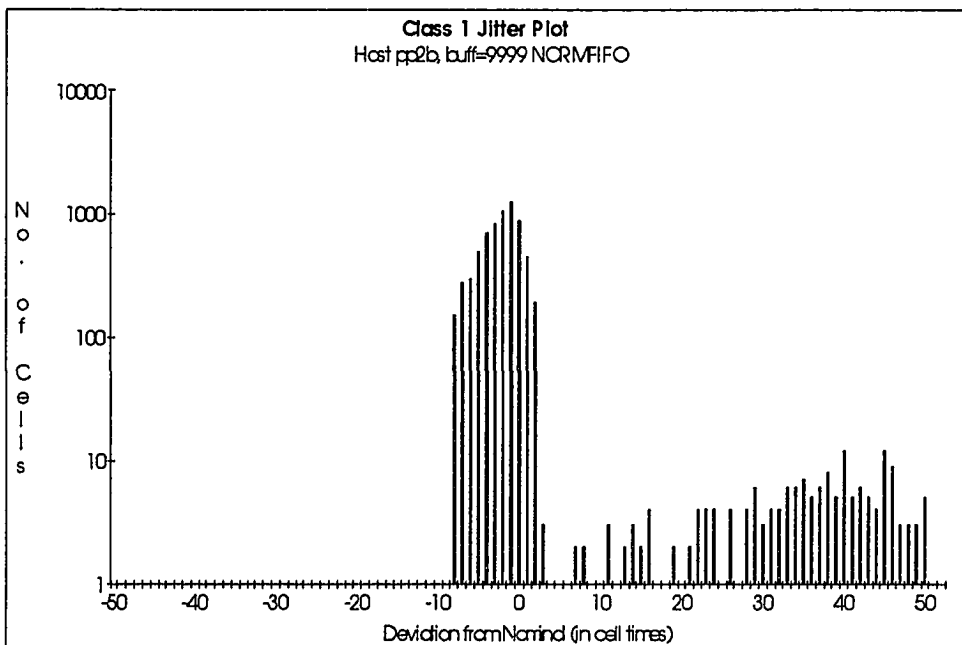
**FIGURE 35. Jitter at Host pp2a (NORMFIFO)**



**FIGURE 36. Jitter at Host pp2b (PTDM)**



**FIGURE 37. Jitter at Host pp2b (NORMFIFO)**

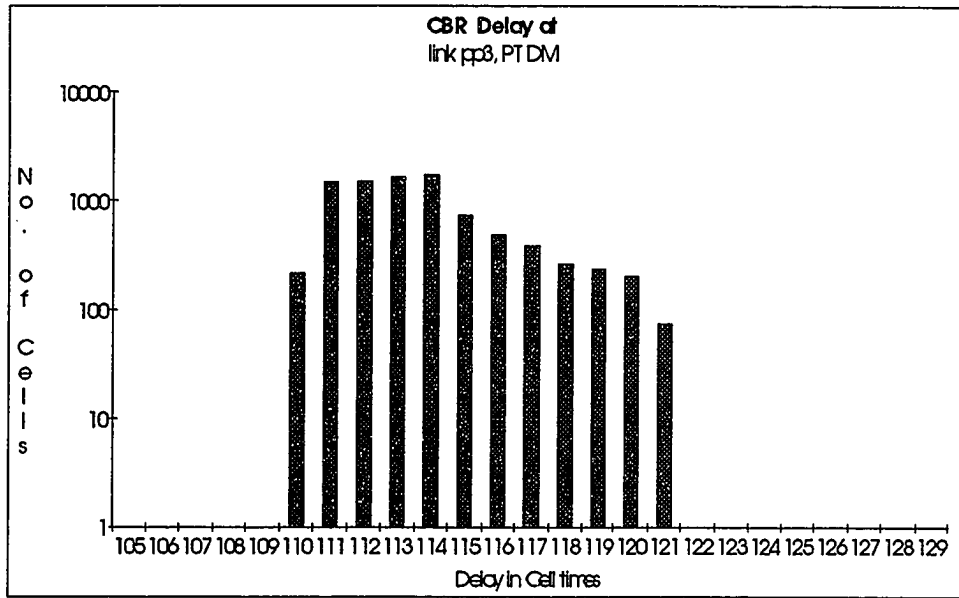


### 7.5.3 Upstream Delay Distributions

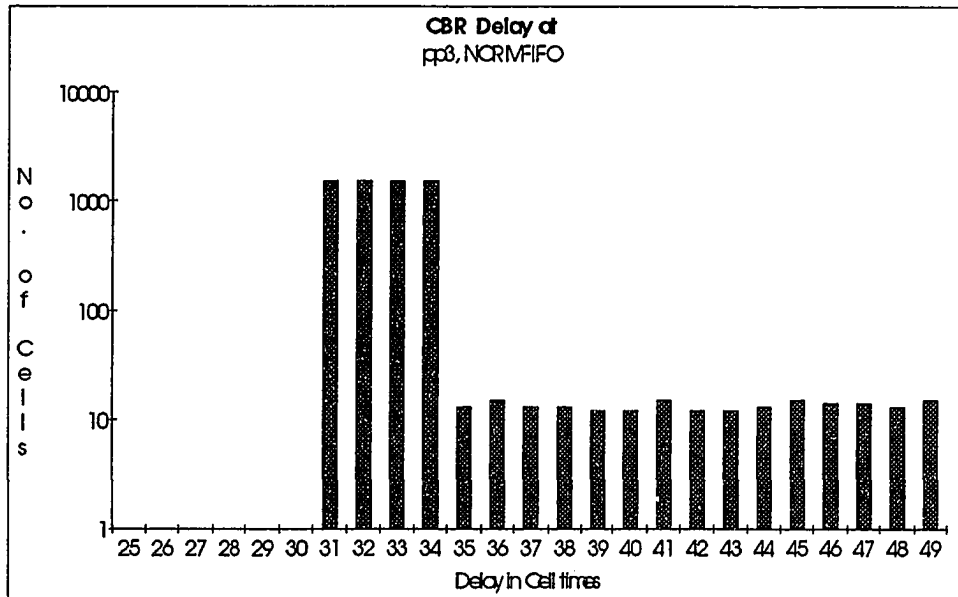
Figures 38 through 43 represent the raw cell delay distributions experienced by CBR cells on the two links upstream of Host *2a* and Host *2b*. These figures plot the time-in-network experienced by each CBR cell as it transits the link described by the figure. These delay distributions are related to the jitter distributions just presented in Figures 34 through 37 as both reflect variance in delay. This is particularly evident when comparing Figure 34 with Figure 40, Figure 35 with Figure 41, Figure 36 with Figure 42, and Figure 37 with Figure 43. The delay introduced by the prediction-based scheduling is evidenced in the much larger minimum delay for *PTDM* as compared to *NORMFIFO*. In Figures 38 and 39 this minimum delay “penalty” at link *pp3* is 79 cells times ( $110-31=79$ ). The penalty increases to 102 cell times in Figures 40 and 41 due to the additional delay introduced by the switch downstream from *pp3*. We discuss this delay penalty further in Section 7.12.

One particularly interesting phenomenon is how the jitter observed upstream at link *pp3* (Figure 38) is actually ameliorated by *PTDM* by the time those cells emerge from *pp2a* and *pp2b* (Figures 40 and 42). This is because *PTDM* is able to judiciously schedule cells in the relatively idle links *pp2a* and *pp2b*. In particular, it inserts delay in order to approach a uniform interarrival time for a connection’s cells. This contrasts sharply with generic FIFO scheduling, which would never take advantage of the freedom to delay a cell and always transmits it at the earliest possible time, even when this induces unnecessary jitter.

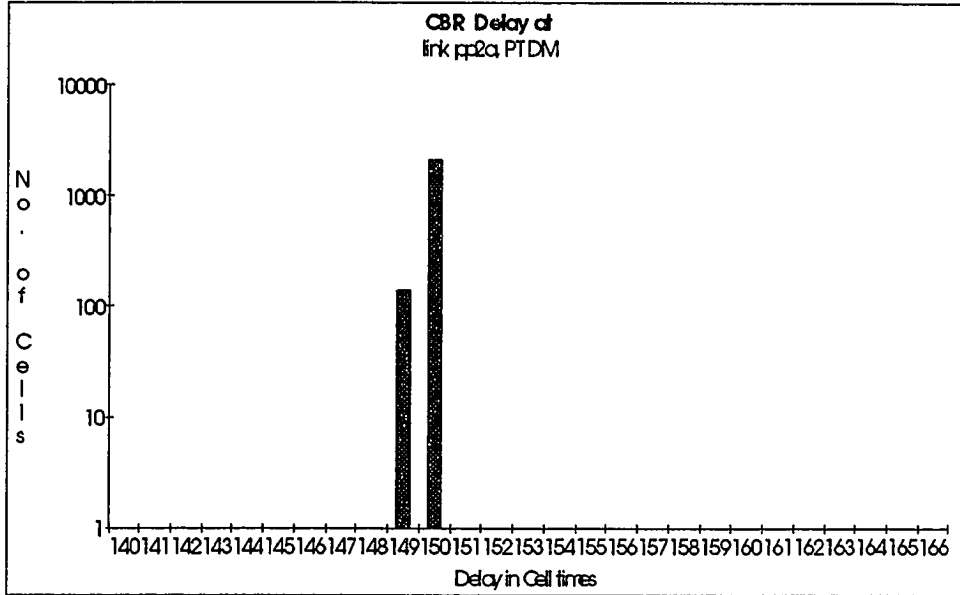
**FIGURE 38. CBR Time-in-Network Distribution at Link PP3 (PTDM)**



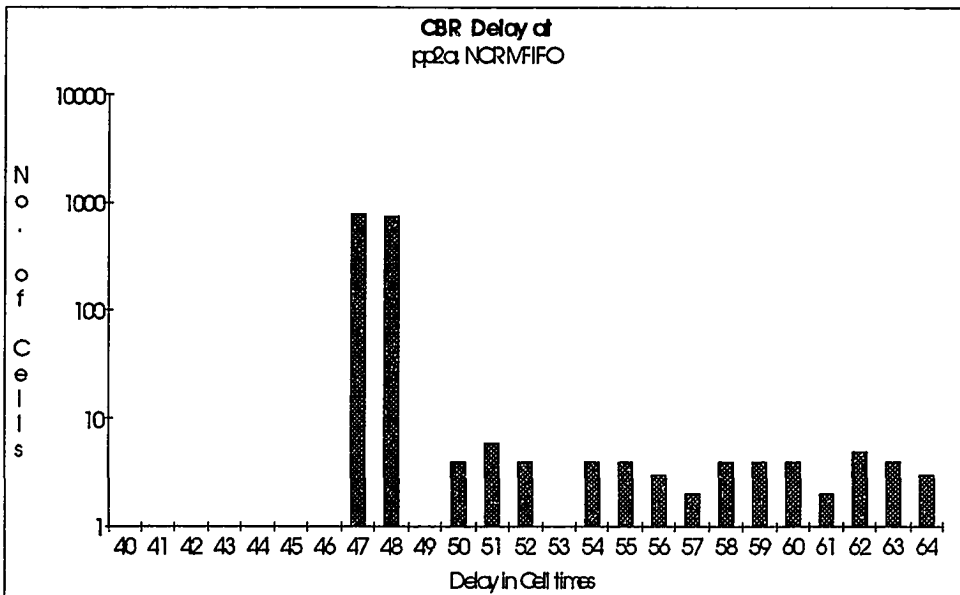
**FIGURE 39. CBR Time-in-Network Distribution at Link PP3 (NORMFIFO)**



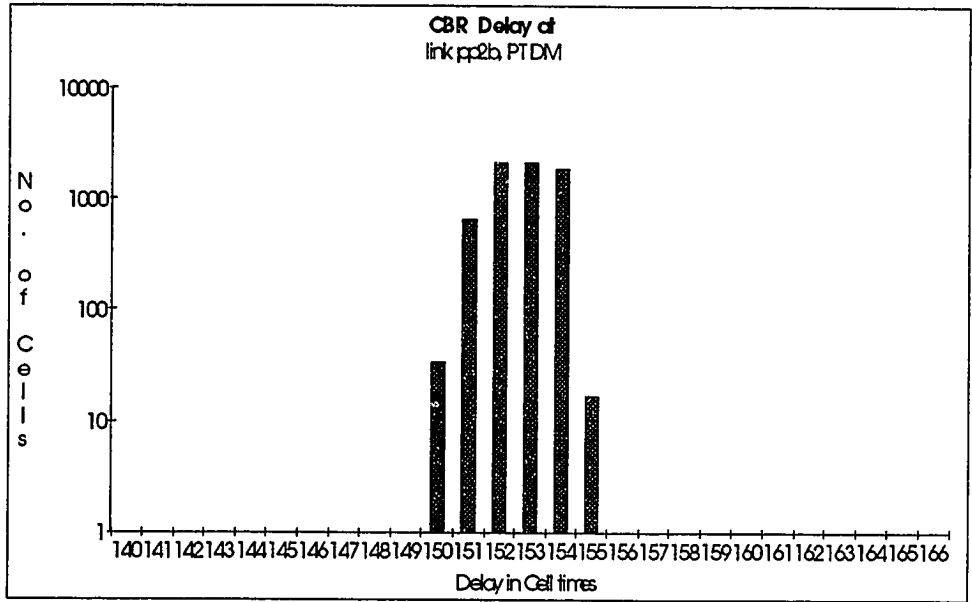
**FIGURE 40. CBR Time-in-Network Distribution at Link PP2a (PTDM)**



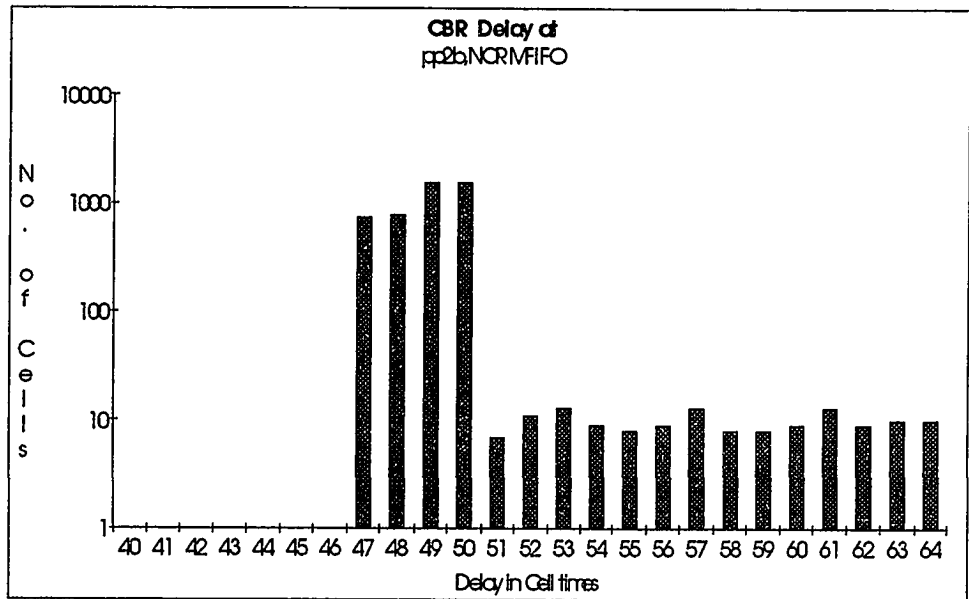
**FIGURE 41. CBR Time-in-Network Distribution at Link PP2a (NORMFIFO)**



**FIGURE 42. CBR Time-in-Network Distribution at Link PP2b (PTDM)**



**FIGURE 43. CBR Time-in-Network Distribution at Link PP2b (NORMFIFO)**



## 7.6 CBR Loss Guarantees Only (Local QoS Knowledge)

### 7.6.1 Simulation Description and Motivation

In this section we provide simulation results based on a simulator implementation of the ideas proposed in Section 6.6.1, "Loss Cushion Depth," and Section 6.4.2, "Predictive FIFO with Displacement (PFIFO\_D)." These simulations are designed to run with a small fixed buffer size (90 cells) and sufficient load to produce traffic loss within the switches. The intent of these tests is to compare what advantages or disadvantages the more sophisticated scheduling strategies exhibit when compared to generic scheduling and to more simple forms of prediction-based scheduling. Like the tests conducted in Section 7.5, we still wish to restrict this comparison to a very simple environment, free of potentially confusing side effects. For this reason, we use the *Bottleneck\_Loss* configuration here as it is our most simple loss-producing scenario. In later sections those policies which appear most useful in these simple tests are evaluated in more complicated test scenarios.

*PFIFO*-style scheduling makes no attempt to provide QoS guarantees about jitter. As such, we only report loss results in this section. We compare two variants of *PFIFO* to the generic scheduler. Baseline *PFIFO* simply discards a predicted arrival if the predicted link buffer is full. The second variant (*PFIFO\_D\_NG*) drops the least-cost candidate cell. (*Candidate cells are those cells that have been predicted but not yet committed and that may be displaced by the new arrival without requiring reordering of other cells.*) This decision involves a calculation of *QoS*Cost for each eligible cell and the computation of a minimum of those costs. If a lesser cost candidate is found, it is *displaced* by the new arrival.

We varied two parameters in these tests: 1) the length of the measurement interval, and 2) the loss tolerance of the user traffic stream during one measurement interval at the receiver traffic boundary. Each missing cell in excess of the loss tolerance during a single measurement interval is called a *loss violation*. Note that in our simulations measurement intervals do not overlap. That is, *measurement interval*<sub>*i*+1</sub> starts at the moment *measurement interval*<sub>*i*</sub> ends.

### 7.6.2 Loss Violations

Figure 44 provides a graphical representation of the loss violation performance of the three schedulers tested in this section. The data depicted in Figure 44 is extracted from Table 10 in Appendix B where a more complete set of results for this section can be found.

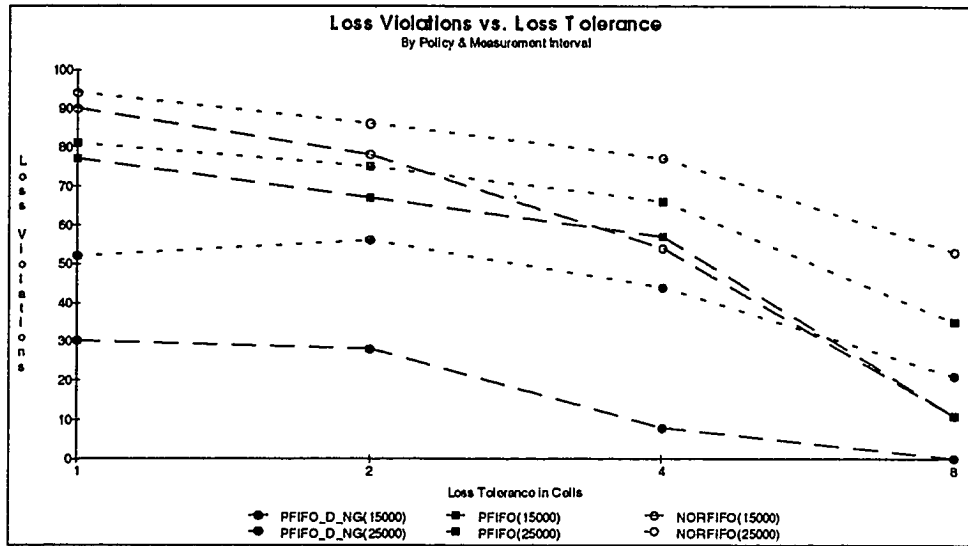
In Figure 44 the dotted lines reflect results for the least tolerant measurement interval of 25000 microseconds whereas the dashed lines show results for the “average” tolerance corresponding to a measurement interval of 15000 microseconds. The same absolute loss tolerance is maintained for all reported measurement intervals. Since the length of the measurement interval increases in each column of Table 10, this effectively represents a net decrease in the loss tolerance. Since we are interested in observing the effects that a set of bursty sources has on the loss characteristics of CBR streams, Table 10 only reports loss violations at Host 2a and Host 2b, as all the CBR streams terminate in these AAL functions.

Figure 44 and Table 10 show that *PFIFO\_D\_NG* (*PFIFO* with displacement) is superior to the other schedulers shown. The number of loss violations is universally less than either *PFIFO* or *NORMFIFO*. This is true for each combination of the test parameters, measurement interval and loss tolerance.

It is obvious from Figure 44 and Table 10 that *PFIFO* enjoys a slight but consistent advantage with respect to *NORMFIFO*. The reason for this is not immediately apparent as both *PFIFO* and *NORMFIFO* are fundamentally FIFO schedulers, and both ignore QoS issues. We must recall that the AAL hosts in our prediction tests use *AALSchedulers* and, therefore, emit a prediction cell once every time cycle. While the prediction cell bandwidth overhead slows the effective queue service rate at *pp3*, the same overhead is experienced on all the links feeding our *pp3* bottleneck. The presence of the prediction cell overhead on these input links reduces the maximum effective arrival rate by  $1/(CellsInPrediction + 1)$ . This slight reduction in the maximum arrival rate occurs on all the input links, and the multiplied effect of these reductions mitigates the congestion at *pp3* because the buffer size at the switch is not correspondingly reduced by  $1/(CellsInPrediction + 1)$ . Therefore, when comparing prediction results to non-prediction results in this chapter, the reader should remember that a small portion of the prediction advantage can be explained by this side-effect rather than by the prediction scheduler itself.



FIGURE 44. Loss Violations vs. Loss Tolerance (*PFIFO\_D\_NG*, *Bottleneck\_Loss*)



## 7.7 CBR Loss Guarantees Only (Global QoS Knowledge)

### 7.7.1 Simulation Description and Motivation

In this section we provide simulation results based on a simulator implementation of the ideas proposed in Section 6.3.2, "Global Knowledge of QoS State." These tests expand upon the tests conducted in Section 7.6 and make use of global knowledge about cell loss. Our intent is that by maintaining knowledge regarding upstream and downstream cell losses we can better meet individual connections' QoS loss requirements.

Since the primary goal of the tests in this section is to determine whether global QoS knowledge can improve QoS performance (this term was introduced in Chapter 1), we need to extend the simple test scenarios of the earlier sections to include multiple network hops. For this reason, the results in this section include tests using the *Cross\_Traffic\_Hom-Link* configuration shown in Figure 74 and discussed in Appendix A, Section A.3.

A secondary goal of these tests is to illustrate differences in the algorithms' performance when some of the NNI links are significantly longer than others. The *Cross\_Traf-fic\_HetLink* configuration discussed in Appendix A, Section A.5 includes some NNI links that are several times longer than (and thus produce several times the propagation delay of) other links in the topology. Thus, this section also includes results based on *Cross\_Traffic\_HetLink*. We do not expect QoS performance to vary significantly for a

given algorithm between *Cross\_Traffic\_HomLink* and *Cross\_Traffic\_HetLink*. This is because the additional propagation delay present in *Cross\_Traffic\_HetLink* will only increase delay and not affect the other metrics of QoS performance. Since this increase in delay will be felt equally by all the algorithms tested, little relative difference in QoS performance should be expected. *Cross\_Traffic\_HetLink* is introduced primarily to illustrate the use of prediction-based scheduling in the more realistic case of a network with varied length network links. Since the longer communications links add latency to the propagation of global knowledge, it is possible that global knowledge may appear somewhat less effective in *Cross\_Traffic\_HetLink* than in *Cross\_Traffic\_HomLink*.

The results reported in this section are for *level loss tolerance* across all reported measurement intervals. Since none of the prediction-based schedulers evaluated in this section pay attention to jitter guarantees, the jitter requirements for *Cross\_Traffic\_HomLink* and *Cross\_Traffic\_HetLink* (stipulated in Table 4 on page 188) are ignored here. Hence, results are only shown for the small-buffer switch case. We consider these jitter requirements as well as the loss requirements in Section 7.8. We defer reporting throughput and link utilization results until Sections 7.10 and 7.11, respectively.

