Edge-based quality-of-service architecture

Lim, Teck Meng

2005

Lim, T. M. (2005). Edge-based quality-of-service architecture. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2358>

<https://doi.org/10.32657/10356/2358>

Nanyang Technological University

Downloaded on 26 Apr 2023 07:55:42 SGT

a357863
✓

An Edge-based Quality-of-Service Architecture

Lim Teck Meng

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2005

TK
5105.55
L732
2005

Acknowledgements

I would like to express my profound appreciation to Associate Professor Lee Bu Sung, Francis, my thesis advisor, for giving me good guidance throughout my graduate studies. His excellent feedback and strong arguments have contributed greatly to this dissertation. I am also extremely grateful to Associate Professor Yeo Chai Kiat, my thesis co-advisor for her consistent guidance, advice and help. I appreciate the many brainstorming and discussion sessions we had together. It is their continuous advice and feedback that shaped my research and presentation skills. I hope and look forward to continuous collaboration with Prof Lee and Prof Yeo in the future.

Thanks to David Ng for his feedback and comments that helped to improve the quality of this dissertation. Thanks to Juki Wirawan Tantra for his help with my countless questions. Special thanks to Bee Keow for accommodating her busy schedule to proofread this thesis, which helped significantly in improving the presentation of this dissertation.

I am also grateful to all my friends and colleagues with whom I spent my time as a graduate student at the Centre of Multimedia and Network Technology and Digital Signal Processing Laboratory in Nanyang Technological University.

Finally, I would like to express my earnest gratitude to my parents and my sister for their love and support. Thanks to my parents for their countless sacrifices to raise me and provide me with the best possible education.

Contents

Title Page	iii
Acknowledgments	v
Table of Contents	vii
List of Figures	xi
List of Tables	xv
Acronyms	xvii
Abstract	xxi
Citations to Previously Published Work	xxiii
Dedication	xxv
1 Contribution	1
1.1 Introduction	1
1.2 Main Contribution	3
1.3 Organisation	4
2 Background	7
2.1 Internet Quality-of-Service Overview	7
2.2 Quality-of-Service Architectures	8
2.2.1 Integrated Services	9
Guaranteed Service	11
Controlled-Load Service	11
2.2.2 Stateless Core Architecture	12
The Dynamic Packet State Techniques	13
2.2.3 Differentiated Services	13
Per-hop Behaviour - Expedited Forwarding	16
Per-hop Behaviour - Assured Forwarding	16
2.2.4 Virtual Time Reference System	18
2.3 Quality-of-Service Control Mechanisms	19
2.3.1 Classifiers	19
Type-of-Service Classifier	20
Multifield Classifier	20
2.3.2 Shapers and Policers	20
Leaky Bucket Filter	21
Token Bucket Filter	21
Dual Token Bucket Filter	22

2.3.3	Packet Scheduling Algorithms	23
	Weighted Round Robin and Deficit Round Robin	24
	Weighted Fair Queuing	24
	Virtual Clock	26
	Earliest Deadline First	26
	Comparison of Packet Scheduling Algorithms	27
2.4	Summary	28
3	Edge-based QoS Architecture	31
3.1	Edge-based QoS Architecture	31
3.1.1	The Edge-based Virtual Time Reference System	33
3.1.2	Multi-service Scheduling	35
3.1.3	Signalling and Centralised Reservation	36
3.2	Simulation	37
3.2.1	Simulation Settings	37
3.2.2	Simulation Results	41
	Edge-to-edge delay	42
	Jitter	45
	Throughput	45
3.3	Summary	46
4	Edge-based Virtual Time Reference System	49
4.1	Introduction	50
4.2	EVTRS Architecture	51
4.3	Extraction of EVTRS Information	56
4.3.1	Computing the Departure Time of packet for Core Routers	56
4.3.2	Relative Virtual Clock	57
	Packet Filtering for a Flow	57
	Arrival Rate Estimation for a Queue	58
4.4	Mapping flows to EVTRS	58
4.4.1	Mapping Time-Critical flow	58
4.4.2	Mapping Jitter-Sensitive flow	59
4.4.3	Mapping Rate-Based flow	60
4.4.4	Ensuring Fine Granular Packet Loss Rate and Packet Loss Size	61
4.5	Analytical Results	62
4.5.1	Inter-packet Departure Time	62
4.5.2	EVTRS updating algorithm	63
4.6	Numerical Examples	66
	Numerical example for a non-ideal case	67
	Numerical example for split node case	67
4.7	Discussion	68
4.7.1	Packet Lost at Medium Access Control Layer	68
4.7.2	Scalability of Core Routers	69
4.7.3	Complexity of Packet Dropping Algorithm	70
4.8	Summary	70