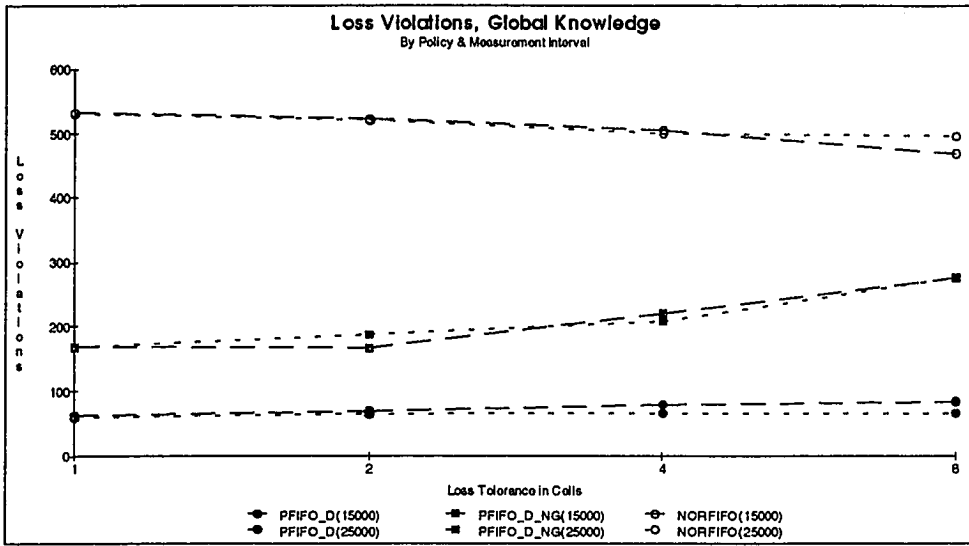
### 7.7.2 Loss Violations

As in Section 7.6, we are interested only in loss results in this section. Figures 45 and 46 provide graphical representations of the loss violation performance of the three schedulers tested in this section. The figures are based on data in Tables 11 and 12 in Appendix B and their interpretation is similar to that of Figure 44 in Section 7.6.2. Tables 11 and 12 report total loss violations for sources *1b-1*, *1b-2* and *1b-3*. These connections are reported because they are the CBR sources that transit the most network hops and, hence, are most subject to network-induced problems. Note that the loss tolerance is varied from one to eight cells per measurement interval for sources *1b-1*, *1b-2*, and *1b-3* only. The loss violations reported here are the sum of the violations suffered by these three connections.

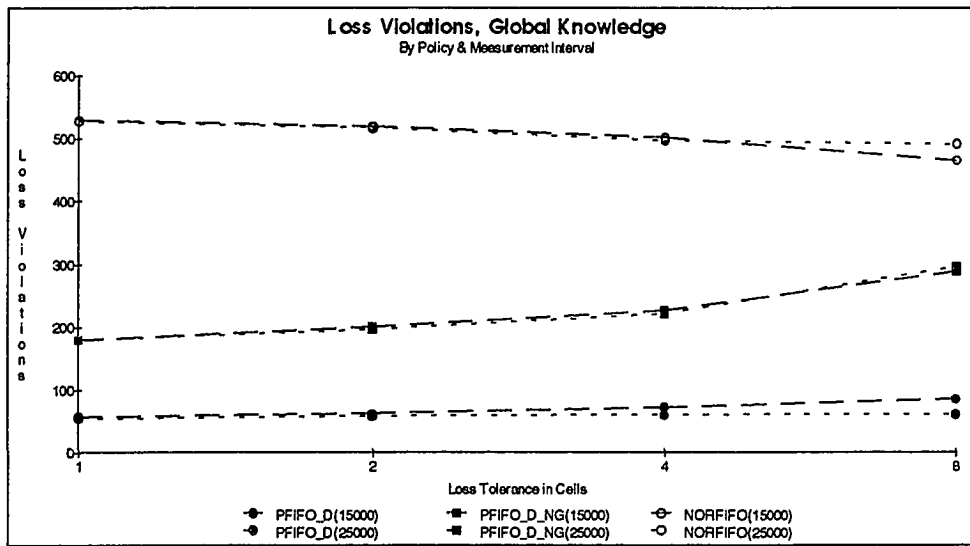
Tables 11 and 12 illustrate that use of global knowledge about QoS state by *PFIFO\_D* does provide a *significant* advantage over *PFIFO\_D\_NG* which uses only local knowledge about QoS state. Loss violation performance of *NORMFIFO* is worse than that of both *PFIFO\_D* and *PFIFO\_D\_NG*. Also, the difference in measurement interval between 15,000 and 25,000 has essentially no effect on these results. (This is due to the *level loss tolerance* across measurement intervals explained above.)

Figures 45 and 46 reveal an unexpected relationship between increased loss tolerance and increased loss violations for *PFIFO\_D* and *PFIFO\_D\_NG*. While *NORMFIFO* displays the expected drop in violations, *PFIFO\_D* and *PFIFO\_D\_NG* actually show a slight increase in violations in spite of the increased loss tolerance. This is due to the fact that the increasing loss tolerance allows these two schedulers to drop additional cells and still stay within their *perceived* loss tolerance. The problem with perceived loss tolerance at a given switch is that it is not well synchronized with loss tolerance as actually measured at the network egress point. With *PFIFO\_D* this lack of synchronization exists because 1) measurement interval boundaries are not coordinated across the different network elements that utilize measurement intervals, and 2) there is delay in propagating global loss information between network elements. The affect of these is compounded in *PFIFO\_D\_NG* since each switch, unaware of upstream and downstream loss, independently assumes that during each measurement interval it may drop a number of cells equal to the loss tolerance. The *PFIFO\_D\_NG* schedulers operating in successive switches along the path of a connection can unknowingly cause violations by independently dropping their full loss tolerance of cells. The sum of these losses over a single measurement interval at the network egress point determines loss violations. Because the cell dropping decisions are performed independently, this sum can exceed the loss tolerance at the network edge despite no individual switch having perceived excessive losses. For this reason, the slight increase in loss violations observed for *PFIFO\_D* is more pronounced for *PFIFO\_D\_NG*.

**FIGURE 45. Loss Violations vs. Loss Tolerance (PFIFO\_D, Cross\_Traffic\_HomLink)**



**FIGURE 46. Loss Violations vs. Loss Tolerance (PFIFO\_D, Cross\_Traffic\_HetLink)**



## 7.8 CBR Loss And Jitter Guarantees Together

### 7.8.1 Simulation Description and Motivation

We have shown that *PTDM* can be effective at improving jitter performance and that *PFIFO\_D* improves cell loss QoS performance. The jitter tests, however, were conducted in the absence of cell loss (i.e. with large buffer switches) and the loss tests were conducted without any sources requesting jitter guarantees. In this section we examine how different cell scheduling policies perform when both loss and jitter guarantees are required. This is the first section that provides results for scheduler *PTDM\_D*, which is described in Section 6.4.4.

The test configuration used is *Cross\_Traffic\_HetLink* (see Appendix A, Section A.3). We select this configuration as the basis for evaluating loss and jitter guarantees together since it represents the most variety of the test configurations used thus far. Both loss and jitter violations are produced with this test configuration. The multiple hops and varied link lengths allow the Global Knowledge propagation feature to be exercised. It is probably the most realistic example of all of the test configurations used up to this point and therefore provides a convenient vehicle to perform a general comparison of the scheduling algorithms that we have thus far studied individually. In this section, as in Section 7.7, loss tolerance is *level* across the different measurement intervals tested.

### 7.8.2 Loss and Jitter Violations

Figures 47 through 49 summarize the loss and jitter violations for both the small buffer as well as the large buffer case. (The detailed tabular data supporting these figures is provided in Tables 13 through 16 in Appendix B. We do not provide surface plots for the large buffer loss violations shown in Table 14 since there is essentially no loss in that case.) As in Section 7.7.2, the figures depict QoS violations for sources *1b-1*, *1b-2* and *1b-3* combined. It is apparent that *PFIFO\_D* fares far better than *PTDM* in providing loss guarantees. It is also obvious that *PTDM* is superior to *PFIFO\_D* in providing jitter guarantees. Neither of these results should be surprising as *PTDM* ignores QoS loss requirements and *PFIFO\_D* ignores QoS jitter requirements. *PTDM\_D*, however, does attempt to provide loss and jitter guarantees simultaneously. The fact that *PTDM\_D* provides loss guarantees favorably affects its jitter performance compared to that of *PTDM*, as illustrated in Figure 49. This is due to the fact that an additional loss will also be reported as a jitter violation.

Note that in some cases a policy that performs very well in providing one type of QoS guarantee performs abysmally with another type of guarantee. For example, the results in Figure 47 show that while *PFIFO\_D* out-performs *PTDM\_D* with respect to cell loss, *PFIFO\_D* is so much worse (Figures 48 and 49) in providing jitter guarantees to the jitter sensitive traffic in the same test, where *PTDM\_D* is clearly preferable. From this we conclude that *PTDM\_D* is the best choice as a general all-purpose scheduler for loss and jitter. *PTDM\_D* clearly out-performs *PTDM* and *NORMFIFO* in all three figures.

FIGURE 47. Loss Violations, 90 cell buffer, *PTDM\_D* vs. others

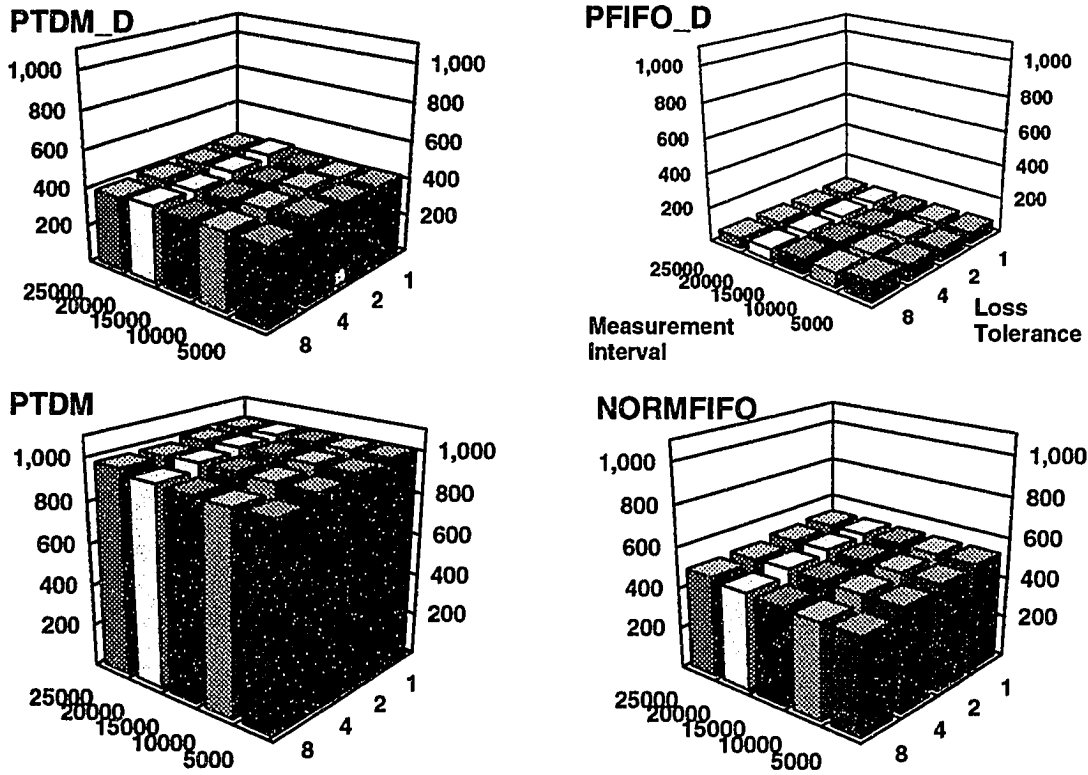


FIGURE 48. Jitter Violations, 90 cell buffer, *PTDM\_D* vs. others

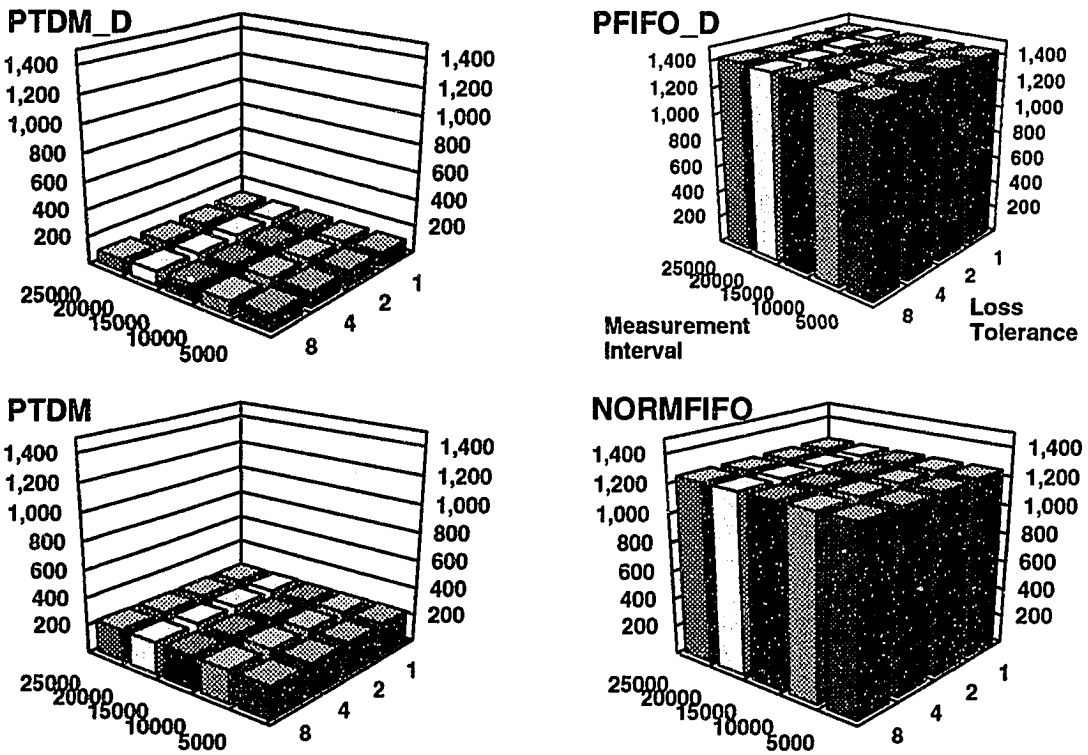
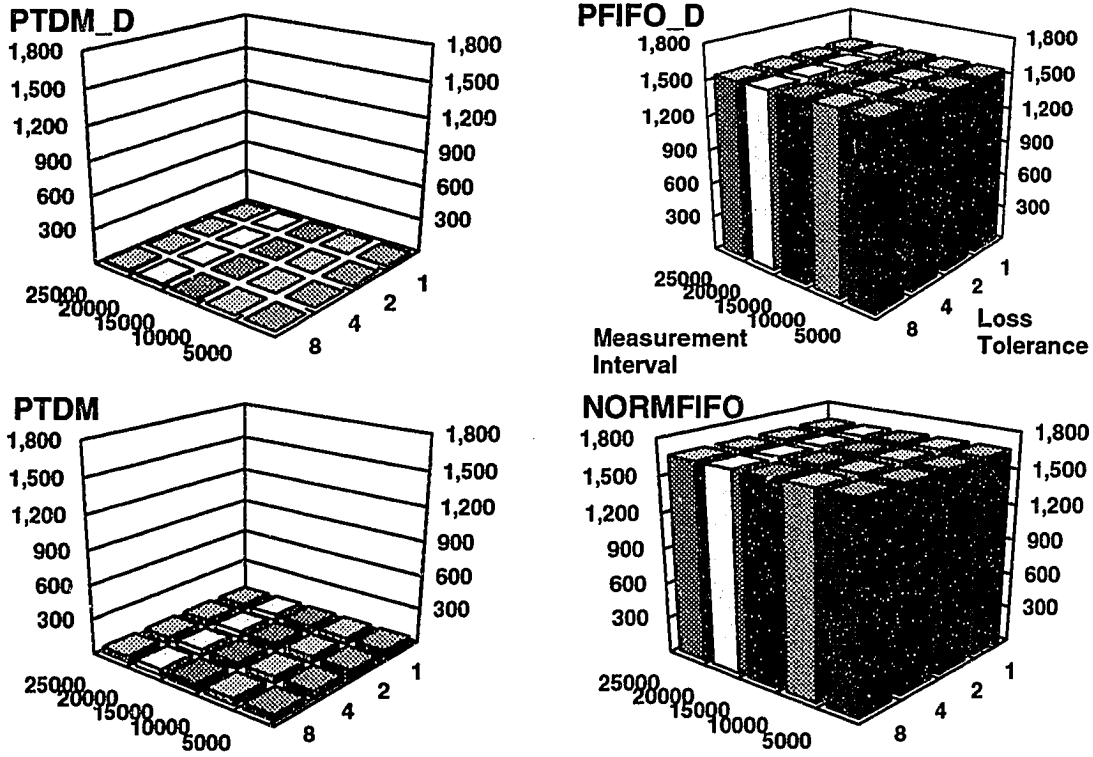


FIGURE 49. Jitter Violations, 9999 cell buffer, *PTDM\_D* vs. others

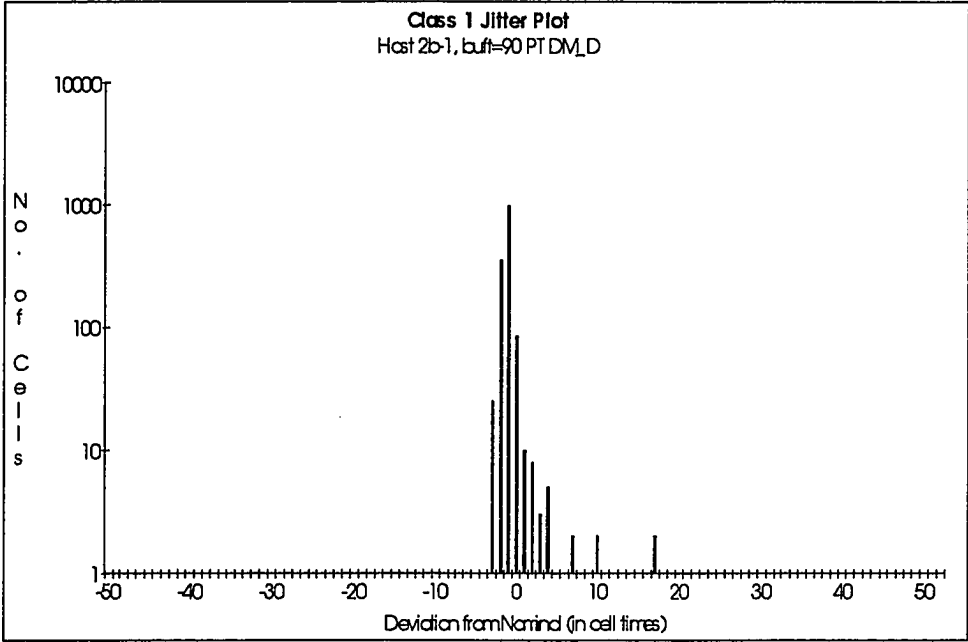


### 7.8.3 Jitter Distributions

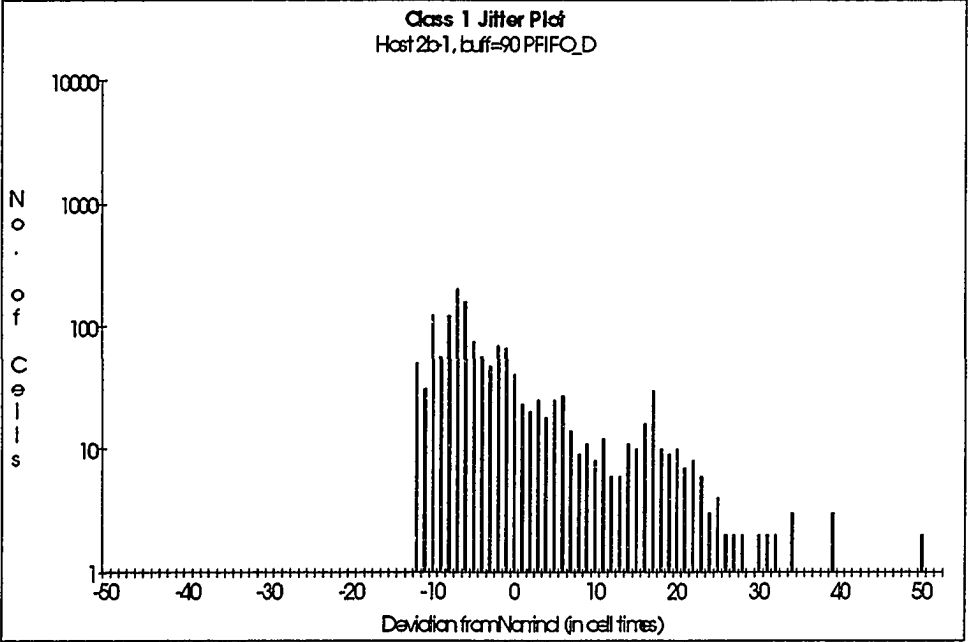
Figures 50-57 show jitter plots for connection *b-1* for the four scheduling policies and two switch buffer sizes covered by Tables 13 through 16 in Appendix B. The test parameters for these plots were a loss tolerance of 2 cells per measurement interval and a measurement interval length of 15000 microseconds. It is clear that the jitter control of *PTDM\_D* is superior to that of *NORMFIFO* and *PFIFO\_D*. The results also show that the jitter control of policies *PTDM* and *PTDM\_D* is much more visible in the large buffer case than in the small buffer case. This is because the cell losses that occur in the small buffer case are also interpreted as jitter violations whereas there are no losses in the large buffer case.



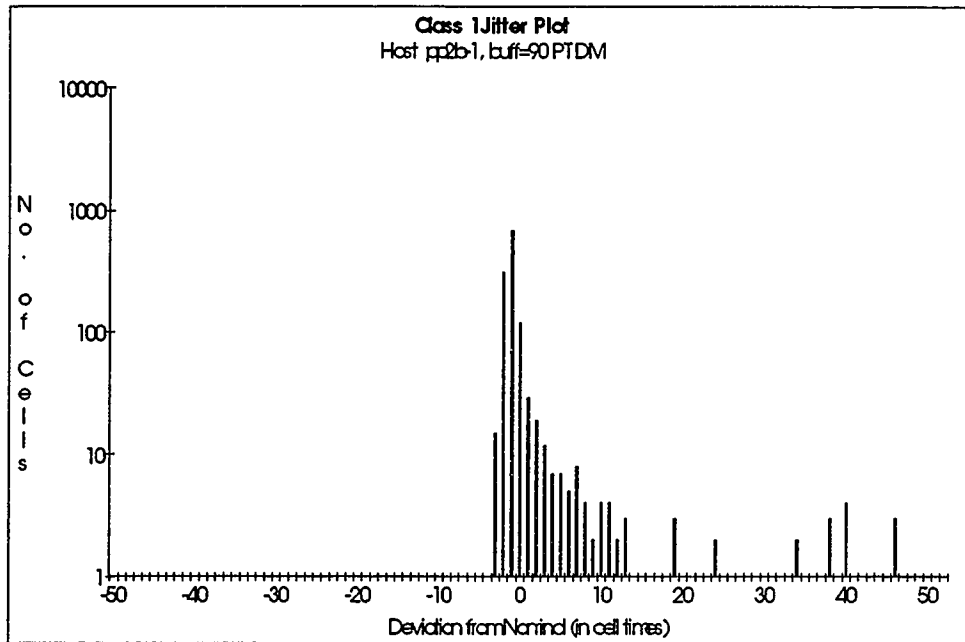
**FIGURE 50. Jitter at Host 2b-1, Small Buffer (PTDM\_D)**



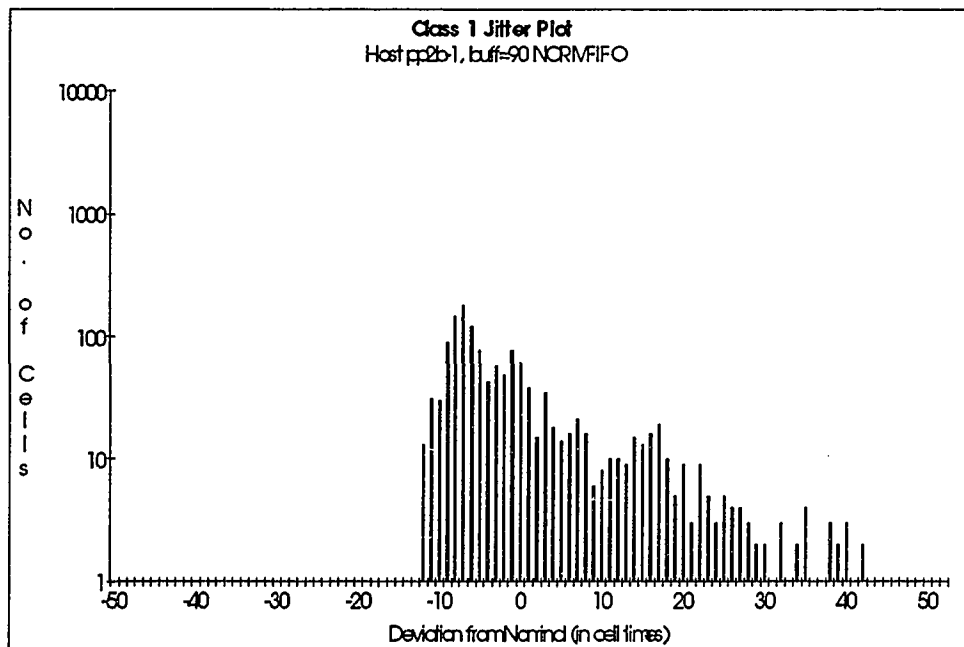
**FIGURE 51. Jitter at Host 2b-1, Small Buffer (PFIFO\_D)**



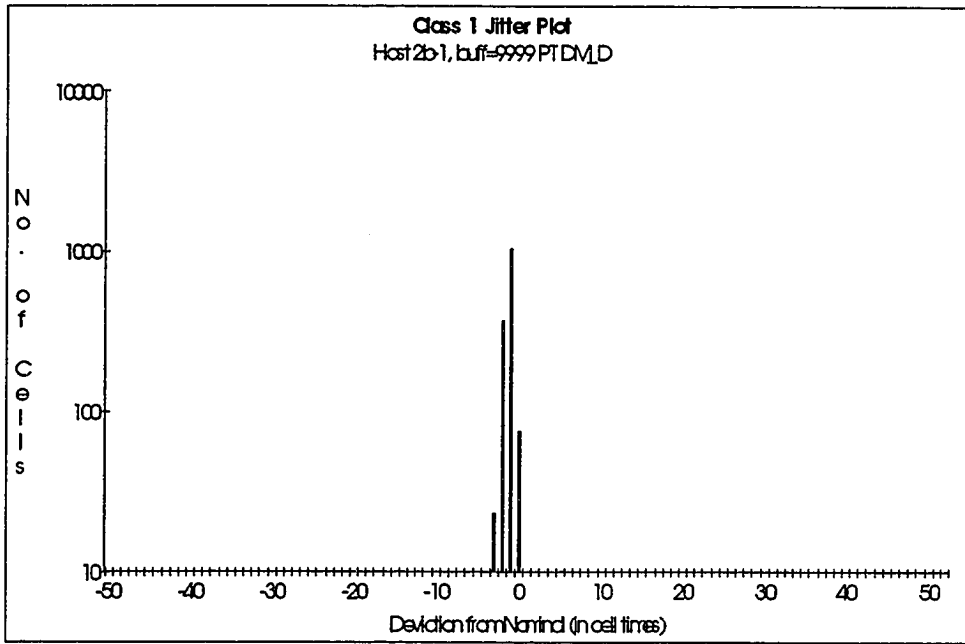
**FIGURE 52. Jitter at Host 2b-1, Small Buffer (PTDM)**



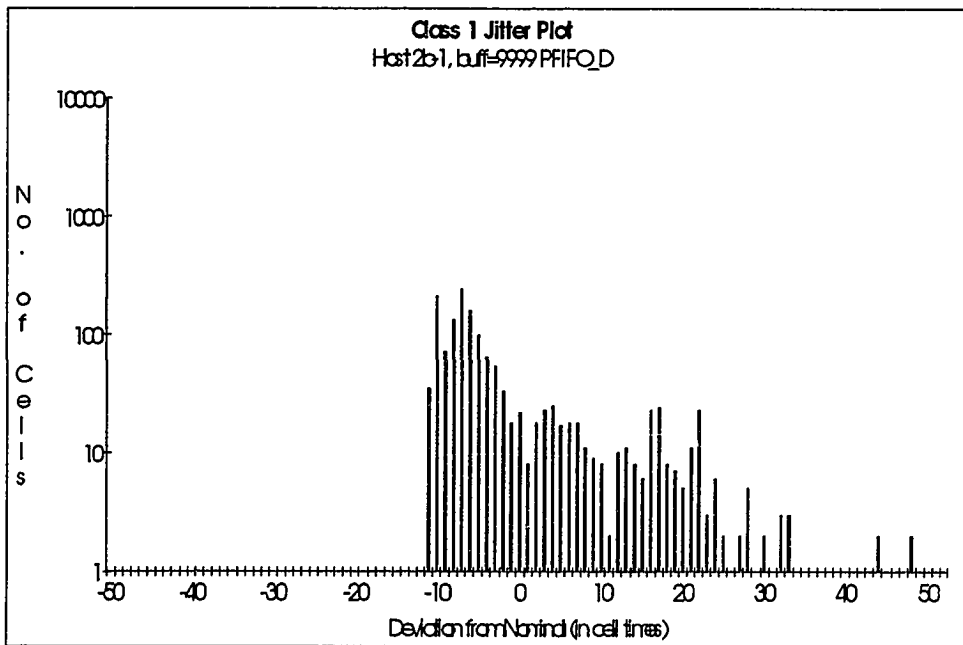
**FIGURE 53. Jitter at Host 2b-1, Small Buffer (NORMFIFO)**



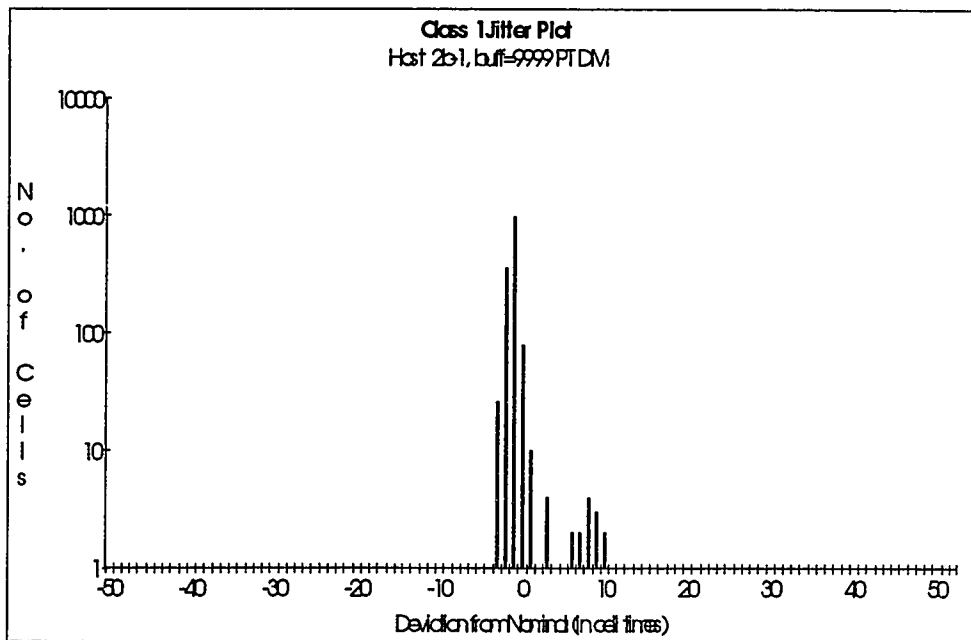
**FIGURE 54. Jitter at Host 2b-1, Large Buffer (PTDM\_D)**



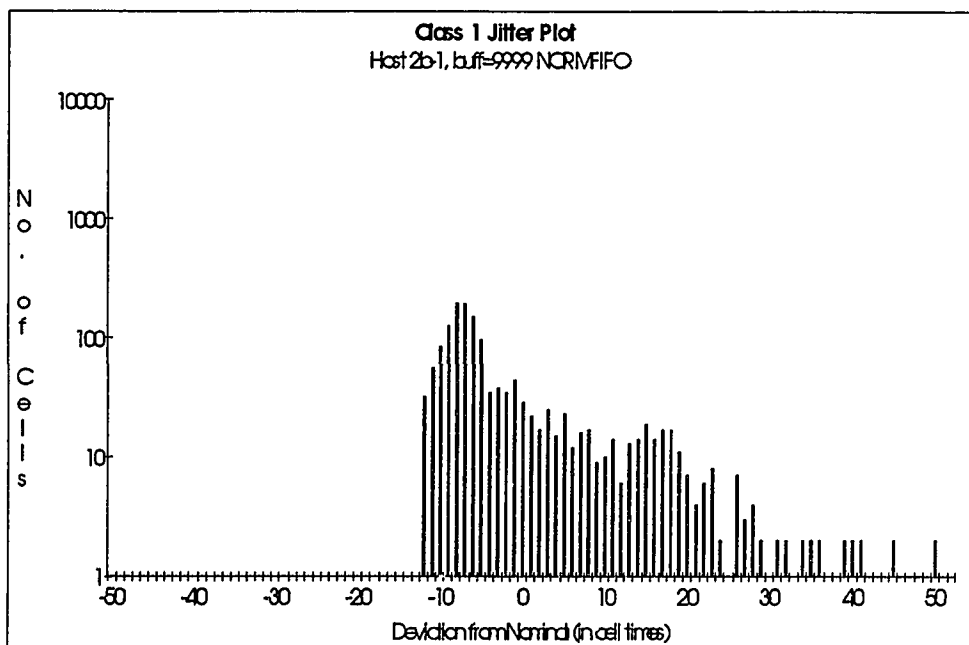
**FIGURE 55. Jitter at Host 2b-1, Large Buffer (PFIFO\_D)**



**FIGURE 56. Jitter at Host 2b-1, Large Buffer (PTDM)**



**FIGURE 57. Jitter at Host 2b-1, Large Buffer (NORMFIFO)**



## 7.8.4 Observations

*PTDM\_D* is superior to policies *PFIFO\_D*, *PTDM* and *NORMFIFO* in an environment that requires both jitter and loss guarantees. Jitter control is easier to demonstrate in large buffer switches than in the high-loss environment of small-buffer switches under overload conditions because losses are also interpreted as jitter violations.

The preceding comparisons focussed on jitter and loss for selected CBR connections only. In the following section we will extend the simulations to measure QoS performance for all active CBR and VBR connections in the network and introduce greater diversity into the QoS requirements of the connections.

## 7.9 CBR/VBR: Loss and Jitter Guarantees Together

### 7.9.1 Simulation Description and Motivation

The connections studied in this section are characterized by more diverse QoS requirements than those used in the results presented thus far. By “more diverse” we mean that the jitter tolerances and loss tolerances vary more noticeably between the different connections than in the first series of tests. Another significant difference is that in this section we measure the overall QoS performance of all the active CBR and VBR connections in the network, rather than focussing on the QoS performance of a selected subset of CBR connections.

The schedulers evaluated in this section include *PTDM\_D*, *NORMFIFO*, and *HOLDISP*. We choose to report prediction-based results for *PTDM\_D* alone here as we concluded in Section 7.8.4 that *PTDM\_D* is superior to policies *PFIFO\_D* and *PTDM* in an environment where overall QoS performance was paramount. In addition to *NORMFIFO*, we now compare results with the dual priority queue scheduler *HOLDISP*, introduced in Section 6.4 on page 101. We include the *HOLDISP* algorithm in order to evaluate *PTDM\_D*'s performance against a non-prediction cell scheduler that is well known to provide better QoS performance than the *NORMFIFO* scheduler we have used for comparison thus far.

The test configuration used is that described as *Cross\_Traffic\_HetTraffic* (see Appendix A, Section A.5). This configuration is identical to *Cross\_Traffic\_HetLink* except for the source traffic characteristics. The connections include 9 jitter and loss-sensitive connections and 14 loss-sensitive connections. Within the 9 that are both jitter and loss sensitive, there is a wide range of jitter tolerances. Similarly, within the 14 loss-sensitive connec-

tions, *Cross\_Traffic\_HetTraffic* provides a variety of loss tolerances. The loss-sensitive set includes both CBR and VBR connections. The jitter-sensitive connections include 8 CBR connections and 1 VBR connection. We provide details of these 23 connections in Table 5 on page 192. The *Cross\_Traffic\_HetTraffic* configuration allows us to vary three important parameters: 1) buffer size of the switches, 2) traffic loading level, and 3) overall jitter tolerance. The ranges and definitions of these three parameters are provided in Sections A.5.4, A.5.5 and A.5.6, respectively.

### **7.9.2 Overall QoS Violations**

Figures 58 through 60 depict the overall (loss and jitter) violations for the full range of buffer sizes, loading levels and jitter tolerances tested. (The detailed tabular data supporting these figures is provided in Tables 17 through 19 in Appendix B.) The figures depict the combined QoS violations for all active connections in the network. The clear superiority of *PTDM\_D* is evident in all three sets of data shown in these figures.

FIGURE 58. Low Jitter Tolerance Overall QoS Violations

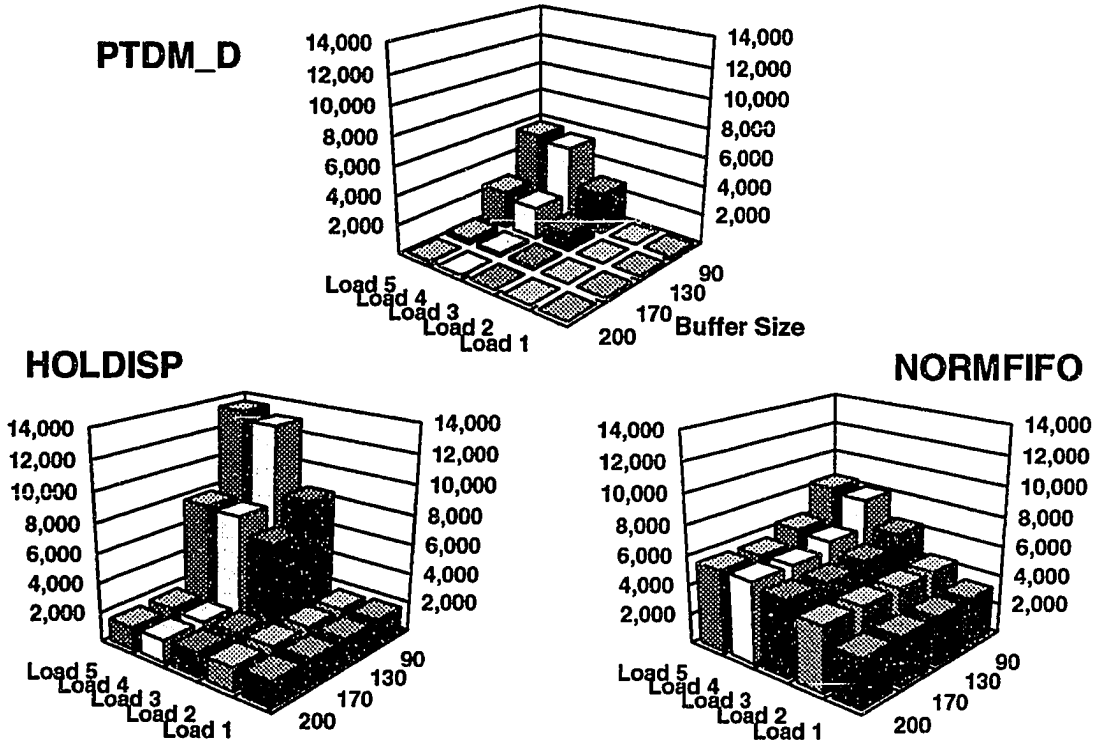
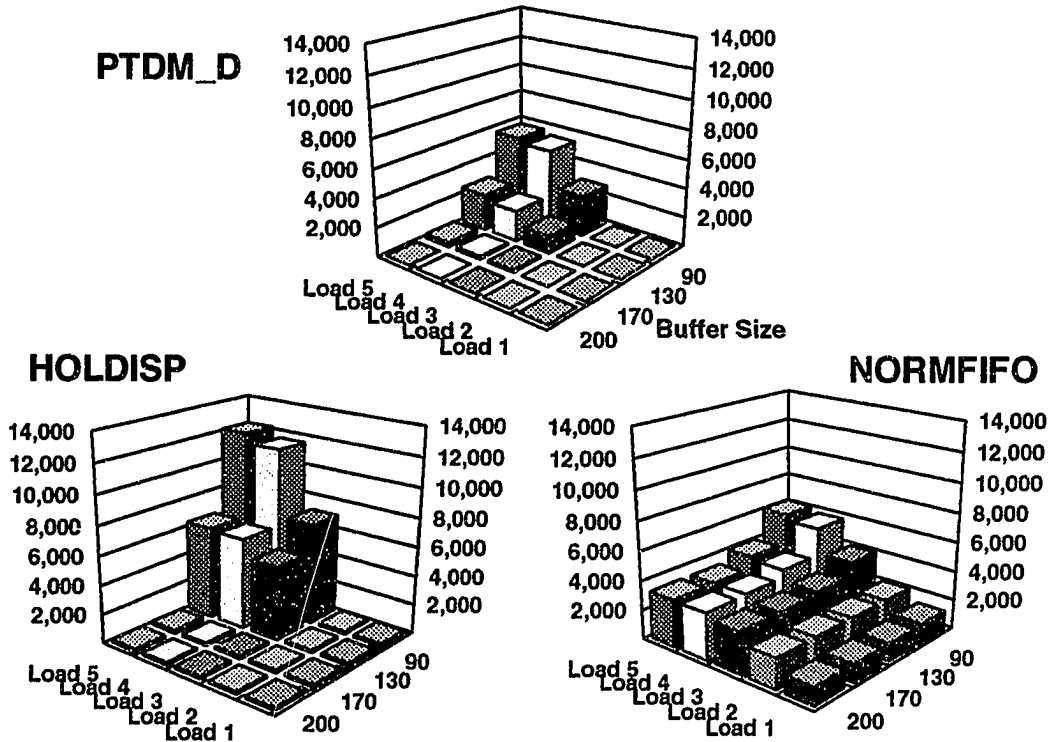
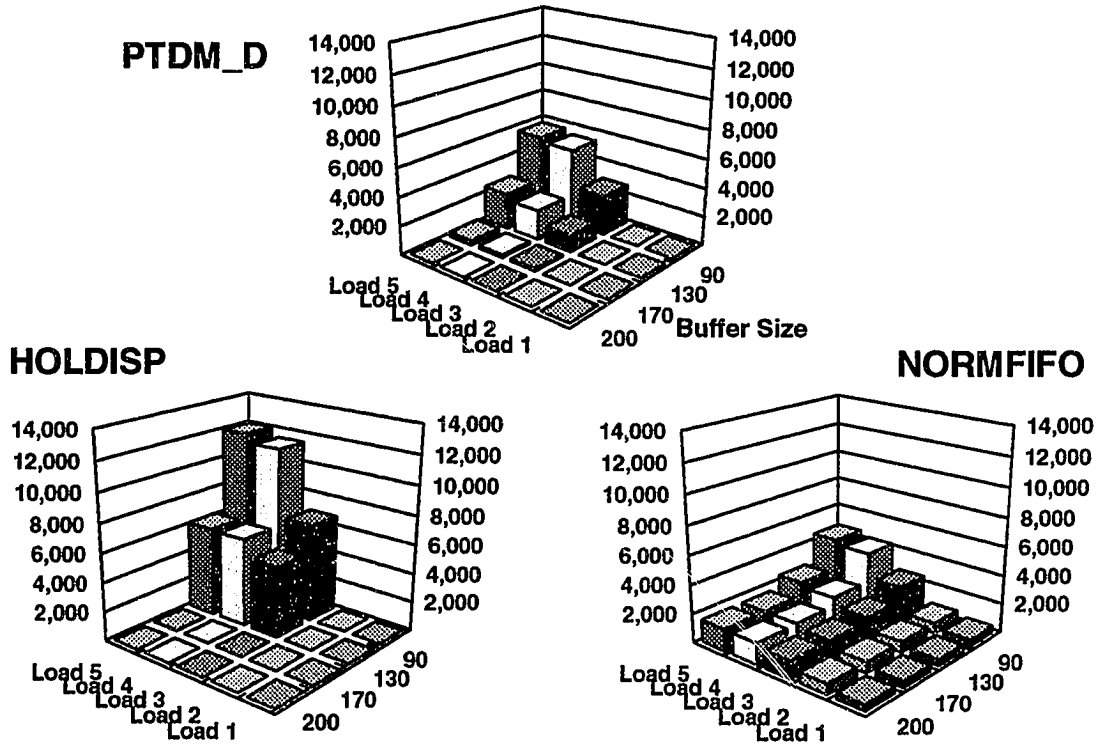


FIGURE 59. Medium Jitter Tolerance Overall QoS Violations



**FIGURE 60. High Jitter Tolerance Overall QoS Violations**



### 7.9.3 Jitter and Loss Violations Individually

Figures 61 through 63 summarize the jitter violations for all buffer sizes, loading levels and jitter tolerances tested. Tables 20 through 22 in Appendix B contain the detailed data from which these figures were derived. The loss violation component of the results is provided in Figures 64 through 66, and the tables corresponding to those plots are Tables 23, 24 and 25 in Appendix B. The sum of the jitter and loss violation counts provided in this section yield the overall violation count provided in the preceding section.