5	Weighted Deficit Earliest Departure First Scheduler	73
5.1	Introduction	74
5.2	Weighted Deficit Earliest Departure First Scheduler	75
5.2.1	Forwarding Head-Of-Line Packet with the Earliest Deadline	76
	Packet Arrival	80
	Packet Departure	80
5.2.2	Quantum computation and QoS support	84
5.2.3	QoS Forwarding Rule	86
5.3	Analytical Results	89
5.3.1	Fairness of WDEDF Scheduler	89
5.3.2	Delay bound comparison with DRR scheduler	93
5.4	Simulation Results	96
5.4.1	Fair and Weighted Bandwidth Sharing and Flow Protection	97
5.4.2	Performances of WDEDF scheduler with CBR and VBR Traffic.	99
	Packet Delay	99
	Deadline Misses	103
5.4.3	Delay bound of WDEDF scheduler as a Latency-Rate scheduler	103
5.4.4	Performance of WDEDF scheduler for Regulated Traffic with QoS	105
	Delivering HOL packets with the earliest deadline	106
	Delivery of Time-Critical Flow with low rate and intolerable delay	108
	Delivery of Time-Critical Flows and Rate-Based Flows	109
	Delivery of Jitter-Sensitive Flows	112
	Bandwidth guarantees on WDEDF	115
	Effect of Different Quantum Allocation Methods	115
5.5	Related Work	117
5.6	Summary	119
6	Path and Oracle Discovery Protocol	121
6.1	Introduction	122
6.2	PODP Overview	123
6.3	Operation of PODP	127
6.3.1	Operation of Option 1	130
6.3.2	Operation of other options	131
6.4	Discovery Message Essentials	132
6.4.1	Raw Socket / UDP Socket	132
6.4.2	IP Router Alert Option	133
6.4.3	Maximum Transmission Unit	133
6.4.4	Refreshing	134
6.4.5	Error Detection and Recovery of PODP	135
6.5	PODP Prototype	135
6.6	Evaluating PODP	137
6.6.1	Storage Requirements	137
6.6.2	Traffic generated at each Router	139
6.6.3	Discussion on the Performance of PODP	140
	Computation cost for processing <i>Discovery Message</i> on the fly	140
	Trade-off for refreshing <i>Discovery Message</i>	141
	Scalability of Centralised Management Control	141

PODP on EVTRS	141
6.7 Related Work	142
6.8 Summary	144
7 Conclusion and Future Work	145
7.1 Contribution	145
7.2 Future Work	148
7.2.1 Scalability and Robustness	148
7.2.2 Multicast	149
7.2.3 Granular packet dropping and relative packet dropping	149
7.2.4 Effect of EVTRS with flow control protocol	149
7.2.5 Inter-operation with other QoS architectures	150
7.2.6 Other QoS requirements	151
7.2.7 Bandwidth Broker	151
7.3 Final Remarks	152
Bibliography	153
A Performance bound of WDEDF when used as a LR scheduler	159
B Specifications for PODP Messages	163
B.1 <i>Discovery Message</i>	163
B.2 <i>Reporting Message</i>	165
B.3 <i>Notifying Message</i>	166
B.4 <i>Recovery Message</i>	167