FIGURE 61. Low Jitter Tolerance Jitter Violations Only

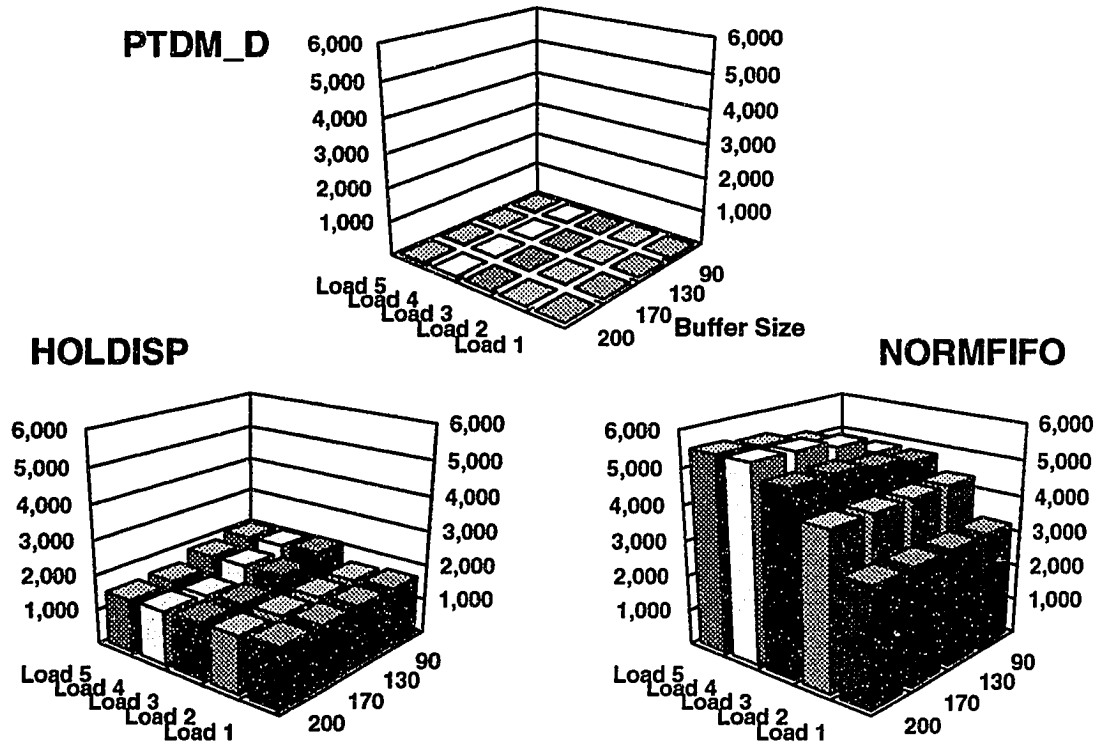


FIGURE 62. Medium Jitter Tolerance Jitter Violations Only

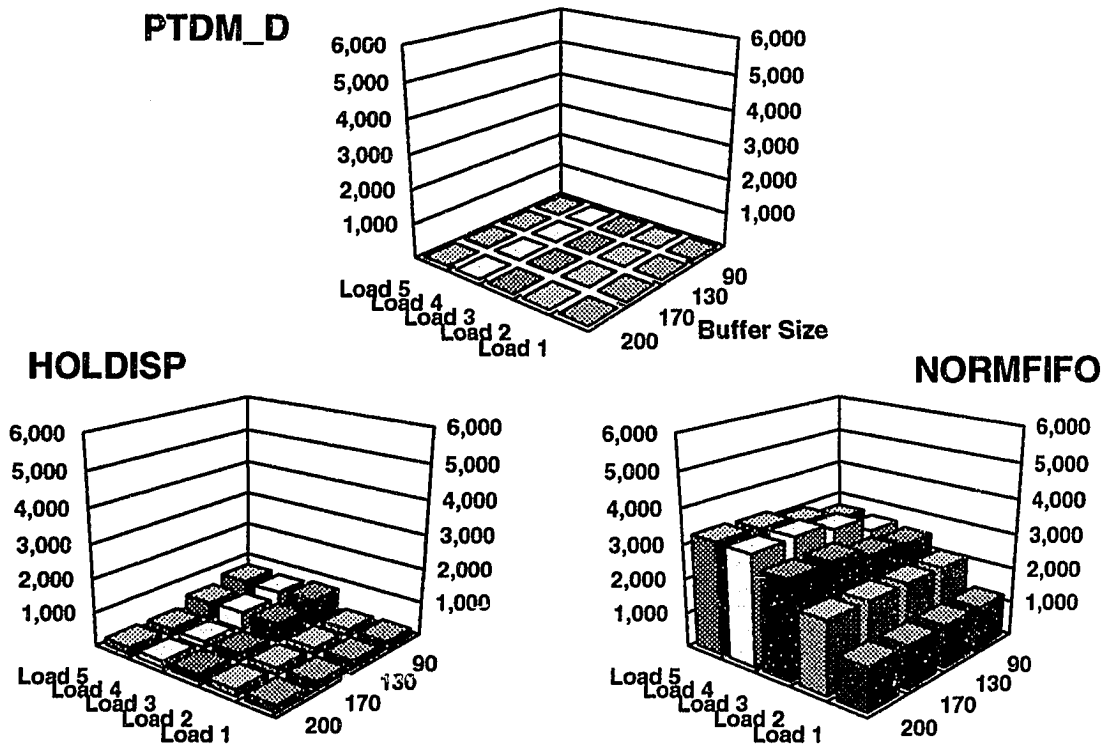


FIGURE 63. High Jitter Tolerance Jitter Violations Only

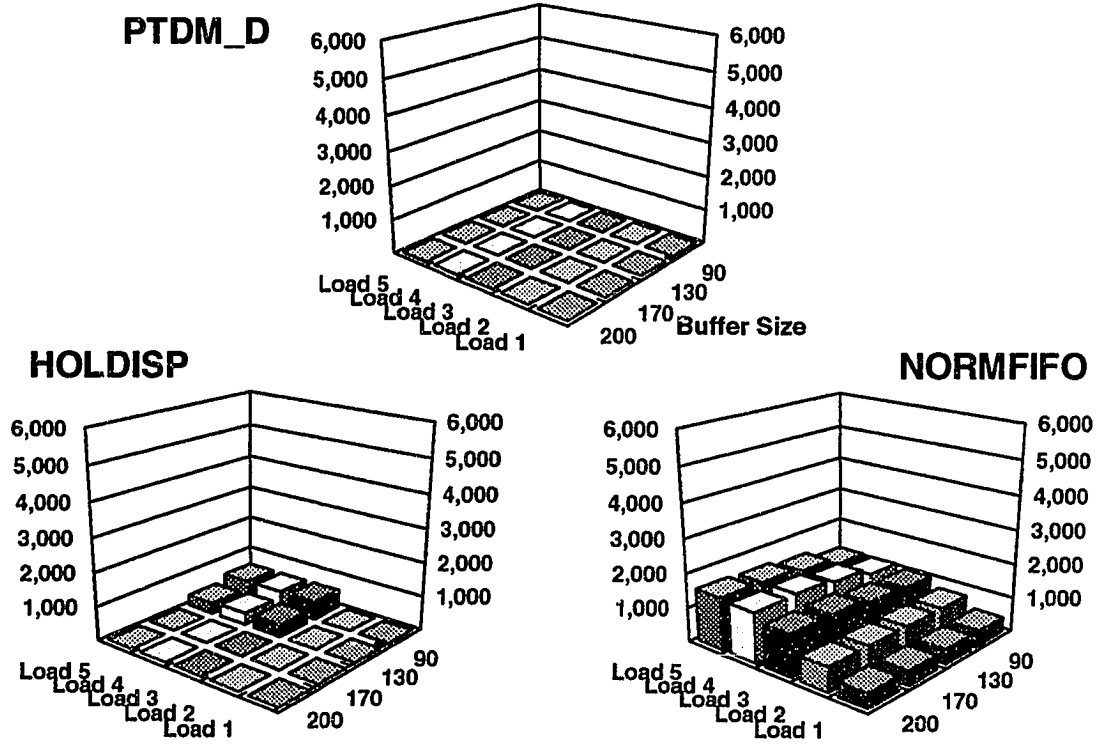


FIGURE 64. Low Jitter Tolerance Loss Violations Only

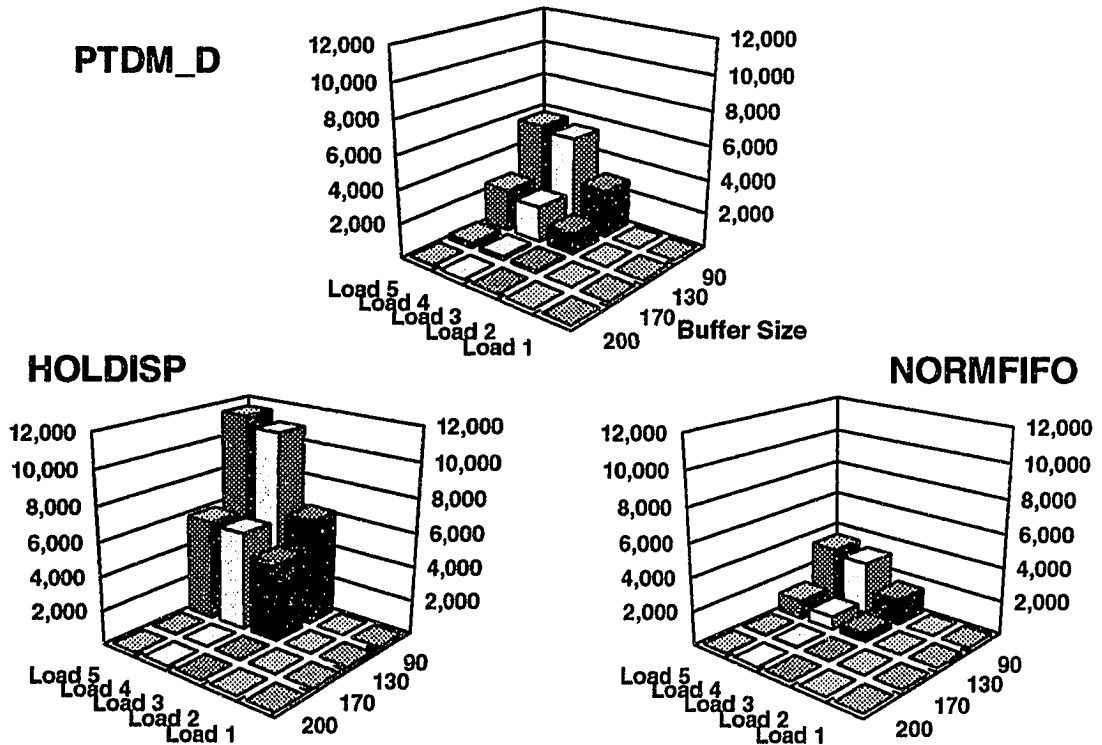


FIGURE 65. Medium Jitter Tolerance Loss Violations Only

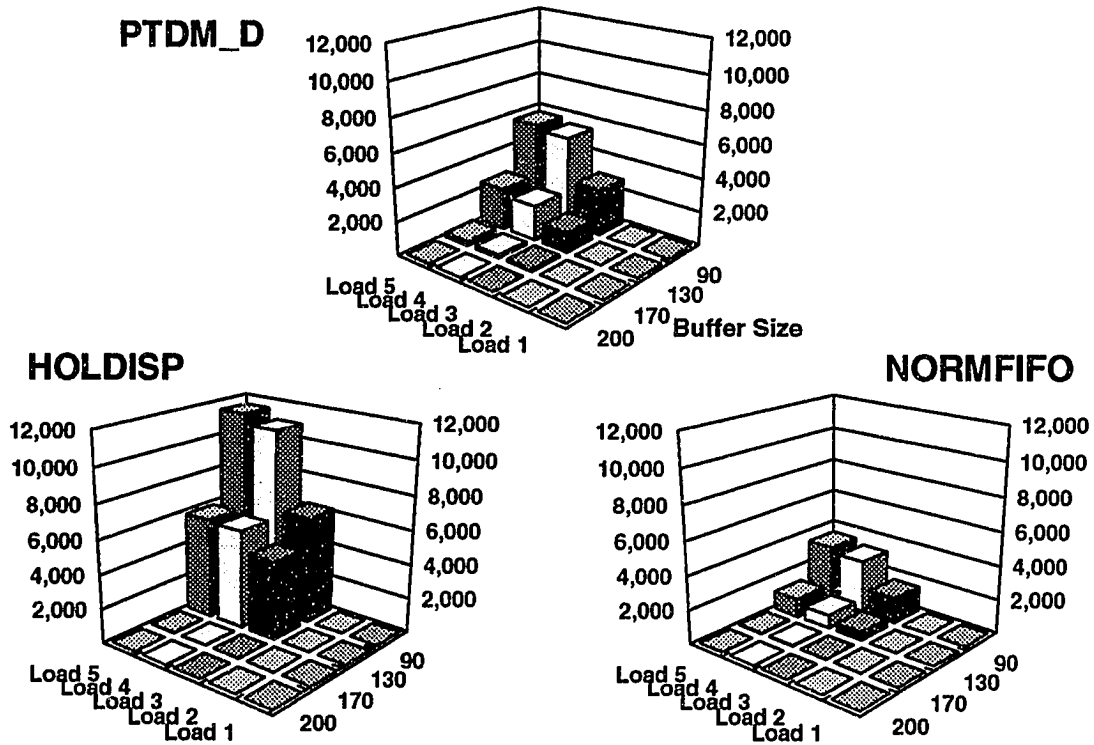
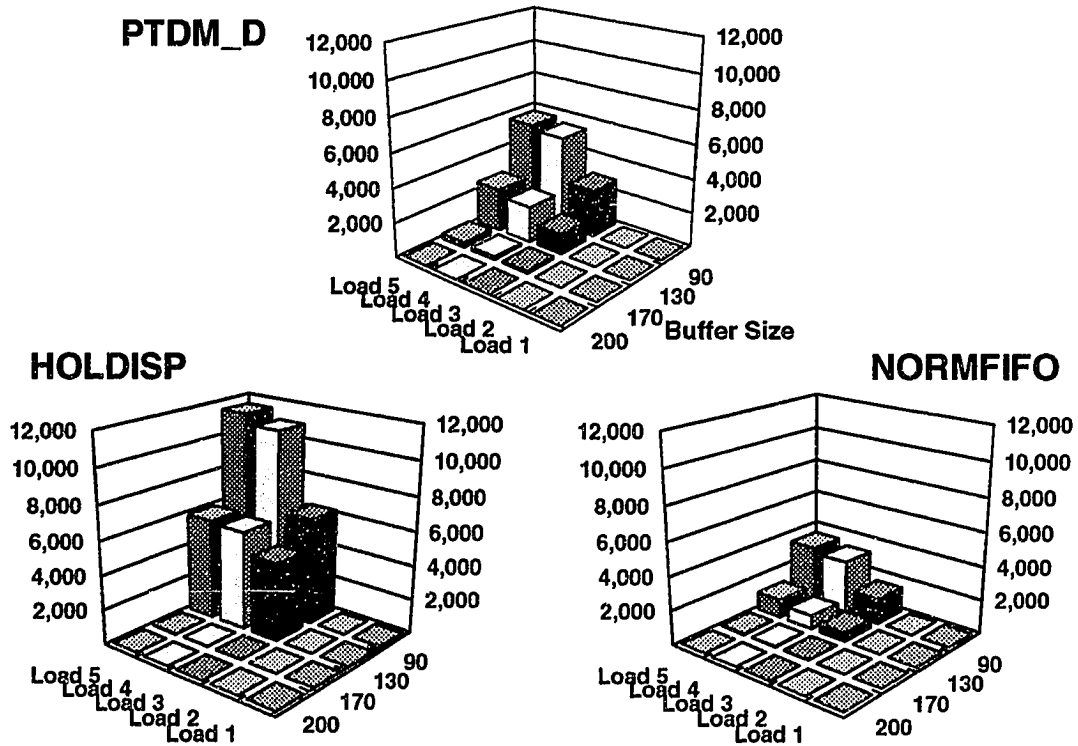


FIGURE 66. High Jitter Tolerance Loss Violations Only



#### 7.9.4 Observations

A number of important trends are observed in Figures 58 through 66. *PTDM\_D* exhibits superior overall QoS performance relative to *HOLDISP* and *NORMFIFO* under all conditions tested. All three algorithms show lower QoS violations as load decreases and buffer size increases. This is a direct result of the lower overall cell loss that results under these conditions. *HOLDISP* and *NORMFIFO*'s overall QoS violations are significantly higher than *PTDM\_D*'s in the case of lower overall jitter tolerance. This is attributable to the fact that even moderate amounts of jitter are considered violations in the low jitter tolerance case, and, unlike *PTDM\_D*, neither algorithm exercises precise control over jitter.

*HOLDISP*'s overall QoS violations diminish, however, to levels comparable with *PTDM\_D* in the case of high overall jitter tolerance, light loads and larger buffers. This is due to the fact that *HOLDISP* prioritizes jitter sensitive traffic, and the lower delay this prioritized traffic suffers directly correlates with lower jitter. While this lower jitter still exhibits too much variance to avoid violations in a low jitter tolerance scenario, it is sufficiently low to avoid violations when there is high jitter tolerance.

Under heavier loads or small buffer conditions, *HOLDISP*'s overall QoS violations remain worse than those of *PTDM\_D*. Inspection of the jitter and loss components of QoS violations (Figures 61 through 66), shows that *HOLDISP*'s overall violations originate primarily from loss violations. *HOLDISP*'s exclusive prioritization of jitter-sensitive traffic has the negative consequence of increased loss violations for the non-jitter sensitive traffic. It is noteworthy that this bias against non-jitter sensitive traffic results in *HOLDISP* having far higher QoS loss violations than the QoS-ignorant *NORMFIFO*.

The merit of enforcing QoS guarantees at the connection level is most evident in comparing not only the overall QoS performance, but particularly the jitter (Figures 61 through 63) and loss (Figures 64 through 66) components individually. By attending to each connection's requirements individually, *PTDM\_D* is able to control each class of violation effectively. *HOLDISP*'s strengths in jitter control lead to exaggerated weaknesses in loss control. *NORMFIFO*'s advantage over *HOLDISP* in loss violations *reverses* when jitter performance is considered. The observation that *PTDM\_D* has a strong showing simultaneously in loss and jitter QoS performance in a diverse traffic mix is a direct result of connection-level control of QoS guarantees.

Up to this point, our evaluation of scheduler performance has focussed on jitter and loss. In the following sections we evaluate the prediction-based scheduling heuristics according

to other commonly used metrics of network performance. These include link utilization, throughput, and delay.

## 7.10 Link Utilization

Figure 67 compares the link utilization achieved for the prediction-based policies *PTDM\_D*, *PFIFO\_D*, *PTDM* and no-prediction default scheduling *NORMFIFO* for the small buffer switch case. Figure 68 provides analogous results for the large buffer switch case. Detailed data for Figures 67 and 68 is found in Tables 26 and 27 in Appendix B. These utilization results are derived from the same tests discussed in Section 7.8. The links' utilization remains constant for the 2 parameters varied in the test (measurement interval length and loss tolerance). For this reason we do not show the utilization figures for all the combinations of measurement interval and cell loss tolerance shown in Tables 13 through 16. Also, since these utilization results support the loss and jitter results in Tables 13 through 16, we report only those links traversed by the connections reflected in these tables. That is, since the tests in Section 7.8 focus on connections *b-1*, *b-2* and *b-3*, Figures 67 and 68 show only the NNI links traversed by those connections (i.e. *pp3a*, *pp3b*, *pp3c* and *pp3d*) and the UNI links reaching each of the connections' receiver network boundaries (i.e. *pp2b-1*, *pp2b-2*, and *pp2b-3*, respectively). We refer the reader to Figure 74 on page 186 for an illustration of where these links are located within the test topology.

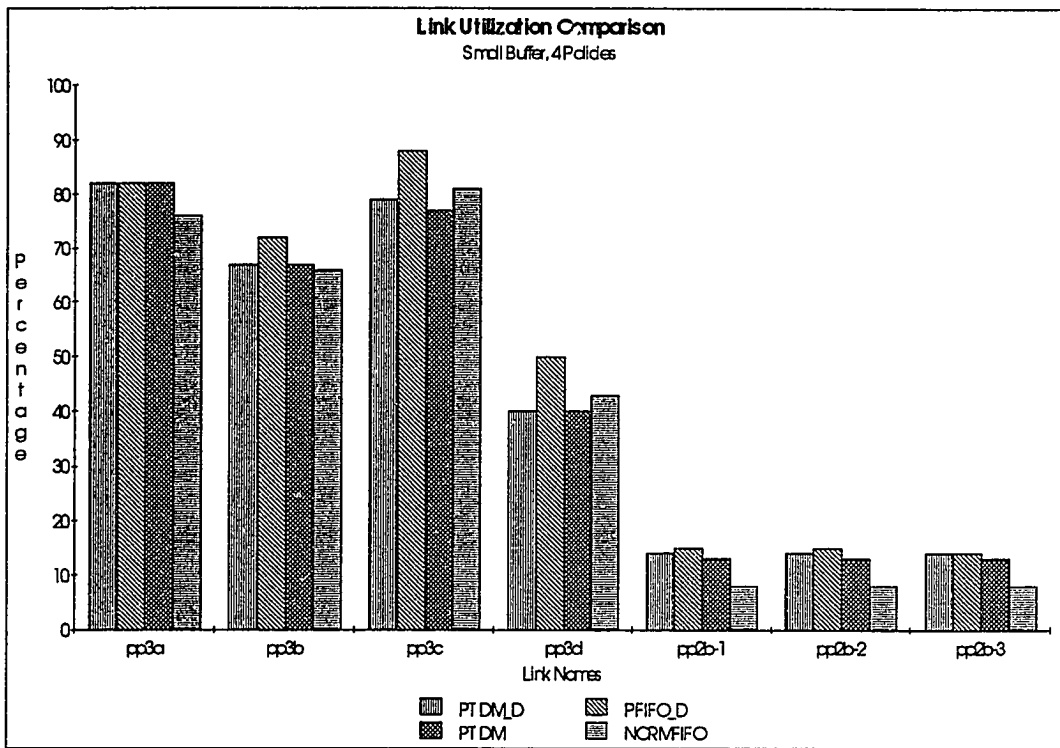
We draw attention to the *upstream* link utilization shown in Tables 26 and 27 for the no prediction results as compared to the three prediction-based schedulers. As there is no upstream user traffic, the upstream utilization is zero for the generic scheduler. The constant six percent utilization shown for the prediction-based schedulers is entirely attributable to the transmission of prediction cells. This overhead of six percent is a function of how many predictions are made in a single 48-byte ATM cell payload. (Note that in our tests one network cycle includes sixteen cells, one of which is a prediction cell and fifteen are data. Therefore, 6.25% (1/16) of the link is always utilized for prediction cells, even when all fifteen data cells are empty.) We discuss strategies to improve on this in Section 6.5.3.

For a given scheduler, *downstream* utilization levels vary greatly between different NNI links due to the different cross traffic patterns of *Cross\_Traffic\_HetLink*. For example, in Figure 67, the fact that we see *NORMFIFO* utilization levels of 76%, 66%, 81% and 42% is explained by this. Also, for a given link, utilization levels vary much more between dif-

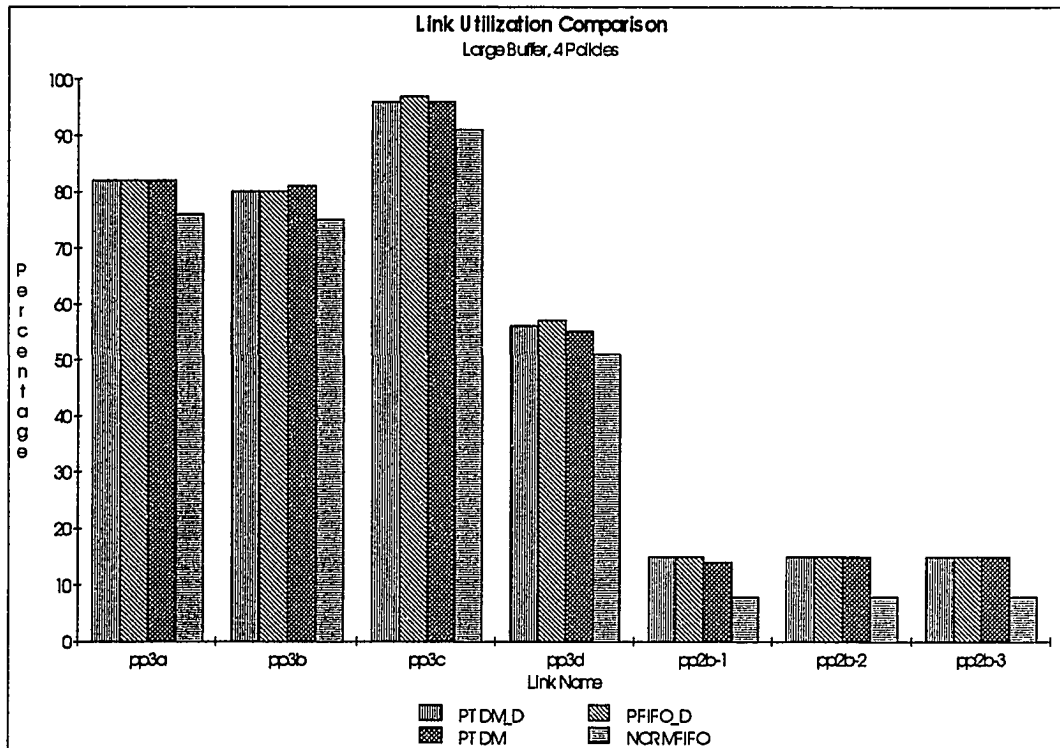
ferent schedulers on the NNI trunks since those links carry considerable other traffic in addition to our selected three connections. For example, in Figure 67, utilization levels for *pp3d* are 40%, 50%, 40% and 43%. The additional traffic includes connections with higher loss tolerance than our studied connections. The prediction-based policies actively delete some of these cells as they strive to maintain other connections' more stringent QoS commitments. These deletions explain the lower utilization levels of some of the prediction policies on the NNI links.

For the UNI links *pp2b-1*, *pp2b-2* and *pp2b-3*, Figures 67 and 68 show that the utilization levels for the three prediction-based policies are 7% higher than the no prediction case. (The difference between this and the 6% reported above for upstream traffic is due to rounding.) The fact that the utilization levels reported for the prediction-based schedulers exceed the generic scheduler is due to the bandwidth consumed by the prediction overhead. Utilizing unused bandwidth is itself not a problem until it begins to constrain throughput. In our tests, throughput was not thus constrained. (Section 7.11 explains that throughput remained high for our tests even with the prediction overhead.) In theory, as the offered load approaches 100% utilization of network bandwidth, the prediction overhead would have a definite negative impact on throughput. We believe that such high levels of utilization are not useful to study, however, as we suspect that acceptable levels of QoS performance will not be achievable with extremely high (e.g., over 90%) loading levels with any ATM scheduling policy.

**FIGURE 67. Link Utilization (%), Comparison of 4 policies, small buffer switches**



**FIGURE 68. Link Utilization (%), Comparison of 4 policies, large buffer switches**



## 7.11 Throughput

Figures 69 and 70 graph the cell throughput achieved for the three CBR sources studied in this section. Detailed data for Figures 69 and 70 is found in Tables 28 and 29 in Appendix B. Since there is generally little or no cell loss in the large buffer switches, the throughputs shown in Figure 70 are higher than those shown in Figure 69. Because of limited space in the figures, the results are reported here only for loss tolerance of 2 and measurement interval length of 15000 microseconds. Since this pairing of the parameter values is used uniformly throughout Figures 69 and 70, the throughputs for the different policies may be freely compared. The throughput figures shown in Tables 28 and 29 are expressed in cells per second. Because the measured duration of our simulations is only 50000 microseconds, the per second throughput is extrapolated from the number of cells received in the 50000 microseconds of the test.

It is clear that for the large buffer switch case (Figure 70), policies *PTDM\_D* and *PFIFO\_D* have virtually identical throughput to that of the default scheduler. This is not true for the small buffer results in Figure 69, where *PTDM\_D*, *PTDM* and *NORMFIFO* show throughputs relative to *PFIFO\_D* of 93%, 83% and 93%, respectively. The small buffer switches produce cell loss for all the scheduling policies in this test. The different throughputs shown are the result of the different amounts of cell loss incurred by the four algorithms. We discuss the relationship of *raw* cell loss vs. cell loss *violations* in the following paragraph.

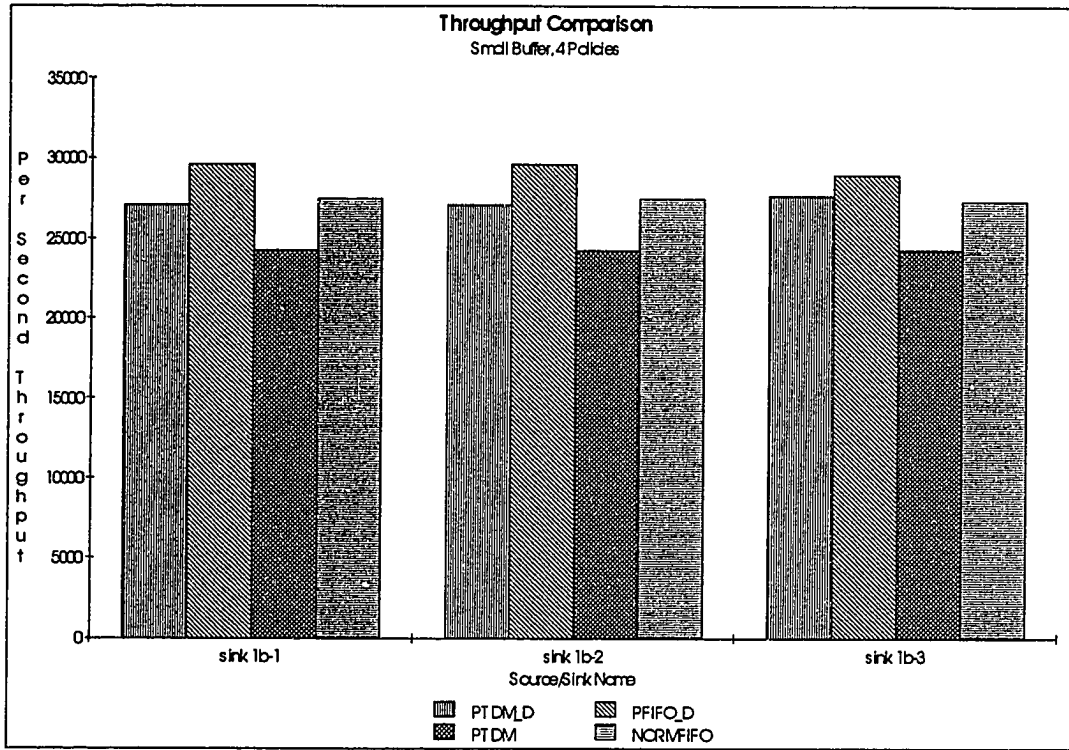
Table 30 on page 216 compares the raw cell loss experienced on the three connections measured in Figure 69 with the total cell loss violations that occurred on those connections. (The loss violations are extracted from Table 13.) The most revealing statistic in Table 30 is the last column, “violations as a percentage of raw loss.” This statistic measures how successful the algorithm is in orienting loss so as to minimize loss violations. Clearly, the most successful algorithm is *PFIFO\_D*. This is because this algorithm gives exclusive priority to QoS loss guarantees. The second best performer in this regard is *PTDM\_D*, which is consistent with our observations that *PTDM\_D* is the best all-around QoS-performer of the algorithms we studied.

While enforcing cell drop policies may *increase* cell loss, we find that, in spite of this loss, the appropriate scheduler can in fact *decrease* the loss violations. Unfortunately, there is no easy way to incorporate this nuance into the traditional throughput metric. The fact that the throughput metric does not capture the improved QoS suggests the need for a new

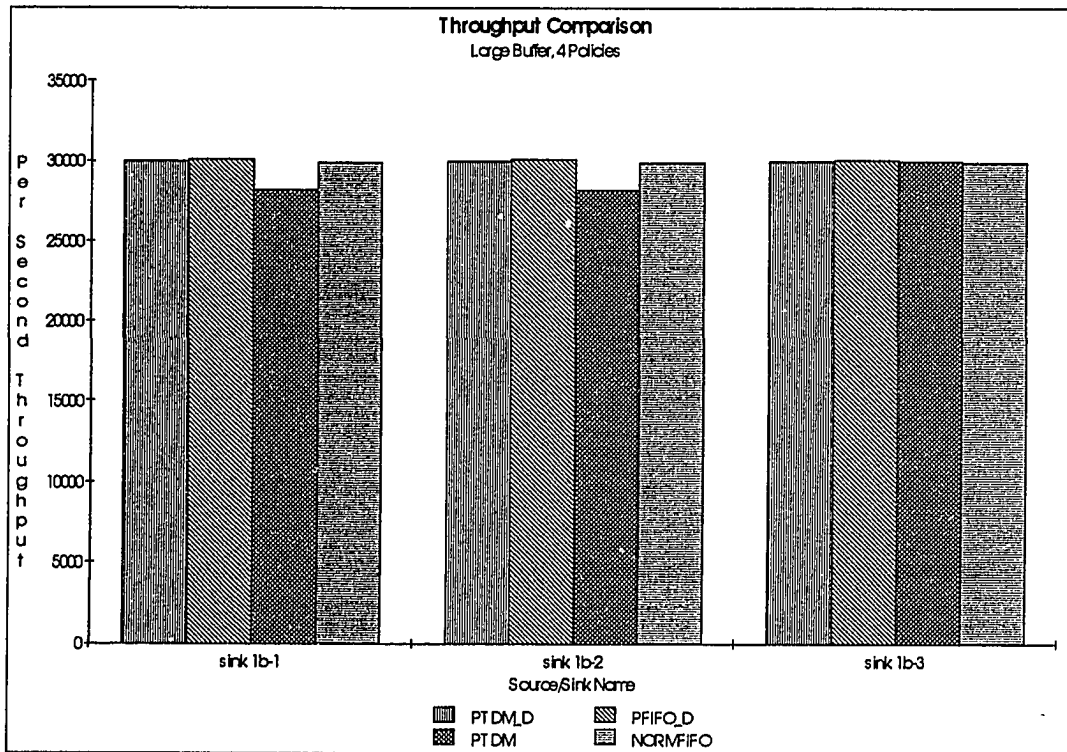


metric. This metric could be called *Effective-Throughput* or *QoS-Throughput*, and would reflect no penalty from the loss of a cell that causes no QoS violation.

**FIGURE 69. Throughput, Comparison of 4 policies, small buffer switches**



**FIGURE 70. Throughput, Comparison of 4 policies, large buffer switches**



## 7.12 Delay

We have stated that a key component of our prediction-based model is the insertion of some delay in order to generate and exploit accurate predictions. In this section we measure that increased delay. Using *Bottleneck\_Jitter* as an example, we see that Figure 42 on page 137 depicts a minimum cell delay of 150 cell times for *PTDM* scheduling, whereas Figure 43 shows a minimum of 47 cell times for *NORMFIFO* scheduling. The delay of 47 is just the serialization and propagation delay over the three links (16 cell times each). The reason for the greater delay in the prediction case is the accumulation of the delays incurred as the cells pass through the different delay buffers of our model. Specifically, the additional 103 (150-47) cell times of delay in Figure 42 is attributable to the components of delay shown in Table 31 on page 216 for a *PredictHorizon* of 3. (Note that as Table 31 indicates, the 103 cell time delay penalty would be reduced to 71 cell times if we were to use a *PredictHorizon* of 1 instead of 3. Reducing the *PredictHorizon* to 1 would not have negatively affected any of the other results presented earlier in this chapter.)

The prediction/no prediction delay differential is even more pronounced in the tests using configurations with more network hops, such as *Cross\_Traffic\_HetLink*. For example, we contrast the 115 cell-time earliest arrival in Figure 72 on page 168 (*NORMFIFO*) with the 260 cell-time earliest arrival in Figure 71 (*PTDM*). These delay distributions are drawn from the tests described in Section 7.8.

We have included a number of additional delay distributions that correspond to the results for *PFIFO\_D*, *PTDM* and *NORMFIFO* that were presented in Tables 13 through 16 in Appendix B. These delay distributions appear in Appendix B, Section B.2. We encourage the reader to examine them in support of the following points related to delay:

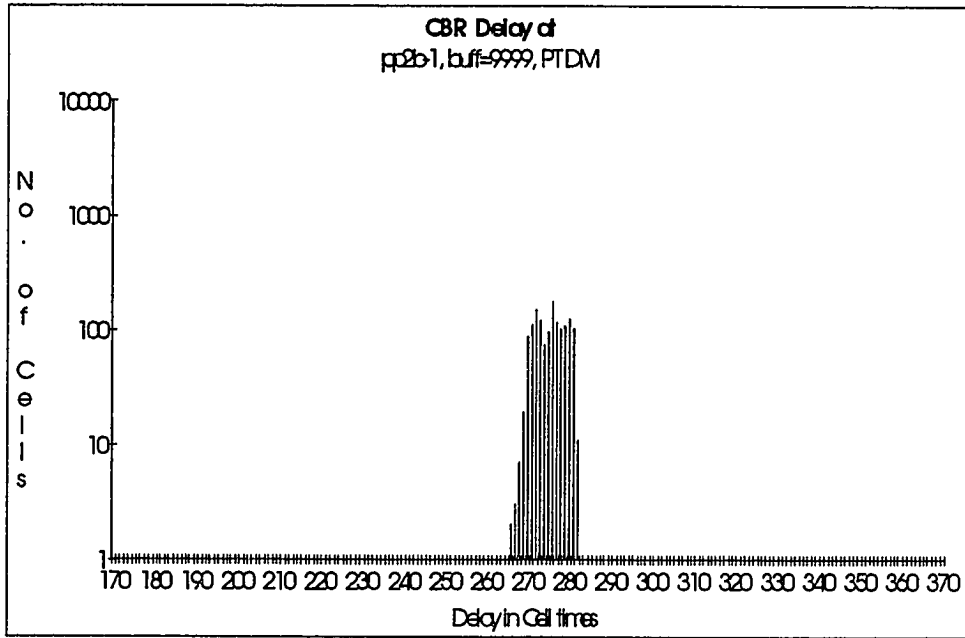
1. We expect increased delay variance to correlate with increased jitter. This correlation can be observed by comparing the jitter depicted in Figures 53 and 52 on page 149 to the corresponding delay distributions in Figures 78 and 80 on page 220, respectively. The greater delay variance evidenced in Figure 78 for *NORMFIFO* correlates with the comparatively high jitter seen in Figure 53. We contrast this with the smaller delay variance shown in Figure 80 for *PTDM* and the correspondingly less jitter in Figure 52.

2. The increased delay variance of *NORMFIFO* may completely negate the advantage that *NORMFIFO* holds with respect to minimum end-to-end delay. The worst case delay for *PTDM* in Figure 80 on page 220 is 266 cell times whereas the worst case delay for *NORMFIFO* in Figure 78 is in excess of 275 cell times.
3. Delay variance is significantly worse for *PFIFO\_D* (Figure 76 on page 218) than for *PTDM* (Figure 80 on page 220). This is to be expected since only *PTDM* applies any explicit jitter control.
4. *PTDM*-style schedulers exhibit good jitter control not only at the network egress point but at intermediate network links as well. This is illustrated in the dual modes in Figure 79 on page 220. This bimodal trait is due to different sets of connections multiplexed over the same hop and scheduled by *PTDM*. The set that experiences less delay traversed fewer network hops (or shorter links) than the other and, thus, experienced less delay. Note that the delay distributions of the two modes have a similar shape and that this bimodal trait, though less obvious, appears in Figure 75 for *PFIFO\_D* and in Figure 77 for *NORMFIFO*. In these figures, however, the sharp bimodal characteristic maintained by *PTDM*'s tight control of jitter is blunted by the lack of jitter control of *PFIFO\_D* and *NORMFIFO*.

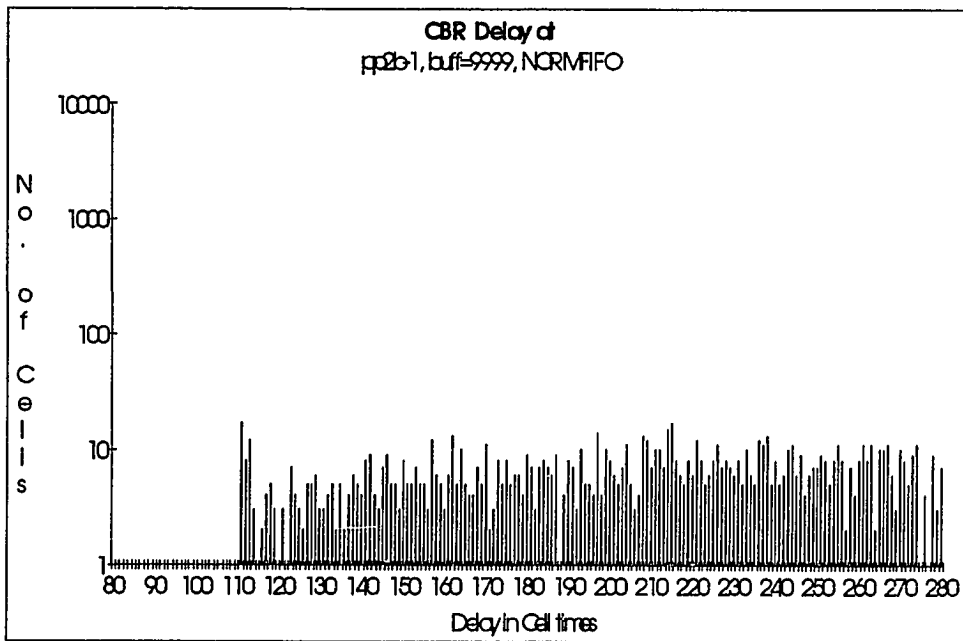
We summarize by listing the following key points regarding delay:

1. The prediction-based schedulers pay a penalty of higher minimum end-to-end delay.
2. Delay variance is strongly correlated with jitter.
3. Improved control over delay variance is not inherent in all prediction-based schedulers. While the *PTDM* family exhibits superb jitter control, *PFIFO\_D* shows the same weakness in jitter control as *NORMFIFO*.
4. Despite having a lower minimum end-to-end delay than the prediction policies, *NORMFIFO* produces a worst-case delay significantly higher than *PTDM*. The benefits of the lower minimum delay are nullified for most delay-sensitive traffic by the worst-case delay scenario.

**FIGURE 71. Time-in-system, pp2b-1, large buffer switch (PTDM)**



**FIGURE 72. Time-in-system, pp2b-1, large buffer switch (NORMFIFO)**



### 7.13 Evaluation

Through a progressive series of tests we have demonstrated that prediction-based scheduling algorithms can provide better QoS performance than the non-prediction-based schedulers evaluated. In particular, *PTDM\_D* simultaneously provides better QoS loss and jitter performance. The improvement is particularly striking for QoS jitter performance. While *NORMFIFO* shows inferior jitter performance in all tests conducted, *HOLDISP* exhibits good control over jitter violations similar to that of *PTDM\_D* under certain operating parameters. *HOLDISP* was shown to be inferior to *PTDM\_D* in two significant ways, however. First, since *HOLDISP* gives exclusive and uniform priority to all jitter sensitive traffic, it performs poorly in providing loss guarantees to other, non-jitter sensitive traffic. Secondly, as jitter tolerances become tighter, *HOLDISP* is unable to provide the near-TDM levels of jitter control of *PTDM\_D*. This superior jitter control of *PTDM\_D* is due in part to the fact that network-induced jitter can actually be mitigated by downstream *PTDM\_D* schedulers. Such proactive amelioration of jitter is not performed by *HOLDISP*.

Our results show that the link utilization achievable by the prediction scheduler is comparable to the default scheduler *NORMFIFO* when the prediction cell overhead is discounted. We have also shown that throughput for the prediction-based schedulers is similar to that of a generic scheduler. If QoS guarantees are considered, the *useful* throughput of the prediction-based schedulers surpasses that of the default scheduler.

At the outset of this thesis we asserted that the single most significant weakness of the prediction-based schedulers is in the area of network latency. The results shown in this chapter corroborate this assertion. The predictions that form the basis of this new scheduling paradigm come at the cost of increased network delay. While the extent of this delay can be reduced for readily predictable traffic, such as CBR, prediction-based scheduling will always increase end to end delay. Thus, in an environment where minimizing delay takes *absolute* precedence over QoS loss and jitter performance, it would be difficult to argue in favor of prediction-based cell scheduling. We believe that while this increase in delay is real, the penalty is usually justified by the greatly enhanced control over QoS loss and jitter performance that is afforded by prediction-based scheduling.

Most significantly, these results demonstrate the feasibility and benefits of providing connection-specific QoS control. In the diverse traffic mix we described in Section 7.9, neither *NORMFIFO* nor *HOLDISP* approaches the overall level of QoS performance of *PTDM\_D* for the wide range of operating parameters and traffic tested. This diverse traffic mix represents a more realistic range of QoS requirements such as those likely to be

exhibited in ATM networks. We suggested at the beginning of this thesis that as the diversity of network traffic grows, QoS guarantees must ultimately be enforced at the connection-level. The results in this chapter support that notion and provide explicit examples of how connection-specific QoS control can improve overall QoS performance.

## 7.14 References

- [1] R. Chauhan and R. Russell. Simulation of ATM networks. Technical Report TR-94-19, University of New Hampshire, December 1994.
- [2] C. Fang, H. Chen, and J. Hutchins. Simulation analysis of TCP performance in congested ATM LAN using DEC's flow control scheme and two selective cell-drop schemes. ATM Forum 94-0119, January 1994.
- [3] P. Goransson. Bandwidth reservation on a commercial router. *Computer Networks and ISDN*, volume 28, no. 3, pages 351-370, January 1996.
- [4] Rony Holter. SONET - A Network Management Viewpoint. In *IEEE INFOCOMM 91*, volume 1, pages 131-136, June 1991.
- [5] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM 88*, pages 314-329, September 1988.



## Chapter 8

# FUTURE WORK

### 8.1 Intelligent Scheduling without Prediction

We have shown that we can use predictions of cell arrivals as input to scheduling algorithms that produce improved QoS performance. The most striking results regarding jitter (Section 7.5.2, Section 7.8.3, Section 7.9.3) are only evident when there is considerable cell queueing taking place. While the predictions certainly allow us to see farther into the future, the large numbers of queued cells provide a database on which similar scheduling decisions could be performed without the costs of prediction. Since these cells are already waiting for service, one can argue that complex schedule computation can be performed on some of the waiting cells while the cells closest to receiving service represent “committed cells” which are not subject to reordering or discard. This would allow a relatively complex out-of-band schedule computation to be performed on the “uncommitted queue.” The justification as to why this computation can be more complex is similar to that which we have provided for the prediction-based schedulers: the computation does not introduce any delay between the transmission of individual cells and thus does not adversely impact throughput.

A drawback of this approach is that it does not work uniformly when there are few or no queued cells. This is because there is no “waiting cell database” on which to compute schedules, and the implementation would have to elect between the alternatives of dynamically introducing delay just to provide computation time or dynamically abandoning intelligent scheduling when queues are small. While some would argue that there is less need for enforcement of QoS guarantees when there is no queueing, we demonstrated that

the presence of the intelligent queuing on a *lightly loaded* link downstream from the point of congestion markedly reduced the upstream-induced jitter. The fact that the system would have to dynamically switch between the normal intelligent-scheduling mode and either of these two alternatives probably poses implementation challenges.

Despite this drawback, we believe that the advantage of eliminating predictions make this a worthwhile area to investigate.

## 8.2 Prediction Cell Compression

In order to achieve high network utilization, it is imperative that in exchanging predictions and schedules, distributed cell scheduling not impose a large bandwidth overhead on the communications links. It is possible to use compression techniques used by predictive CODEC's to reduce this bandwidth consumption (e.g., [9]). Neighboring switches can simultaneously compute a prediction using available history only. If the neighboring switches use the same prediction algorithm and base their prediction on histories that are old enough to coincide, *these predictions are guaranteed to be identical*. The upstream switch then compares its computed prediction to the schedule  $S$  that *LinkScheduler* derives from arriving prediction cells (which are still guaranteed 100% accurate, similar to those without compression). The upstream switch need only communicate the differences between the computed prediction and  $S$ . In many cases there may be no difference and nothing need be communicated. In particular, as the percentage of CBR traffic increases the accuracy of the computed predictions will increase. In general, this can achieve a significant compression while still propagating 100% accurate predictions throughout the network. While we have not implemented these ideas, they represent possible avenues for reducing the bandwidth overhead attributable to prediction cells.

## 8.3 Higher Level Protocol Packet Protection

Other researchers have already recognized that very high penalties are paid within the ATM network if the cell discard policy discards cells indiscriminately of their role in a multi-cell higher level protocol packet. For example, if a 1470 byte TCP buffer is transmitted over ATM, this is segmented into approximately 30 AAL5-encapsulated ATM cells. If any of these 30 cells is dropped, the entire packet is discarded upon delivery at the receiver traffic boundary. Clearly, it is unwise to drop only one of the cells in such a packet and continue to transmit the remainder, which are doomed. Different discard policies related to cell groups have been studied in [10].