List of Figures

2.1	An Integrated Services Model [1].	9
2.2	A SCORE Domain which comprises edge routers and core routers	12
2.3	A Differentiated Services Domain Model [3].	14
2.4	Reclaiming of IP header fields for DSCP field	20
2.5	A Leaky Bucket Filter.	21
2.6	A Token Bucket Filter.	21
2.7	A Dual Token Bucket Filter.	22
2.8	Arrival Curve of a Dual Token Bucket Filter with parameters (σ, r, ρ, M) . . .	22
3.1	Edge-based QoS Domain Model that consists of two ingress routers. Ingress 1 router contains TC, JS, RB and BE service classes in which each service class contains two flows; and each core router ensures QoS for each edge router's service classes.	33
3.2	An illustration showing the flow of PODP messages in a single SCORE Domain.	37
3.3	A parking-lot network	38
3.4	A chain network	38
3.5	Edge-to-edge worst-case delay for TC and JS flows of Flow Set A in parking-lot network	43
3.6	Edge-to-edge worst-case delay for TC and JS flows of Flow Set A in chain network	43
3.7	Normalised Inter-packet Arrival Delay for JS flows of Flow Set A at node E1, plotted with 99% confidence interval in parking-lot network	44
3.8	Normalised Inter-packet Arrival Delay for JS flows of Flow Set A at node E1, plotted with 99% confidence interval in chain network	44
4.1	Packet trains in Edge and Core routers with multiple queues	51
4.2	Timing relations between per-flow, per-queue and per-link <i>packet trains</i> in which $\widehat{T}_D^{k-1} = \widehat{T}_D^{m-1}$ and $\widehat{T}_D^k = \widehat{T}_D^{n-1}$, where $\widehat{T}_D$ is the expected departure time, $\widetilde{T}_D$ is the actual departure time of a packet, m identifies a packet in Flow D1, n identifies a packet in Flow D2 and k identifies a packet in Queue q	52
4.3	Timing Diagram at edge router e which aggregates Flow X , Flow Y and Flow Z	66

4.4	Timing Diagram at core router c with $D_e = 2$ in an ideal situation whereby packet's arrival at core router c is equal to the expected departure time ($\widehat{T}_D$) at the edge router e and $\delta = 0$	66
4.5	Timing Diagram at core router c with $D_e = 2$ in a non-ideal situation whereby packet's arrival at core router c is equal to actual departure time ($\widehat{T}_D$) at edge router e and δ is taken into account.	67
4.6	Timing Diagram at core router i and j with $D_e = 2$ in a non-ideal situation whereby packet's arrival at core router i is equal to actual departure time ($\widehat{T}_D$) at edge router e and δ is taken into account.	68
5.1	General Queue Timing Diagram.	77
5.2	State Diagram of a queue implementing the WDEDF Scheduling Algorithm.	83
5.3	Block Diagram of WDEDF for multi-services support.	87
5.4	A single hop configuration used to evaluate WDEDF scheduler.	97
5.5	Throughput of each VBR flow when router implements WFQ, DRR, EDF and WDEDF schedulers. All flows generate traffic at the same rate except for the malicious Flow 9.	97
5.6	Throughput of each VBR flow when router implements WFQ, WDRR, EDF and WDEDF schedulers. All flows generate traffic at their reserved rate (weighted) except for the malicious Flow 9.	98
5.7	Average delay of each CBR flow when router implements WFQ, EDF, WDRR and WDEDF schedulers.	100
5.8	Average delay of each VBR flow when router implements WFQ, EDF, WDRR and WDEDF schedulers.	100
5.9	Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. Flow 1 to 10 are CBR flows, and Flow 11 to 20 are VBR flows.	101
5.10	Worst-case delay of each CBR flow when router implements WFQ, EDF, WDRR and WDEDF schedulers.	101
5.11	Worst-case delay of each VBR flow when router implements WFQ, EDF, WDRR and WDEDF schedulers.	102
5.12	Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. Flow 1 to 10 are CBR flows, and Flow 11 to 20 are VBR flows.	102
5.13	Worst-case delay of each flow when router implements WDRR and WDEDF schedulers. Flow RB 1 to RB6 are CBR flows.	104
5.14	Worst-case delay of each flow when router implements WDRR and WDEDF schedulers. Flow RB 1 to RB6 are VBR flows.	104
5.15	Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF (without QoS forwarding rule) schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. WDEDF scheduler forwards any HOL packet with the earliest $\widehat{T}_D$	106
5.16	Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF (without QoS forwarding rule) schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. WDEDF scheduler forwards any HOL packet with the earliest $\widehat{T}_D$	107