It would be interesting to continue the research conducted in this thesis by extending the cell discard policies suggested in [10]. In particular, all of those policies try to deal with a cell sequence after it has begun to arrive. Unfortunately, frequently the congestion that forces a cell scheduler to drop one or more of the cells in a packet is only detected after some cells of that packet have already been forwarded. For example, *LinkScheduler* might foresee a multi-cell packet in the input horizon and, although the initial cells can be transmitted, some of the later cells will arrive during a period of congestion and will have to be dropped. By taking advantage of the predictions available in our model, we can anticipate loss-producing congestion and begin discarding the cells of a cell sequence starting at the first cell. Discarding all of the cells of the sequence in this way not only limits the loss to a single packet, but additionally reduces network load by not forwarding (as many) doomed cells as the non-prediction methods.

#### **8.4 ABR Flow Control Schemes**

A significant body of research has been performed by the ATM Forum's members to evaluate different schemes for flow control of ABR traffic within ATM networks. These schemes can be divided into two classes, closed-loop rate-based control schemes [4] and hop-by-hop credit based control schemes [8]. Several papers have appeared describing the performance of ABR transport protocols such as TCP over ATM networks with different flow control schemes [3]. A comparison of the trade-offs between the two classes can be found in [7].

Under the rate-based control model, control information indicating the maximum rate the source can transmit is provided by the network to the source. In the credit-based paradigm, a sender initially holds a predefined number of credits that is some function of the buffer space in his receiver-peer. He may only send cells while he has credit, and he decrements his credit count for each cell sent. The receiver-peer reissues credit back to the sender at a rate that is a function both of the cells received from that sender and competition for his buffer space from other senders. Both models can incur bandwidth overhead for the communication of control information.

Since the focus of this research is to evaluate a mechanism to improve the QoS guarantees that an ATM network can provide, we have concentrated our work on the CBR and VBR traffic classes. We believe that it would be fruitful to investigate the interaction of prediction-based scheduling with ABR flow control schemes. There are two areas of particular interest. The first is to study whether the prediction-based scheduler adversely affects the

performance of ABR traffic, and whether or not the QoS guarantees provided to the CBR/VBR traffic suffer in the presence of the flow control scheme. The second area for study would include investigating a possible marriage of the predictions with the ABR flow control schemes to see if benefits can be derived. One benefit would be a possible bandwidth savings by piggybacking control information for the flow control scheme onto the prediction cells. Another benefit might be that knowing about the arrival of an ABR cell *before* it arrives might allow for returning the credit for that cell earlier than would normally be possible.

The *Defer\_Q* queuing method discussed in Chapter 6 has already been used to successfully simulate ABR connections (TCP) in our simulator. While we have not had sufficient time to investigate the behavior we observed, the fact that we have laid this groundwork may yield results in further research using our simulator in the ABR area.

## 8.5 Phased VBR Sources

Frequently, the cause of congestion and cell loss in an ATM network is the simultaneous arrival of bursts of traffic from VBR sources. Presumably, the call admission process accepted these connections with the expectation that statistical multiplexing would normally keep such overlapping of bursts at low levels. Nevertheless, it is not possible to preclude this possibility and simultaneously strive for high network utilization.

It is noted in [2] that if VBR bursts can be conveniently phased so that their arrivals at network nodes do not overlap, high network utilization can be achieved with virtually no queuing in the network switches. Several burst reservation schemes have been proposed to dynamically reserve network bandwidth on a burstwise basis [1].

It may be possible to exploit the fact that we already induce delay in our AAL functions and to tune this network edge delay on a burstwise basis to align with the phase boundaries established at connection set up time. This presumes a call admission function that assigns phases to VBR sources in such a way as to minimize the likelihood of overlapping bursts. In this way, we may be able to achieve the same goals sought by burst reservation schemes without the latency inherent in waiting for a reservation acknowledgment for each burst.

## 8.6 Incomplete Predictions

This research would explore a hybrid cell scheduler capable of transmitting cells that are not predicted at all, which we describe as incomplete predictions. We denote this type as

type C scheduling, contrasted with the scheduling classes discussed in Section 6.3.1.1, "Future Time Slot Assignment (FTSA)," and in Section 6.3.1.2, "Deferred FIFO Queueing." While type B scheduling waits for an otherwise unused prediction slot to predict a cell awaiting scheduling, type C scheduling utilizes an otherwise unused cell time to send a waiting type C cell *without first predicting it*. Type C scheduling has the advantage that delay-tolerant cells suffer less queueing delay in the switch than under type B scheduling, but has the disadvantage of violating the principle that a downstream switch always has complete foreknowledge of arriving traffic. It would be of interest to determine if there is any value to having the complete foreknowledge for cells that have no strict requirements on cell delay variation. If such foreknowledge is of no benefit, then class C scheduling might be appropriate for ABR and UBR services.

## **8.7 Multicast**

This research has not specifically addressed how prediction-based cell scheduling would deal with multicast VCIs. This area merits further study.

## **8.8 Packet Networks**

There is an existing base of experience for providing a level of QoS guarantees in traditional packet networks such as the Internet [5] [6]. While this thesis focuses on how prediction-based scheduling can be used to improve the ATM network's capability to provide QoS guarantees, it would be of interest to investigate if the idea of prediction-based scheduling could be applied in packet-based networks.

## **8.9 Global QoS Information About Cell Delay**

The notion of Global QoS State described in Section 6.3.2 does not contemplate global knowledge of a cell's cumulative delay. This seems to be a very difficult problem. To have such information available would significantly buttress the Global QoS state that we have implemented in our work.

## **8.10 Extending the Simulations in This Thesis**

### **8.10.1 Comparison to Other Researchers' Jitter Results**

It would be interesting to compare already-published jitter results from simulations of Partial Buffer Sharing or HOL priority queueing that provide concrete sets of results to those

presented in this thesis. One of the drawbacks here is that other researchers have tended to diffuse the computational problems related to connection-specific QoS parameters and only study a small number of traffic classes. This makes comparison difficult.

### **8.10.2 Alternative Measures of DistanceFromQoSViolation**

This would experiment with ideas proposed in Section 6.6, "Distance from Quality of Service Violation Metrics." This would examine QoS loss performance for alternate measures of *DistanceFromQoSViolation*.

## **8.11 References**

- [1] P. Boyer et al. A reservation principle with applications to the ATM traffic control. *Computer Networks and ISDN*, 24:321–334, 1992.
- [2] R. Chauhan and R. Russell. Simulation of ATM networks. Technical Report TR-94-19, University of New Hampshire, December 1994.
- [3] C. Fang, H. Chen, and J. Hutchins. Simulation analysis of TCP performance in congested ATM LAN using DEC's flow control scheme and two selective cell-drop schemes. ATM Forum 94-0119, January 1994.
- [4] ATM Forum. ATM user-network interface specification ver. 3.0. September 1993.
- [5] P. Goransson. ST2 and resource reservation. In *First Annual Conference on Telecommunications R&D in Massachusetts*, volume 5, pages 38–49, October 1994.
- [6] P. Goransson. Bandwidth reservation on a commercial router. *Computer Networks and ISDN*, volume 28, no. 3, pages 351-370, January 1996.
- [7] A. Iwata, I. Mori, H. Suzuki, and M. Ott. ATM connection and traffic management schemes. *Communications of the ACM*, Vol. 38, No. 2:72–89, February 1995.
- [8] H. Kung and T. Blackwell. Adaptive credit allocation for flow-controlled VCs. ATM Forum 94-0282, March 1994.
- [9] P. Pancha and M. El-Zarki. Bandwidth requirements of variable bit rate MPEG sources in atm networks. In *IEEE INFOCOMM 93*, volume 3, pages 902–909, March 1993.
- [10] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. In *ACM SIGCOMM 94*, pages 79-88, October 1994.

## Chapter 9

# CONCLUSION

The problem of trading off high network utilization versus QoS guarantees is a challenging one confronted during recent years by researchers worldwide. Our research explored some novel insights into the problem in hopes of identifying ways that the ATM network of the future can realize its promise of good QoS performance. Novel insights we have made in the course of our research include:

- Judicious insertion of a small delay at key points in the network can normalize the network and remove a large stochastic component from the network behavior.
- We developed a mechanism to generate accurate short-term predictions about cell arrivals based on the insertion of a small delay.
- The normalized network permits easy distribution of these predictions to the locations where they are needed.
- The added network regularity and the predictions can be used as the basis for heuristic cell schedulers that improve QoS performance.
- QoS guarantees should be provided on a per-connection basis. Forcing connections to conform to one of a small number of QoS classes is just an implementation convenience that artificially restricts the QoS options available to the user.
- QoS loss violations must be distinguished from cell loss. A lost cell does not always constitute a QoS violation.

- We developed the *PTDM\_D* scheduler and showed that it can concurrently provide loss and jitter QoS guarantees on a per-connection basis.
- Our jitter results demonstrate that use of predictions allows a scheduler to decide to send a jitter-sensitive cell earlier than its ideal transmission time to avoid impending competition for that ideal time slot. Without the anticipation of the future enabled by the predictions, jitter-sensitive scheduling can actually add to congestion in attempting to provide good jitter control. In general, it is impossible to know when it is wise to artificially delay a cell without this anticipation.

We have shown that we can produce accurate predictions about future network traffic, and that these predictions can ultimately produce improvements in QoS performance. These improvements are most notable in the area of jitter control, where other scheduling techniques have exhibited considerable weakness. Although penalties paid are increased average delay and added scheduling complexity, these did not adversely affect the overall network performance and, in fact, improved QoS performance. This is an original approach to scheduling in those cases where a significant percentage of the presented load is jitter sensitive traffic, and that jitter sensitive traffic is itself comprised of traffic with distinct degrees of jitter and loss tolerance. Actual test simulations performed in the course of this research support the capability of this scheduling method to improve QoS performance under this type of traffic scenario, and probably others as well.

These insights can yield an improvement in the utilization/QoS violation quotient for future networks. In short, we hope this work will be of great value as researchers and engineers continue to refine ATM as the hybrid network of the future.



# Appendix A

## SIMULATION

## CONFIGURATIONS

In this chapter we provide details about the four simulation configurations used to produce the results presented in Chapter 7.

As the topology shown in Figure 73, "Bottleneck Simulation Topology," is intended to provide a simple scenario for studying loss or jitter QoS guarantees, we have intentionally excluded cross traffic and multiple network hops as part of the simplification. This topology is used for both configuration *Bottleneck\_Jitter* as well as *Bottleneck\_Loss*.

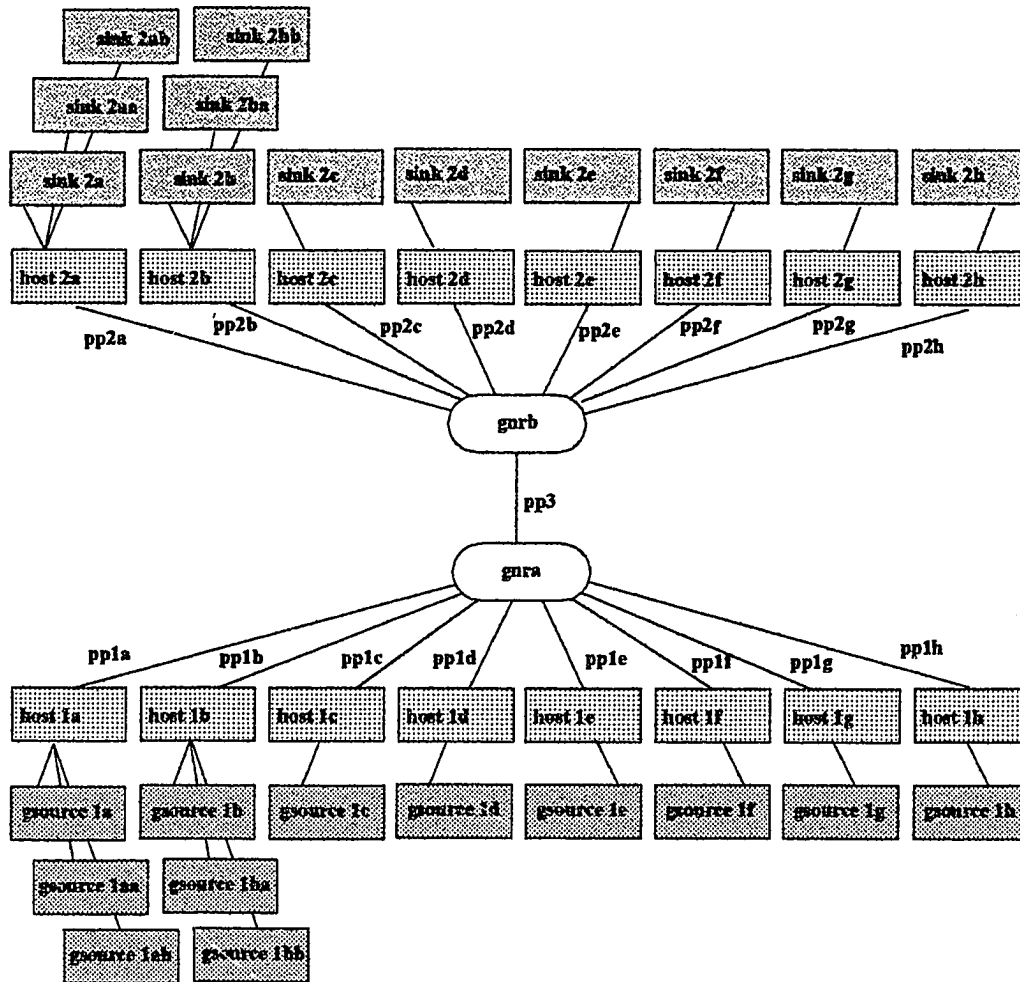
We include cross traffic, multiple hops and varied-length communication links in the topology depicted in Figure 74, "Cross\_Traffic Simulation Topology." We use this topology as the basis for the *Cross\_Traffic\_HomLink*, *Cross\_Traffic\_HetLink* and *Cross\_Traffic\_HetTraffic* configurations. These configurations are used in Chapter 7 for a series of tests to evaluate scheduling methods in a more realistic network scenario than that provided by the *Bottleneck* topology.

### A.1 Bottleneck\_Jitter

The traffic, link and switch configurations for *Bottleneck\_Jitter* provide a simple vehicle for the evaluation of the ability of different scheduling algorithms to control jitter. In this configuration we have selected VBR traffic patterns so as to perturb the CBR traffic flow. These bursts fan in towards the bottleneck link *pp3*. Large buffer switches (9,999 cells per output link) are used to prevent packet loss during these periods of excess arrivals. These

large buffers prevent loss by allowing packets to queue. This is accompanied by the side effect that the amount of time a packet spends buffered may vary greatly from one packet to another. This variability is perceived at the receiver network boundary as jitter.

**FIGURE 73. Bottleneck Simulation Topology**



### A.1.1 Traffic Sources

#### CBR Sources

Figure 73 depicts six identical CBR sources, *gsource 1a*, *gsource 1aa*, *gsource 1ab*, *gsource 1b*, *gsource 1ba*, and *gsource 1bb*. These sources have virtual circuits established to the traffic sinks, *sink2a*, *sink2aa*, *sink2ab*, *sink2b*, *sink2ba*, and *sink2bb*, respectively. Each

source generates ATM cells at a constant rate of 0.030000 cells per microsecond, or 30,000 cells/second.

### VBR Sources

Figure 73 depicts six identical VBR sources, *gsource1c*, *gsource1d*, *gsource1e*, *gsource1f*, *gsource1g*, and *gsource1h*. These sources have virtual circuits established to the traffic sinks, *sink2c*, *sink2d*, *sink2e*, *sink2f*, *sink2g*, and *sink2h*, respectively. Each source is an On-Off source. The active period lasts exactly 105 microseconds, and the silent period lasts exactly 3,895 microseconds. During its active period, each source generates ATM cells at an active rate of 0.353000 cells per microsecond, or 353,000 cells/second. Since cells are only emitted at this rate during the source's active phase, the *average* number of cells per second is:

$$\frac{105}{105 + 3895} \times 353000 = 9266.25$$

#### A.1.2 Communications Links

The communications links are all assumed to be 8.351 kilometers in length, so the link propagation delay (assuming 5085 ns/km propagation in the optical fiber) is 42465 nanoseconds, which is exactly 15 cell times (one cell takes 2831 nanoseconds to transmit at OC-3 rates). Note that the *effective* propagation delay must additionally include the one cell time attributable to cell serialization, so the effective propagation delay is 16 cell times.

#### A.1.3 Quality of Service Requirements of Sources

The CBR sources in this configuration stipulate a target cell interarrival time (*qos\_dly\_par1*) of 33,332 nanoseconds, with a jitter tolerance (*qos\_dly\_par2*) of 28,310 nanoseconds (10 cell times at OC-3 rates). The VBR sources in this configuration impose no requirements related to jitter. The interarrival time of 33,332 ns is the direct result of the CBR emission rate of 30,000 cells per second (i.e.  $1/30000 = 33332 \times 10^{-9}$ ). The jitter tolerance equals 10 cell times at OC-3 rates. We found that selection of much lower jitter tolerance (e.g., 2-3 cell times) invariably resulted in huge numbers of jitter violations due to the presence of the bursty traffic used in this configuration. This was true regardless of the cell scheduling discipline used.

The CBR and VBR sources in this configuration have no loss tolerance at all. That is, any cell loss constitutes a QoS violation. This is done to create an environment where jitter can be studied independently of side effects related to cell loss.

## A.2 Bottleneck\_Loss

In this configuration we modify the traffic and switch characteristics used in *Bottleneck\_Jitter* to provide an environment for the study of QoS loss guarantees in a simple environment. The topology is the same as *Bottleneck\_Jitter* (Figure 73), but two of the VBR sources are disabled and transmit no cells (*gsource1g*, *gsource1h*). The ATM switches used here have limited buffer space (90 cells per output link). As in *Bottleneck\_Jitter*, the VBR bursts fan in towards a single link (*pp3*) and cause it to enter periods of congestion. Unlike the large buffer switches of *Bottleneck\_Jitter*, this periodic high cell arrival/service ratio cannot be absorbed by queueing and instead results in cell loss due to lack of buffer space. For those cells that are not dropped, little difference in the queueing delay is experienced. Thus, this configuration does not provide an interesting scenario for the study of jitter and is used only for studies related to cell loss.

### A.2.1 Traffic Sources

#### CBR Sources

Figure 73 depicts six identical CBR sources, *gsource1a*, *gsource1aa*, *gsource1ab*, *gsource1b*, *gsource1ba* and *gsource1bb*. These sources have virtual circuits established to the traffic sinks, *sink2a*, *sink2aa*, *sink2ab*, *sink2b*, *sink2ba*, and *sink2bb*, respectively. Each source generates ATM cells at a constant rate of 0.030000 cells per microsecond, or 30,000 cells/second. These sources are identical to those described in Section A.1.1.

#### VBR Sources

Figure 73 depicts four identical VBR sources, *gsource1c*, *gsource1d*, *gsource1e*, and *gsource1f*. These sources have virtual circuits established to the traffic sinks, *sink2c*, *sink2d*, *sink2e*, and *sink2f*, respectively. Each source is an On-Off source. The active period lasts 105 microseconds, and the silent period lasts 3895 microseconds. During its active period, each source generates ATM cells at an active rate of 0.353000 cells per microsecond, or 353,000 cells/second. This is identical to the VBR sources described in Section A.1.1, generating 9,266.25 cells per second on the average.

### A.2.2 Communications Links

The communications links are identical to those described in Section A.1.2 for *Bottleneck\_Jitter*.

### A.2.3 QoS Requirements of Sources

The sources in this configuration stipulate their loss tolerance in accordance with Table 3. Neither the CBR sources nor the VBR sources in this configuration impose any requirements related to jitter.

**TABLE 3. Loss Tolerance for *Bottleneck\_Loss***

Source	Losses Tolerated per Interval	Interval Length in Microseconds
gsource1a	2	10000
gsource1aa	2	10000
gsource1ab	40	10000
gsource1b	2	10000
gsource1ba	2	10000
gsource1bb	40	10000
gsource1c	40	10000
gsource1d	40	10000
gsource1e	2	10000
gsource1f	40	10000

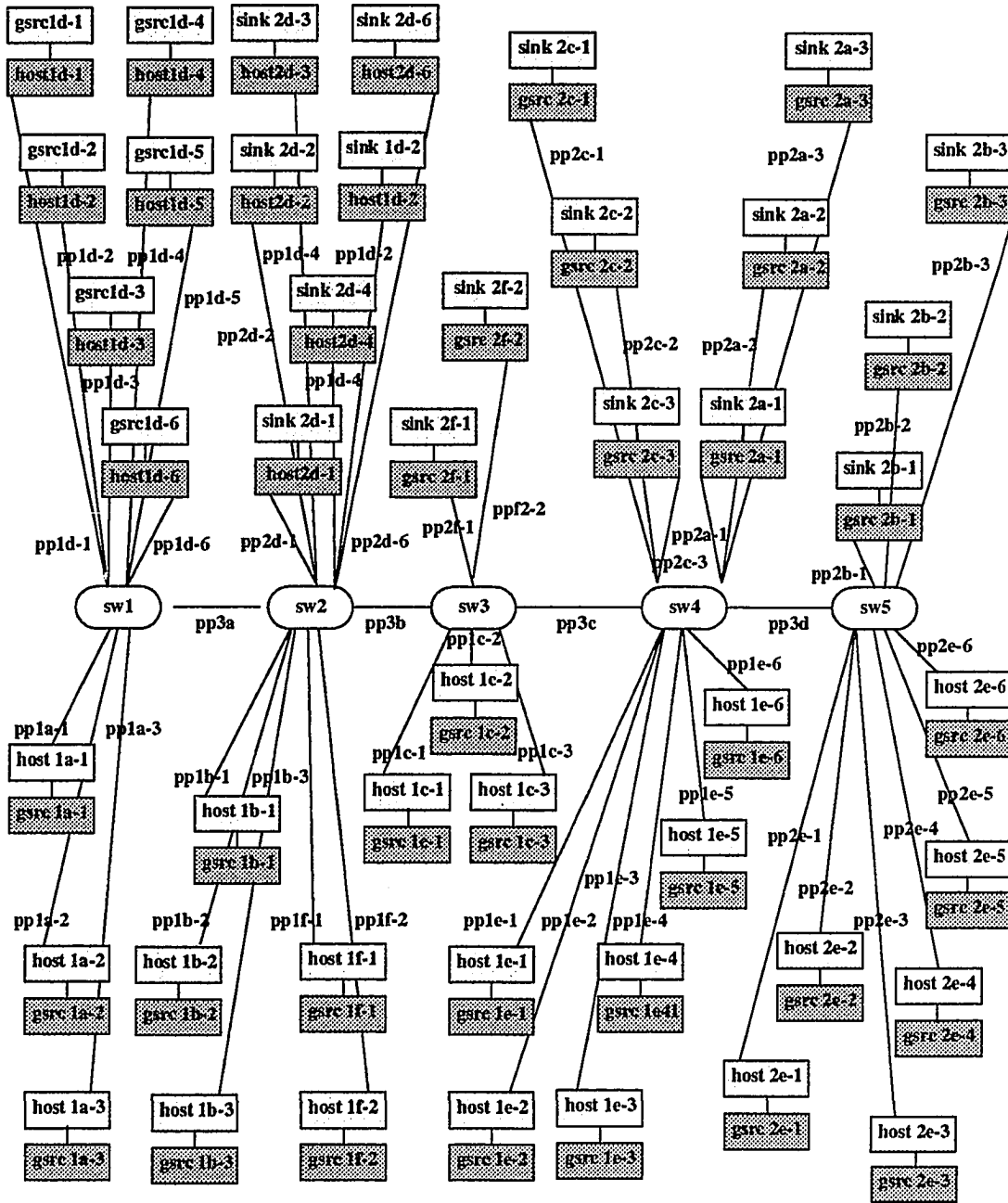
Table 3 reveals that we have selected non-zero loss tolerance for all 10 connections active in this configuration. The loss tolerance falls into two categories: 1) 40 cells per 10000 microseconds (very loss tolerant) and 2) 2 cells per 10,000 microseconds (not very loss tolerant). Since we believe that loss tolerance is not strictly related to a connection's burstiness, we have designated both loss tolerant and loss intolerant connections for both CBR and VBR classes. The selection of the measurement interval length of 10,000 microseconds was arbitrary.