5.17	Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation except TC3. TC3 requested a LR-decoupled reservation of 0.5 kbps with a maximum packet delay of 1.4 s. ω^{MIN} of WDEDF is $30L^{MTU}$	108
5.18	Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation except TC3. TC3 requested a LR-decoupled reservation of 0.5 kbps with a maximum packet delay of 1.4 s. ω^{MIN} of WDEDF is $30L^{MTU}$	109
5.19	Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and $\omega^{MIN} = 30L^{MTU}$	110
5.20	Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and $\omega^{MIN} = 30L^{MTU}$	110
5.21	Throughput of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and all RB flows are well-behaved except the malicious flow - RB5.	111
5.22	Worst-case delay of each flow using WFQ, EDF, WDRR and WDEDF schedulers and $\omega^{MIN} = 30L^{MTU}$. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and all RB flows are well-behaved except the malicious flow-RB5.	111
5.23	Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. To create packet arrival variances at the scheduler, JS flows are token bucket regulated with $\sigma = 3L^{MAX}$	113
5.24	Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. To create packet arrival variances at the scheduler, JS flows are token bucket regulated with $\sigma = 3L^{MAX}$	113
5.25	Average Inter-packet Arrival Delay of each JS flow when router implements EDF and WDEDF schedulers, plotted with a variance at 99% confidence.	114
5.26	Bandwidth allocation for each flow when router implements WDEDF with different ω^{MIN} value.	116
5.27	Average delay of each flow when router implements WDEDF scheduler with different ω^{MIN} value. All TC flows are leaky bucket regulated and requested a LR-coupled reservation.	116
5.28	Worst-case delay of each flow when router implements WDEDF scheduler with different ω^{MIN} value. All TC flows are leaky bucket regulated and requested a LR-coupled reservation.	117
6.1	PODP Reference Network	127
6.2	A flow chart that shows router's response on receiving PODP messages.	128
6.3	A flow chart that shows BB's response on receiving PODP messages	128
6.4	Operation of PODP	131

6.5	Functional Architecture of the PODP Daemon Modules	135
6.6	Storage requirements of <i>Method A</i> against PODP	138

List of Tables

2.1	Comparison of packet scheduling algorithms	28
3.1	Traffic patterns for a flow used for simulation studies	39
3.2	Traffic profiles for a flow set used for simulation studies	39
3.3	Throughput (<i>kbps</i>) of flows from Flow Set A at node E1	45
4.1	Table of notations used in EVTRS	53
5.1	Packet Departure Times used in Example 1	78
5.2	Packet Completion Times used in Example 1	78
5.3	Percentage (%) of packets delivered within specified deadline	103
5.4	Traffic Patterns for flows used in simulation study.	105
5.5	Bandwidth of each flows	115
6.1	Flow Sequence of PODP	129
6.2	Table of parameters used for evaluating PODP	137
B.1	<i>Discovery Message</i>	164
B.2	<i>Reporting Message</i>	165
B.3	<i>Notifying Message</i>	166
B.4	<i>Recovery Message</i>	167

Acronyms

DM	<i>Discovery Message</i>
NM	<i>Notifying Message</i>
RCM	<i>Recovery Message</i>
RM	<i>Reporting Message</i>
BB	Bandwidth Broker
BE	Best-Effort
CBR	Constant Bit Rate
CSFQ	Core Stateless Fair Queuing
DiffServ	Differentiated Services
DPS	Dynamic Packet State
DRR	Deficit Round Robin
EDF	Earliest Deadline First
EVTRS	Edge-based Virtual Time Reference System
HNI	Host-to-Network Interface
HOL	Head-Of-Line
IFD	Inter-flow Departure Time
IntServ	Integrated Services
IPD	Inter-packet Departure Time
JS	Jitter-Sensitive

LR	Latency-Rate
MTU	Maximum Transmission Unit
NNI	Network-to-Network Interface
PODP	Path and Oracle Discovery Protocol
QoS	Quality-of-Service
RB	Rate-Based
RSVP	ResSerVation Protocol
RSVPTE	ResSerVation Protocol - Traffic Enginnering
RVC	Relative Virtual Clock
SCORE	Stateless Core
SIBBS	Simple Inter-Bandwidth Broker Signaling
TC	Time-Critical
VBR	Variable Bit Rate
VTRS	Virtual Time Reference System
WDEDF	Weighted Deficit Earliest Departure First

WFQ	Weighted Fair Queuing
YESSIR	Yet another Sender Session Internet Reservations

Abstract

Service providers and enterprises used to build and support separate networks to carry their voice, video and mission-critical traffic. There is a growing trend to converge all these networks into a single, packet-based Internet Protocol (IP) network. The Internet and Corporate Intranet have grown exponentially during the last few years and applications other than traditional data, such as Voice-over-IP and video-conferencing, are becoming prevalent. Today, the Internet offers only best-effort service. However, this service does not provide IP packet delivery guarantees.

Researchers have overwhelmed the Internet with complex approaches in providing Quality-of-Service (QoS) at the transport layer. The Internet Engineering Task Force has proposed two reservation approaches: hop-by-hop bandwidth reservation (Integrated Services) in which a QoS-sensitive flow makes bandwidth reservation on each hop over its connection path; and per-hop behaviour bandwidth reservation (Differentiated Services) in which QoS-sensitive flows with similar QoS requirements are aggregated at the edge routers of a network domain into service classes that have a predetermined QoS guarantee promised by each core routers in the network domain.

The contribution of this dissertation is threefold: an edge-based QoS architecture; a multi-service scheduling discipline; and a generic signalling protocol. We focus on an edge-based QoS architecture that provisions QoS for flows based on traffic information gathered by edge routers. In this architecture, traffic allowed into the core network by each edge router is viewed as a *packet train*. Edge-based Virtual Time Reference System (EVTRS) is a virtual time reference system proposed to provide information on a flow and an aggregation of flows for core routers. This information includes inter-flow's packet departure time, packet lag time and inter-packet departure time. Using a virtual clock technique, we can derive useful functions like flow extraction, instantaneous service rate and a granular packet dropping algorithm.

We proposed Weighted Deficit Earliest Departure First (WDEDf), a new work-

conserving scheduling discipline. It uses quantum control to maintain the ratio of bandwidth allocated to each queue, and the deadline of packet at the Head-Of-Line of each queue together with a non-preemptive priority scheme that categorises packets by their QoS requirements to determine the next packet to be forwarded. WDEDF is able to provide delay guarantees for Latency-Rate-coupled and Latency-Rate-decoupled reservations; minimises jitter for Jitter-Sensitive flows; and provides throughput guarantees for Rate-Based flows and Best-Effort traffic. These properties allow WDEDF to support delay guaranteed service, delay guaranteed service with jitter minimisation and rate guaranteed service.

Lastly, we proposed Path and Oracle Discovery Protocol (PODP), a generic signalling protocol, that supports hop-by-hop, domain-by-domain and specified domains signalling for centralised network management control. It is a reactive protocol that identifies routers and neighbouring domain Oracles.

Therefore, our edge-based QoS architecture uses PODP for QoS-sensitive flows to make reservations in conjunction with a centralised Oracle - Bandwidth Broker to administer each network domain and interact between neighbouring network domains at the control plane. On the data plane, these flows are classified into their respective service classes whereby each flow is regulated at the edge of the network and EVTRS is used to compute traffic information that is embedded into packets of these flows. The embedded information is used by the WDEDF scheduler at the edge and core routers.