### A.3 Cross\_Traffic\_HomLink

The topology and traffic patterns of *Cross\_Traffic\_HomLink* (see Figure 74) are copied from the GFC2 standard simulation configuration currently in use within the ATM Forum research community. This configuration is designed to facilitate the study of congestion

dynamics of VC's transiting a number of tandem switches. Cross-traffic at intermediate switches competes for link bandwidth and switch buffer space. We illustrate these traffic flows in Figure 28 on page 124. These traffic flows and the source traffic characteristics are designed to produce buffer overflow in the intermediate switches. This topology is used to evaluate how well different cell scheduling paradigms succeed at preserving QoS guarantees in this overload setting.

FIGURE 74. Cross\_Traffic Simulation Topology



### A.3.1 Traffic Sources

The traffic source characteristics are summarized below in Table 4. The peak rate for the VBR sources (353,000 cells/second) represents the rate if the source were to continually emit cells in *on* mode. This is clearly not the average cell emission rate as the relative durations of the *on* and *off* periods must be taken into consideration. The column *Nominal Cell Spacing* captures this in that the nominal cell interarrival time is calculated for these sources as if the VBR bursts were smoothed. Thus, if the cell spacing is divided into one second (1,000,000,000 ns), the quotient provides the average cells/second emission rate of the source. These rates are 55,875 cells/second for a cell spacing of 17,897 ns, 47,987 cells/second for a cell spacing of 20,839 ns, and 9,266 cells/second for a cell spacing of 105,090 ns. (The rates of 55,875, 17,897, and 47,987 cells/second do not appear in Table 4 and are only presented here to explain the apparent discrepancy between the *Peak Rate* and *Nominal Cell Spacing* columns in that table.)



**TABLE 4. Source Traffic Characteristics for *Cross\_Traffic\_HomLink***

Source	Sink	Peak Rate cells/sec	On Duration $\mu$ S	Off Duration $\mu$ S	Losses Tolerated per Interval	Interval Length in $\mu$ S	Nominal Cell Spacing in ns	Jitter Tolerated in ns
1a-1	2a-1	30000	n/a	n/a	5	25000	33333	20000
1a-2	2a-2	30000	n/a	n/a	5	25000	33333	20000
1a-3	2a-3	30000	n/a	n/a	5	25000	33333	20000
1b-1	2b-1	30000	n/a	n/a	test-dep.	test-dep.	33333	18000
1b-2	2b-2	30000	n/a	n/a	"	"	33333	22000
1b-3	2b-3	30000	n/a	n/a	"	"	33333	26000
1c-1	2c-1	353000	205	1395	40	10000	17897	0
1c-2	2c-2	353000	205	1395	40	10000	17897	0
1c-3	2c-3	353000	205	1395	40	10000	17897	0
1d-1	2d-1	30000	n/a	n/a	5	25000	33333	20000
1d-2	2d-2	30000	n/a	n/a	5	25000	33333	20000
1d-3	2d-3	30000	n/a	n/a	5	25000	33333	20000
1d-4	2d-4	30000	n/a	n/a	5	25000	33333	20000
1d-5	2d-5	30000	n/a	n/a	5	25000	33333	20000
1d-6	2d-6	30000	n/a	n/a	5	25000	33333	20000
1e-1	2e-1	353000	305	2500	40	10000	20389	0
1e-2	2e-2	353000	105	3895	40	10000	105090	0
1e-3	2e-3	353000	105	3895	40	10000	105090	0
1e-4	2e-4	353000	105	3895	40	10000	105090	0
1e-5	2e-5	353000	105	3895	40	10000	105090	0
1e-6	2e-6	353000	105	3895	40	10000	105090	0
1f-1	2f-1	353000	305	2295	40	10000	20389	0
1f-2	2f-2	353000	305	2295	40	10000	20389	0

**A.3.2 Communications Links**

The communications links are identical to those described in Section A.1.2 for *Bottle-neck\_Jitter*.

### **A.3.3 Quality of Service Requirements of Sources**

The QoS requirements of the sources are summarized above in Table 4.

## **A.4 Cross\_Traffic\_HetLink**

This configuration is identical to *Cross\_Traffic\_HomLink* except that the propagation delay of some of the communications links is increased. We use this configuration to study the effects of mixed-length communications links on the prediction-based cell scheduler.

### **A.4.1 Traffic Sources**

The traffic source characteristics are summarized in Table 4. These are identical to those in *Cross\_Traffic\_HomLink*.

### **A.4.2 Communications Links**

Links *pp3a* and *pp3b* are extended from 8.351 to 25.053 kilometers in length, so the link propagation delay on these links is increased from 42,465 nanoseconds to 133,057 nanoseconds, which is exactly 47 cell times (one cell takes 2,831 nanoseconds to transmit at OC-3 rates). Note that the *effective* propagation delay must additionally include the one cell time attributable to cell serialization, so the effective propagation delay is 48 cell times, or 3 basic time cycles.

All remaining communications links are identical to those described in Section A.1.2 for *Bottleneck\_Jitter*.

### **A.4.3 Quality of Service Requirements of Sources**

The QoS requirements of the sources are summarized in Table 4. These are identical to those in *Cross\_Traffic\_HomLink*.

## **A.5 Cross\_Traffic\_HetTraffic**

Except for the source traffic characteristics, *Cross\_Traffic\_HetTraffic* is identical to *Cross\_Traffic\_HetLink*. The source traffic is very different, however, from the source traffic in *Cross\_Traffic\_HetLink*. We utilize *Cross\_Traffic\_HetTraffic* to provide a heterogeneous traffic mix that more realistically reflects the traffic that would be present in an ATM network serving multiple types of CBR and VBR traffic.

### **A.5.1 Traffic Sources**

The traffic source characteristics are summarized in Table 5. The format of Table 5 requires some explanation as it includes four new columns. The first of these is the column entitled “*Enabled at Load Level*”, explained in Section A.5.5 below. The remaining three additions are entitled “*Jitter Tolerance in ns.*”, at three levels (low, med, high). These three columns are discussed in Section A.5.6.

### **A.5.2 Communications Links**

All communications links are identical to those described in Section A.4.2 for *Cross\_Traffic\_HetLink*.

### **A.5.3 Quality of Service Requirements of Sources**

The connections may be divided into two sets. The first set consists of connections that are both jitter-sensitive and loss-sensitive. The second set is comprised of connections that are only loss-sensitive. Both sets include both CBR and VBR connections. *Cross\_Traffic\_HetTraffic* is the only configuration of those tested in this research to include a jitter-sensitive VBR connection. *Cross\_Traffic\_HetTraffic* provides a wide variety of loss tolerances within the set of loss-sensitive connections. Similarly, among the connections that are both jitter and loss sensitive, there is a wide range of jitter tolerances. (We maintain this diversity of jitter tolerances within each of the three classes of *overall* jitter tolerance described below in Section A.5.6.) Further details on the QoS requirements of the sources can be found in Table 5.

### **A.5.4 Switch Buffer Sizes**

The ATM switches used in *Cross\_Traffic\_HetTraffic* are configured to have a uniform output link buffer size of 90, 130, 170 or 200 cells. The buffer size is uniform in that for a given simulation run, all switches will be configured to have one of these four values. The switches’ buffer size is varied across different simulation runs in order to observe the effect of this parameter.

### A.5.5 Traffic Loading Level

The column labeled “*Enabled at Load Level*” in Table 5 specifies at which loading level the indicated traffic source becomes active. A source is active at all levels numbered greater than or equal to the level at which it becomes active. For example, at loading level three, active sources include all those that are enabled at levels one, two and three. At loading level five, *all* the sources in Table 5 are active. This parameter is varied across different simulation runs in order to observe the affect of loading level on QoS performance.

### A.5.6 Overall Jitter Tolerance

In order to measure different schedulers’ *overall* performance relative to jitter, we use three sub-variants of *Cross\_Traffic\_HetTraffic* with a *low*, *medium*, and *high* overall jitter tolerance, respectively. The difference between the overall levels of jitter tolerance is that each jitter-sensitive connection’s tolerance is relaxed by an additional 6000 ns with each incremental increase in the overall degree of tolerance. This increase is evident in Table 5 by comparing the jitter tolerance for any jitter-sensitive connection in the three columns entitled “*Jitter Tolerance in ns (low)*”, “*Jitter Tolerance in ns (med)*” and “*Jitter Tolerance in ns (hi)*”. Note that in the presentation of the simulation data, the overall jitter tolerance is held constant for all tested combinations of traffic loading level and buffer size. For this reason, three completely separate sets of results are presented, one for the low, one for the medium, and one for the high overall jitter tolerance.

**TABLE 5. Source Traffic Characteristics for *Cross\_Traffic\_HetTraffic***

Source	Enabled at Load Level	Peak Rate cells/sec	On Duration $\mu$ S	Off Duration $\mu$ S	Losses Toler. per Interval	Interval Length in $\mu$ S	Nominal Cell Spacing in ns	Jitter Toler. in ns (low)	Jitter Toler. in ns (med)	Jitter Toler. in ns (hi)
1a-1	1	30000	n/a	n/a	2	25000	33333	0	0	0
1a-2	1	30000	n/a	n/a	40	25000	33333	0	0	0
1a-3	1	30000	n/a	n/a	100	25000	33333	0	0	0
1b-1	1	30000	n/a	n/a	0	25000	33333	6000	12000	18000
1b-2	1	30000	n/a	n/a	0	25000	33333	9000	15000	21000
1b-3	1	30000	n/a	n/a	30	2500	33333	0	0	0
1c-1	4	353000	205	1395	40	10000	17897	0	0	0
1c-2	3	353000	205	1395	40	10000	17897	0	0	0
1c-3	2	353000	205	1395	40	10000	17897	0	0	0
1d-1	1	30000	n/a	n/a	0	25000	33333	6000	12000	18000
1d-2	1	30000	n/a	n/a	0	25000	33333	6000	12000	18000
1d-3	1	30000	n/a	n/a	0	25000	33333	12000	18000	24000
1d-4	1	30000	n/a	n/a	0	25000	33333	12000	18000	24000
1d-5	1	30000	n/a	n/a	0	25000	33333	20000	26000	32000
1d-6	1	30000	n/a	n/a	0	25000	33333	20000	26000	32000
1e-1	3	353000	105	3895	2	10000	16666	0	0	0
1e-2	4	353000	105	3895	40	10000	16666	0	0	0
1e-3	1	353000	105	3895	2	10000	16666	0	0	0
1e-4	5	353000	105	3895	80	10000	16666	0	0	0
1c-5	1	353000	105	3895	2	1000	16666	0	0	0
1e-6	1	60000	105	3895	0	10000	16666	18000	24000	30000
1f-1	3	353000	305	2295	40	10000	20389	0	0	0
1f-2	2	353000	305	2295	40	10000	20389	0	0	0

# **Appendix B DETAILED SIMULATION RESULTS**

## **B.1 Tables**

In this section we provide the detailed tabular data which support the planar and surface plots presented in Chapter 7.

**TABLE 6. Jitter Violations, constant load**

<b>Number of CBR Connections</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<i>Prediction PTDM</i>						
<i>Jitter Tolerance</i>						
1 cell time	521	2777	4425	3259	3555	3849
2 cell times	0	275	1460	985	916	1283
3 cell times	0	1	236	146	125	181
4 cell times	0	0	0	0	0	0
5 cell times	0	0	0	0	0	0
6 cell times	0	0	0	0	0	0
7 cell times	0	0	0	0	0	0
8 cell times	0	0	0	0	0	0
9 cell times	0	0	0	0	0	0
10 cell times	0	0	0	0	0	0
Link Utilization (pp3)	72%	72%	72%	72%	72%	72%
<i>No Prediction NORMFIFO</i>						
<i>Jitter Tolerance</i>						
1 cell time	505	3429	5501	5569	6713	7264
2 cell times	260	1205	3140	3490	5345	5656
3 cell times	240	256	1201	2173	3914	4157
4 cell times	240	255	440	1166	2434	2795
5 cell times	239	253	261	699	1286	1812
6 cell times	231	249	258	422	824	1329
7 cell times	214	245	255	272	563	992
8 cell times	202	243	251	271	361	724
9 cell times	181	241	249	268	278	512
10 cell times	158	238	248	136	276	356
Link Utilization (pp3)	72%	72%	72%	72%	72%	72%

**TABLE 7. Jitter Violations as a Percentage of Offered Traffic, constant load**

<b>Number of CBR Connections</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<i>Prediction PTDM</i>						
<i>Jitter Tolerance</i>						
1 cell time	6	31	49	36	40	43
2 cell times	0	3	16	11	10	14
3 cell times	0	0	3	2	1	2
4 cell times	0	0	0	0	0	0
5 cell times	0	0	0	0	0	0
6 cell times	0	0	0	0	0	0
7 cell times	0	0	0	0	0	0
8 cell times	0	0	0	0	0	0
9 cell times	0	0	0	0	0	0
10 cell times	0	0	0	0	0	0
Link Utilization (pp3)	72%	72%	72%	72%	72%	72%
<i>No Prediction NORMFIFO</i>						
<i>Jitter Tolerance</i>						
1 cell time	6	38	61	62	75	81
2 cell times	3	13	35	39	58	63
3 cell times	3	3	13	24	43	46
4 cell times	3	3	5	13	27	31
5 cell times	3	3	3	8	14	20
6 cell times	3	3	3	5	9	15
7 cell times	2	3	3	3	6	11
8 cell times	2	3	3	3	4	8
9 cell times	2	3	3	3	3	6
10 cell times	2	3	3	3	3	4
Link Utilization (pp3)	72%	72%	72%	72%	72%	72%



**TABLE 8. Jitter Violations, increasing load**

<b>Number of CBR Connections</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<i>Prediction PTDM</i>						
<i>Jitter Tolerance</i>						
1 cell time	93	712	1771	1712	2305	3257
2 cell times	1	73	596	492	572	1035
3 cell times	1	1	88	76	88	148
4 cell times	0	0	0	0	3	2
5 cell times	0	0	0	0	1	1
6 cell times	0	0	0	0	1	1
7 cell times	0	0	0	0	1	1
8 cell times	0	0	0	0	0	0
9 cell times	0	0	0	0	0	0
10 cell times	0	0	0	0	0	0
Link Utilization (pp3)	31%	40%	48%	56%	64%	72%
<i>No Prediction NORMFIFO</i>						
<i>Jitter Tolerance</i>						
1 cell time	316	1731	3007	3648	5408	7301
2 cell times	292	1134	2102	2322	3995	5731
3 cell times	291	635	1254	1769	2913	4295
4 cell times	287	630	1060	1575	2212	2937
5 cell times	285	627	1049	1539	1820	1957
6 cell times	284	624	1041	1427	1482	1428
7 cell times	282	622	950	1150	968	1063
8 cell times	280	616	774	620	687	780
9 cell times	278	503	414	334	462	550
10 cell times	275	211	224	224	313	389
Link Utilization (pp3)	29%	37%	45%	54%	62%	72%

**TABLE 9. Jitter Violations as a Percentage of Offered Traffic, increasing load**

<b>Number of CBR Connections</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<b><i>Prediction PTDM</i></b>						
<b><i>Jitter Tolerance</i></b>						
1 cell time	6	24	40	29	31	37
2 cell times	0	2	13	8	8	12
3 cell times	0	0	2	1	1	2
4 cell times	0	0	0	0	0	0
5 cell times	0	0	0	0	0	0
6 cell times	0	0	0	0	0	0
7 cell times	0	0	0	0	0	0
8 cell times	0	0	0	0	0	0
9 cell times	0	0	0	0	0	0
10 cell times	0	0	0	0	0	0
Link Utilization (pp3)	31%	40%	48%	56%	64%	72%
<b><i>No Prediction NORMFIFO</i></b>						
<b><i>Jitter Tolerance</i></b>						
1 cell time	21	58	67	61	72	81
2 cell times	20	38	47	39	52	64
3 cell times	19	21	28	29	39	48
4 cell times	19	21	24	26	30	33
5 cell times	19	21	23	26	24	22
6 cell times	19	21	23	24	20	16
7 cell times	19	21	21	19	13	12
8 cell times	19	21	17	10	9	9
9 cell times	19	17	9	6	6	6
10 cell times	18	7	5	7	4	4
Link Utilization (pp3)	29%	37%	45%	54%	62%	72%

**TABLE 10. Loss Violations, Local QoS Knowledge, Bottleneck Loss**

<b>Length of Measurement Interval (<math>\mu</math>S)</b>	<b>5000</b>	<b>10000</b>	<b>15000</b>	<b>20000</b>	<b>25000</b>
<b><i>Prediction w/ Displacement (PFIFO_D_NG)</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	10	23	30	36	52
2 cells	4	15	28	38	56
4 cells	0	3	8	24	44
8 cells	0	0	0	1	21
<b><i>Prediction w/o Displacement (PFIFO)</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	69	73	77	79	81
2 cells	51	59	67	71	75
4 cells	15	31	47	55	66
8 cells	0	0	11	14	35
<b><i>No Prediction (NORMFIFO)</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	80	86	90	94	94
2 cells	58	70	78	86	86
4 cells	12	38	54	70	77
8 cells	0	1	11	30	53

**TABLE 11. Loss Violations, Global QoS Knowledge, *Cross\_Traffic\_HomLink*, small buffer**

<b>Length of Measurement Interval (<math>\mu</math>s)</b>	<b>5000</b>	<b>10000</b>	<b>15000</b>	<b>20000</b>	<b>25000</b>
<b><i>Prediction w/ Global Knowledge with Displacement (PFIFO_D) Loss Tolerance per 5000 <math>\mu</math>s</i></b>					
1 cell	70	66	62	63	59
2 cells	76	71	69	67	65
4 cells	87	85	79	68	66
8 cells	113	89	84	73	66
<b><i>Prediction w/o Global Knowledge with Displacement (PFIFO_D_NG) Loss Tolerance per 5000 <math>\mu</math>s</i></b>					
1 cell	176	173	169	163	168
2 cells	198	196	187	183	188
4 cells	226	225	221	212	209
8 cells	256	272	276	290	277
<b><i>No Prediction (NORMFIFO) Loss Tolerance per 5000 <math>\mu</math>s</i></b>					
1 cell	531	531	532	533	531
2 cells	521	521	523	525	521
4 cells	501	501	505	509	501
8 cells	461	461	469	477	497

**TABLE 12. Loss Violations, Global QoS Knowledge, *Cross\_Traffic\_Hetlink*, small buffer**

<b>Length of Measurement Interval (<math>\mu s</math>)</b>	<b>5000</b>	<b>10000</b>	<b>15000</b>	<b>20000</b>	<b>25000</b>
<b><i>Prediction w/ Global Knowledge with Displacement (PFIFO_D) Loss Tolerance per 5000 <math>\mu s</math></i></b>					
1 cell	66	64	57	56	54
2 cells	72	69	62	61	58
4 cells	88	76	72	62	60
8 cells	110	82	85	68	61
<b><i>Prediction w/o Global Knowledge with Displacement (PFIFO_D_NG) Loss Tolerance per 5000 <math>\mu s</math></i></b>					
1 cell	192	184	179	178	179
2 cells	211	201	200	192	196
4 cells	241	232	226	225	221
8 cells	268	289	288	304	296
<b><i>No Prediction (NORMFIFO) Loss Tolerance per 5000 <math>\mu s</math></i></b>					
1 cell	527	527	528	529	527
2 cells	517	517	519	521	517
4 cells	497	497	501	505	497
8 cells	458	457	465	473	491

**TABLE 13. Loss Violations, 90 cell buffer, PTDM\_D vs. others**

<b>Length of Measurement Interval (<math>\mu</math>s)</b>	<b>5000</b>	<b>10000</b>	<b>15000</b>	<b>20000</b>	<b>25000</b>
<b><i>PTDM_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	419	410	408	431	418
2 cells	424	433	406	415	405
4 cells	438	418	405	396	400
8 cells	398	401	394	405	388
<b><i>PFIFO_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	66	64	57	56	54
2 cells	72	69	62	61	58
4 cells	88	76	72	62	60
8 cells	110	82	85	68	61
<b><i>PTDM Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	1007	1007	1008	1009	1007
2 cells	997	997	999	1001	997
4 cells	977	977	981	985	977
8 cells	937	937	945	953	977
<b><i>NORMFIFO Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	527	527	528	529	527
2 cells	517	517	519	521	517
4 cells	497	497	501	505	497
8 cells	458	457	465	473	491

**TABLE 14. Loss Violations, 9999 cell buffer, PTDM\_D vs. others**

<b>Length of Measurement Interval (<math>\mu</math>s)</b>	<b>5000</b>	<b>10000</b>	<b>15000</b>	<b>20000</b>	<b>25000</b>
<b><i>PTDM_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	0	0	0	0	0
2 cells	0	0	0	0	0
4 cells	0	0	0	0	0
8 cells	0	0	0	0	0
<b><i>PFIFO_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	0	0	0	0	0
2 cells	0	0	0	0	0
4 cells	0	0	0	0	0
8 cells	0	0	0	0	0
<b><i>PTDM Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	163	161	162	163	161
2 cells	155	151	153	155	151
4 cells	139	131	135	139	131
8 cells	108	95	99	104	120
<b><i>NORMFIFO Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	0	0	0	0	0
2 cells	0	0	0	0	0
4 cells	0	0	0	0	0
8 cells	0	0	0	0	0

**TABLE 15. Jitter Violations, 90 cell buffer, *PTDM\_D* vs. others**

<b>Length of Measurement Interval (<math>\mu</math>s)</b>	<b>5000</b>	<b>10000</b>	<b>15000</b>	<b>20000</b>	<b>25000</b>
<b><i>PTDM_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	98	93	95	94	95
2 cells	108	98	94	96	98
4 cells	110	102	95	91	93
8 cells	112	96	102	104	90
<b><i>PFIFO_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	1430	1435	1437	1439	1433
2 cells	1423	1427	1434	1429	1429
4 cells	1408	1411	1421	1408	1412
8 cells	1380	1376	1395	1405	1420
<b><i>PTDM Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	229	229	229	229	229
2 cells	229	229	229	229	229
4 cells	229	229	229	229	229
8 cells	229	229	229	229	229
<b><i>NORMFIFO Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	1247	1247	1247	1247	1247
2 cells	1247	1247	1247	1247	1247
4 cells	1247	1247	1247	1247	1247
8 cells	1247	1247	1247	1247	1247



**TABLE 16. Jitter Violations, 9999 cell buffer, *PTDM\_D* vs. others**

<b>Length of Measurement Interval (<math>\mu</math>S)</b>	<b>5000</b>	<b>10000</b>	<b>15000</b>	<b>20000</b>	<b>25000</b>
<b><i>PTDM_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	0	0	0	0	0
2 cells	0	0	0	0	0
4 cells	0	0	0	0	0
8 cells	0	0	6	12	16
<b><i>PFIFO_D Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	1538	1538	1538	1538	1538
2 cells	1538	1538	1538	1538	1538
4 cells	1538	1538	1538	1538	1538
8 cells	1538	1538	1538	1538	1538
<b><i>PTDM Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	53	53	53	53	53
2 cells	53	53	53	53	53
4 cells	53	53	53	53	53
8 cells	53	53	53	53	53
<b><i>NORMFIFO Results</i></b>					
<b><i>Loss Tolerance per meas. int.</i></b>					
1 cell	1661	1661	1661	1661	1661
2 cells	1661	1661	1661	1661	1661
4 cells	1661	1661	1661	1661	1661
8 cells	1661	1661	1661	1661	1661

**TABLE 17. Low Jitter Tolerance Overall QoS Violations**

Loading Level	1	2	3	4	5
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	2783	5376	5669
130 cells	0	1	1159	2122	2512
170 cells	0	1	140	192	289
200 cells	0	1	3	14	65
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	1684	1592	8358	12955	13532
130 cells	1684	1592	6433	7637	7745
170 cells	1684	1592	1562	1466	1580
200 cells	1684	1592	1562	1532	1504
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	3235	4337	6267	7977	8275
130 cells	3235	4337	5432	6084	6275
170 cells	3235	4337	5089	5507	5454
200 cells	3235	4337	5089	5506	5497

**TABLE 18. Medium Jitter Tolerance Overall QoS Violations**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	2782	5371	5659
130 cells	0	0	1158	2113	2516
170 cells	0	0	138	179	280
200 cells	0	0	0	7	19
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	197	191	6994	11597	12245
130 cells	197	191	5094	6309	6398
170 cells	197	191	203	190	228
200 cells	197	191	203	183	199
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	1224	2083	3987	5765	6140
130 cells	1224	2083	3073	3841	4070
170 cells	1224	2083	2846	3279	3252
200 cells	1224	2083	2846	3220	3253

**TABLE 19. High Jitter Tolerance Overall QoS Violations**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	2782	5369	5654
130 cells	0	0	1157	2114	2529
170 cells	0	0	138	178	295
200 cells	0	0	0	5	10
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	6772	11430	12040
130 cells	0	0	4907	6112	6197
170 cells	0	0	0	6	25
200 cells	0	0	0	2	0
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	425	722	2509	4255	4580
130 cells	425	722	1545	2138	2459
170 cells	425	722	1220	1503	1535
200 cells	425	722	1220	1461	1530

**TABLE 20. Low Jitter Tolerance Jitter Violations Only**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	1	9	22
130 cells	0	1	1	7	17
170 cells	0	1	2	8	32
200 cells	0	1	3	8	20
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	1684	1592	2114	2041	2008
130 cells	1684	1592	1881	1869	1849
170 cells	1684	1592	1562	1460	1555
200 cells	1684	1592	1562	1530	1504
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	3235	4337	4769	4828	4827
130 cells	3235	4337	4955	5294	5182
170 cells	3235	4337	5089	5479	5412
200 cells	3235	4337	5089	5505	5494

**TABLE 21. Medium Jitter Tolerance Jitter Violations Only**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	0	3	10
130 cells	0	0	0	1	8
170 cells	0	0	0	3	14
200 cells	0	0	0	2	8
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	197	191	750	683	721
130 cells	197	191	542	541	502
170 cells	197	191	203	184	203
200 cells	197	191	203	181	199
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	1224	2083	2489	2616	2692
130 cells	1224	2083	2596	3051	2977
170 cells	1224	2083	2846	3251	3210
200 cells	1224	2083	2846	3218	3250

**TABLE 22. High Jitter Tolerance Jitter Violations Only**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	0	1	8
130 cells	0	0	0	2	4
170 cells	0	0	0	4	7
200 cells	0	0	0	4	5
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	528	516	516
130 cells	0	0	355	344	301
170 cells	0	0	0	0	0
200 cells	0	0	0	0	0
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	425	722	1011	1106	1132
130 cells	425	722	1068	1348	1366
170 cells	425	722	1220	1475	1493
200 cells	425	722	1220	1459	1527

**TABLE 23. Low Jitter Tolerance Loss Violations Only**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	2782	5367	5647
130 cells	0	0	1158	2115	2495
170 cells	0	0	138	184	257
200 cells	0	0	0	6	45
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	6244	10914	11524
130 cells	0	0	4552	5768	5896
170 cells	0	0	0	6	25
200 cells	0	0	0	2	0
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	1498	3149	3448
130 cells	0	0	477	790	1093
170 cells	0	0	0	28	42
200 cells	0	0	0	2	3



**TABLE 24. Medium Jitter Tolerance Loss Violations Only**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<i><b>PTDM_D</b></i>					
<i><b>Buffer Size</b></i>					
90 cells	0	0	2782	5368	5649
130 cells	0	0	1158	2112	2505
170 cells	0	0	138	176	266
200 cells	0	0	0	5	11
<i><b>HOLDISP</b></i>					
<i><b>Buffer Size</b></i>					
90 cells	0	0	6244	10914	11524
130 cells	0	0	4552	5768	5896
170 cells	0	0	0	6	25
200 cells	0	0	0	2	0
<i><b>NORMFIFO</b></i>					
<i><b>Buffer Size</b></i>					
90 cells	0	0	1498	3149	3448
130 cells	0	0	477	790	1093
170 cells	0	0	0	28	42
200 cells	0	0	0	2	3

**TABLE 25. High Jitter Tolerance Loss Violations Only**

<b>Loading Level</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b><i>PTDM_D</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	2782	5368	5646
130 cells	0	0	1157	2112	2525
170 cells	0	0	138	174	288
200 cells	0	0	0	1	5
<b><i>HOLDISP</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	6244	10914	11524
130 cells	0	0	4562	5768	5796
170 cells	0	0	0	6	25
200 cells	0	0	0	2	0
<b><i>NORMFIFO</i></b>					
<b><i>Buffer Size</i></b>					
90 cells	0	0	1498	3149	3448
130 cells	0	0	477	790	1093
170 cells	0	0	0	28	42
200 cells	0	0	0	2	3

**TABLE 26. Link Utilization (%), Comparison of 4 policies, small buffer switches**

Link Name	pp3a	pp3b	pp3c	pp3d	pp2b-1	pp2b-2	pp2b-3
<i>PTDM_D Results</i>							
downstream	82	67	79	40	14	14	14
upstream	6	6	6	6	6	6	6
<i>PFIFO_D Results</i>							
downstream	82	72	88	50	15	15	14
upstream	6	6	6	6	6	6	6
<i>PTDM Results</i>							
downstream	82	67	77	40	13	13	13
upstream	6	6	6	6	6	6	6
<i>NORMFIFO Results</i>							
downstream	76	66	81	43	8	8	8
upstream	0	0	0	0	0	0	0

**TABLE 27. Link Utilization (%), Comparison of 4 policies, large buffer switches**

Link Name	pp3a	pp3b	pp3c	pp3d	pp2b-1	pp2b-2	pp2b-3
<i>PTDM_D Results</i>							
downstream	82	80	96	56	15	15	15
upstream	6	6	6	6	6	6	6
<i>PFIFO_D Results</i>							
downstream	82	80	97	57	15	15	15
upstream	6	6	6	6	6	6	6
<i>PTDM Results</i>							
downstream	82	81	96	55	14	15	15
upstream	6	6	6	6	6	6	6
<i>NORMFIFO Results</i>							
downstream	76	75	91	51	8	8	8
upstream	0	0	0	0	0	0	0

**TABLE 28. Throughput, Comparison of 4 policies, small buffer switches**

Source/Sink Name	sink 1b-1	sink 1b-2	sink 1b-3
<i>PTDM_D Results</i>			
cell received	1353	1353	1382
per second throughput	27060	27060	27640
<i>PFIFO_D Results</i>			
cells received	1478	1478	1446
per second throughput	29560	29560	28920
<i>PTDM Results</i>			
cells received	1214	1214	1216
per second throughput	24280	24280	24320
<i>NORMFIFO Results</i>			
cells received	1373	1373	1364
per second throughput	27460	27460	27280

**TABLE 29. Throughput, Comparison of 4 policies, large buffer switches**

Source/Sink Name	sink1b-1	sink 1b-2	sink1b-3
<i>PTDM_D Results</i>			
cell received	1500	1500	1500
per second throughput	30000	30000	30000
<i>PFIFO_D Results</i>			
cells received	1505	1505	1504
per second throughput	30100	30100	30080
<i>PTDM Results</i>			
cells received	1408	1408	1500
per second throughput	28160	28160	30000
<i>NORMFIFO Results</i>			
cells received	1496	1496	1495
per second throughput	29920	29920	29900

**TABLE 30. Raw Cell Loss vs. Cell Loss Violations**

	Raw Cell Loss 1b-2	Raw Cell Loss 1b-2	Raw Cell Loss 1b-3	Total Raw Cell Loss	Total Loss Violations	Violations as % of raw loss
<i>PTDM_D</i>	192	147	155	494	406	82%
<i>PFIFO_D</i>	23	20	72	115	62	53%
<i>PTDM</i>	358	361	358	1077	999	93%
<i>NORMFIFO</i>	199	195	203	597	519	86%

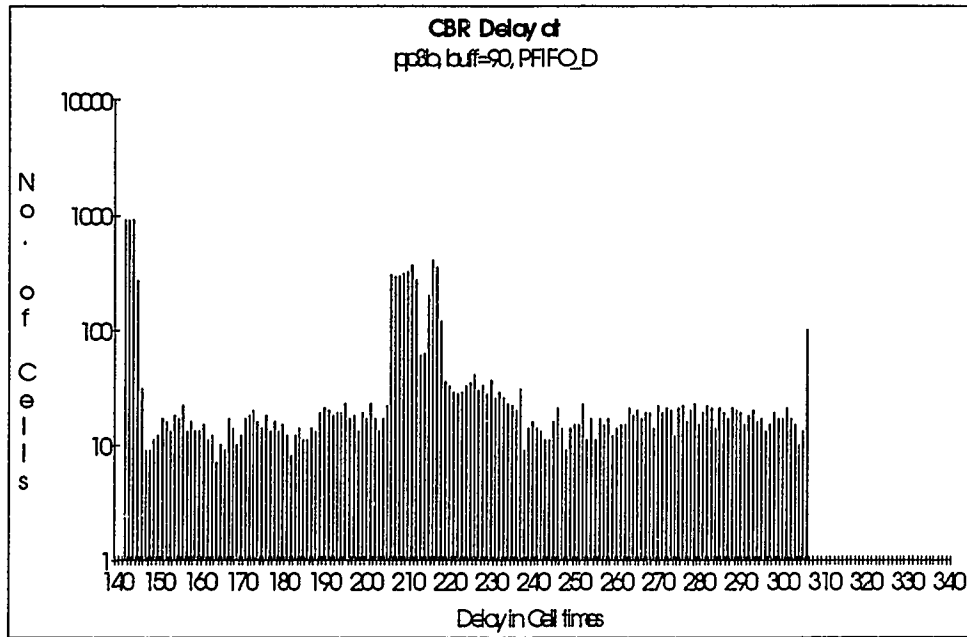
**TABLE 31. Components of Additional Delay Due to Prediction-Based Scheduling (in cell times)**

PredictHorizon	AAL $B^W$	AAL $\beta^D$	gnra $B^B$	gnrb $B^B$	gnrb Cell Spacing Delay	Total Additional Delay
1	16	16	16	16	7	71
2	32	16	16	16	7	87
3	48	16	16	16	7	103

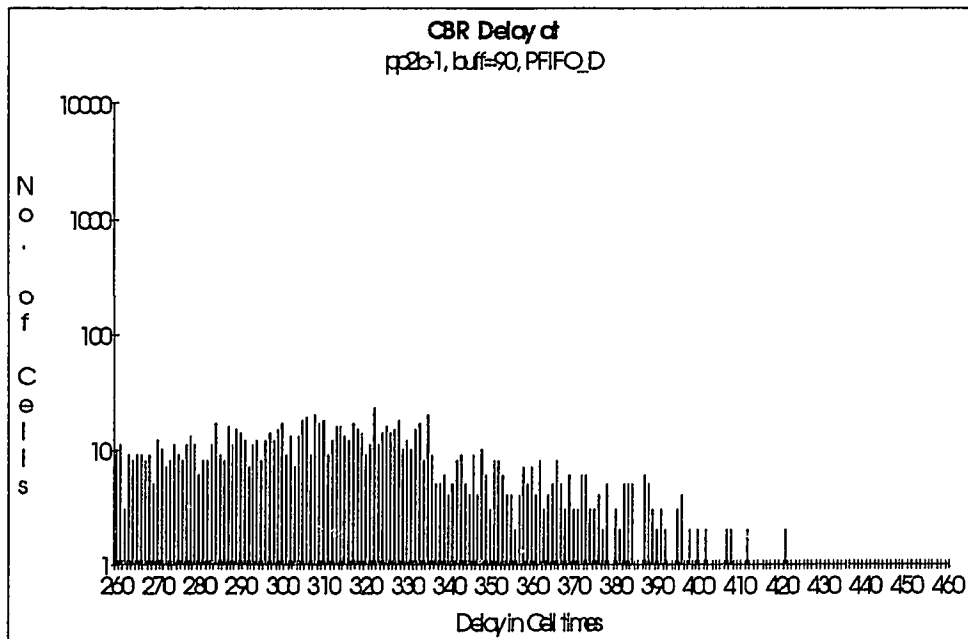
## **B.2 Delay Distributions**

In this section we supplement the delay distribution data that was provided in Chapter 7. The delay distributions provided here were all obtained from the tests described in Section 7.8. The results presented here were discussed in Section 7.12.

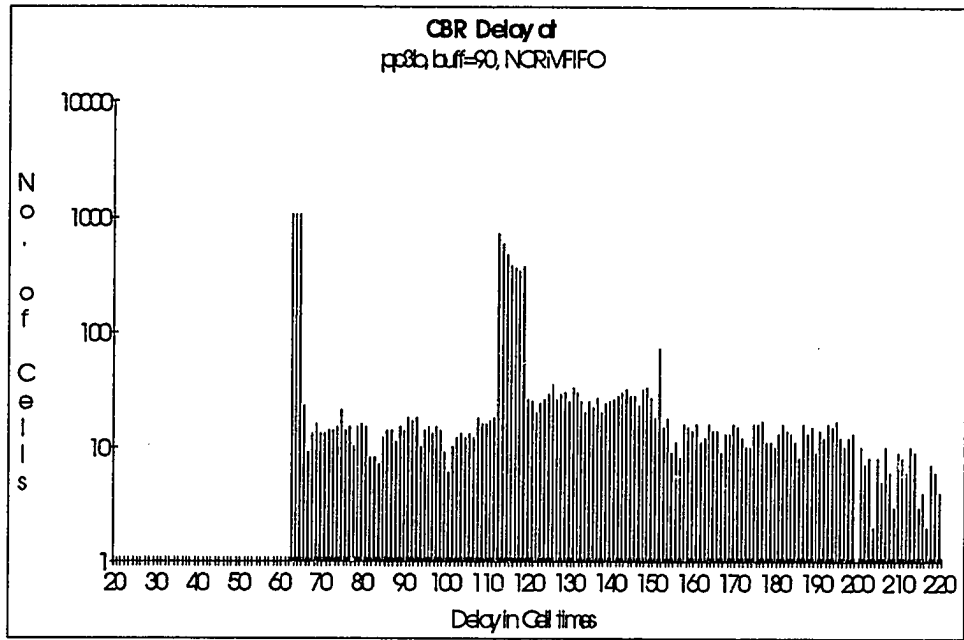
**FIGURE 75. Time-in-system, pp3b, small buffer switch (PFIFO\_D)**



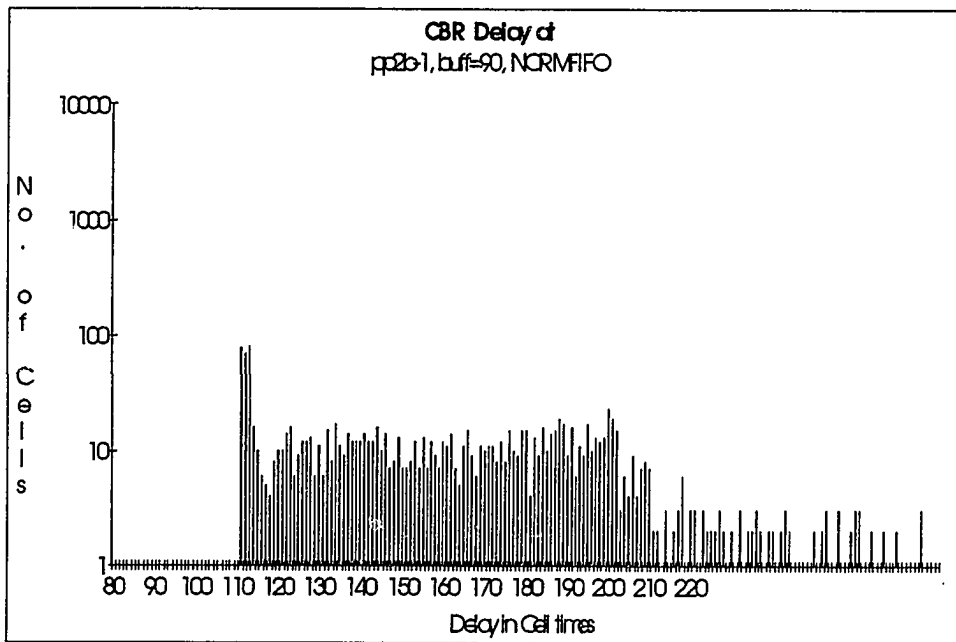
**FIGURE 76. Time-in-system, pp2b-1, small buffer switch (PFIFO\_D)**



**FIGURE 77. Time-in-system, pp3b, small buffer switch (NORMFIFO)**

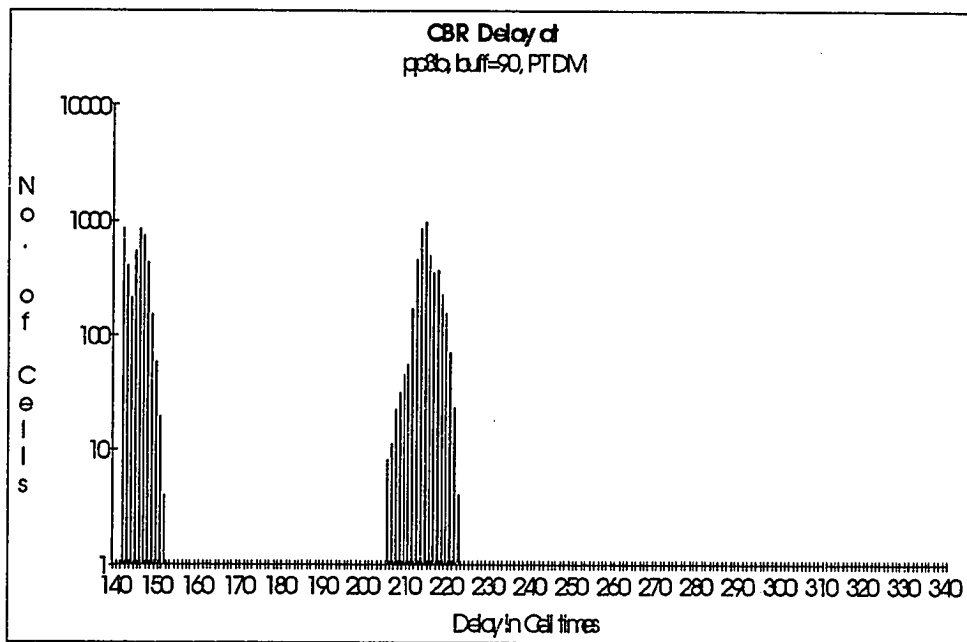


**FIGURE 78. Time-in-system, pp2b-1, small buffer switch (NORMFIFO)**

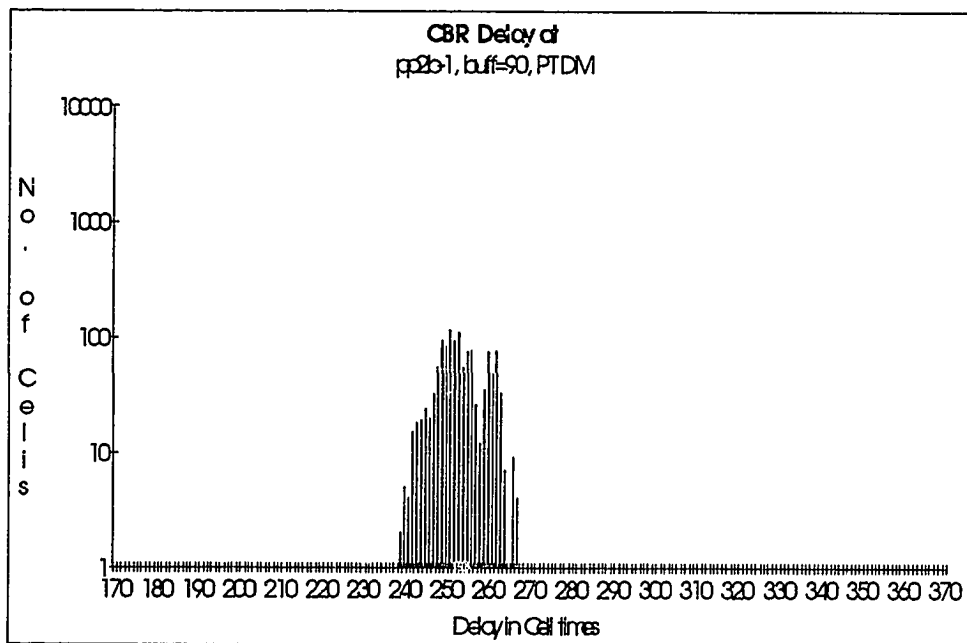




**FIGURE 79. Time-in-system, pp3b, small buffer switch (PTDM)**



**FIGURE 80. Time-in-system, pp2b-1, small buffer switch (PTDM)**



## **Appendix C**

# **GLOSSARY OF ACRONYMS**

<b>AAL</b>	Asynchronous Transfer Mode Adaptation Layer
<b>ABR</b>	Available Bit Rate
<b>ATM</b>	Asynchronous Transfer Mode
<b>B-ISDN</b>	Broadband Integrated Services Digital Network
<b>CBR</b>	Constant Bit Rate
<b>CCITT</b>	International Consultative Committee for Telecommunications and Telegraphy
<b>CODEC</b>	Coder-Decoder
<b>CQSP</b>	Complete Quality of Service Scheduling Problem
<b>ETA</b>	Expected Time of Arrival
<b>FDDI</b>	Fiber Distributed Data Interface
<b>FIFO</b>	First In First Out
<b>FTSA</b>	Future Time Slot Assignment
<b>HOL</b>	Head of Line
<b>HOLDISP</b>	Head of Line with Displacement
<b>ITU</b>	International Telecommunications Union
<b>MARS</b>	Magnet II Algorithms for Real-time Scheduling
<b>MPEG</b>	Motion Picture Experts Group
<b>NAL</b>	Network Adaptation Layer
<b>NNI</b>	Network-Network Interface
<b>NORMFIFO</b>	Normal First In First Out queueing
<b>PBS</b>	Partial Buffer Sharing
<b>PFIFO_D</b>	Predictive First In First Out with Displacement
<b>PFIFO_D_NG</b>	Predictive First In First Out with Displacement, No Global knowledge

<b>PTDM</b>	<b>Pseudo-Time Division Multiplexing</b>
<b>PTDM_D</b>	<b>Pseudo-Time Division Multiplexing with Displacement</b>
<b>QoS</b>	<b>Quality of Service</b>
<b>SAR</b>	<b>Segmentation and Reassembly</b>
<b>SONET</b>	<b>Synchronous Optical Network</b>
<b>SPS</b>	<b>Static Priority Scheduling</b>
<b>SQSP</b>	<b>Simple Quality of Service Scheduling Problem</b>
<b>SWI</b>	<b>Scheduling Within Intervals</b>
<b>TCP</b>	<b>Transmission Control Protocol</b>
<b>TDM</b>	<b>Time Division Multiplexing</b>
<b>UBR</b>	<b>Unspecified Bit Rate</b>
<b>UNI</b>	<b>User Network Interface</b>
<b>VBR</b>	<b>Variable Bit Rate</b>
<b>VC</b>	<b>Virtual Circuit</b>
<b>VCI</b>	<b>Virtual Circuit Indicator</b>
<b>VciCTL</b>	<b>Virtual Circuit Control Block</b>
<b>VPI</b>	<b>Virtual Path Indicator</b>