Citations to Previously Published Work

Chapter 3 and Chapter 4 describing the Edge-Based QoS Architecture and Edge-based Virtual Time Reference System have appeared in the following paper :

“EVTRS: Edge-based Virtual Time Reference System”, Teck Meng Lim, Bu Sung Lee and Chai Kiat Yeo, In the proceedings of the IASTED International Conference on Internet and Informational Technology, Scottsdale, Az, USA, November 17 to 19, 2003, page number 66-71.

Chapter 5 has appeared in the following paper :

“Weighted Deficit Earliest Deadline First Scheduling”, Teck Meng Lim, Bu Sung Lee and Chai Kiat Yeo, In the proceedings of the IEEE International Conference on Networks, Sydney, Australia, October 30 to November 1, 2003, page number 437-442.

and

“Weighted Deficit Earliest Deadline First Scheduling”, Teck Meng Lim, Bu Sung Lee and Chai Kiat Yeo, Invited for publication in a special issue for the Journal of Computer Communications, To appear in mid 2005.

The Path and Oracle Discovery Protocol presented in Chapter 6 has appeared in the following :

“Path and Oracle Discovery Protocol for Centralized Bandwidth Reservation Mechanisms”, Teck Meng Lim, Bu Sung Lee and Chai Kiat Yeo, Journal of Networks and Systems Management, 11(4):447-468, December 2003.

and an application of the Path and Oracle Discovery Protocol for the DiffServ Architecture has appeared in the following paper :

“BBDP: Bandwidth Broker Discovery Protocol”, Teck Meng Lim, Bu Sung Lee and Chai Kiat Yeo, In the proceedings of the IEEE International Conference on Information, Communication & Signal Processing, Singapore, October 15 to 18, 2001.

*Dedicated to my father,
my mother,
and my sister.*

Chapter 1

Contribution

1.1 Introduction

The Internet standardisation of Quality-of-Service (QoS) architecture begins with the Internet Engineering Task Force (IETF) ¹ who proposed the Integrated Services (IntServ) [1] architecture as a QoS architecture for Internet Protocol (IP) networks which builds on the initial work of Clark et al. [2].

IntServ, developed in the early and mid-1990s, provides absolute QoS guarantees on packet delays and no packet losses for individual flows, so long as the traffic of these flows conforms to a pre-specified set of parameters. Every packet is inspected at each router to determine the service guarantees it will receive, thus, routers must keep per-flow state information which poses a scalability concern on the architecture.

In late 1990s, the interest in QoS shifted to the Stateless Core (SCORE) architecture that makes a distinction between the operation performed by edge routers and core routers. The core of a network does not permit complex QoS mechanisms; while most of the QoS control mechanisms should be executed at the edge of a network where the volume of traffic is smaller.

¹<http://www.ietf.org>

The Differentiated Services (DiffServ) [3] is proposed based on the SCORE architecture where flows of similar QoS requirements are bundled and classified into a set of predetermined service classes. This mapping of flows into service classes is performed at the edges of the network where computational resources are less scarce than in the core of the network. This leaves the core network with a small set of service classes to work with, thus, complexity is reduced. DiffServ offers two different types of *Per-hop Behaviours* for core routers : Expedited Forwarding (EF) [4] and Assured Forwarding (AF) [5] whereby each Per-hop Behaviour (PHB) [6] offers multiple services. EF offers absolute service guarantees on delay variations for classes of traffic that can be viewed as a virtual leased line. However, this leads to low resource utilisation. Thus, it is envisioned that the EF service can only be applied to a very limited number of flows [7]. AF provides isolation between different service classes, and qualitative loss differentiation using a *drop precedence* within each service class. As AF can only provide qualitative relative guarantees, service assurance is limited. Thus, DiffServ provides the soft delay guarantees.

Zhang et al. proposed a Virtual Time Reference System (VTRS) [8] for the SCORE architecture to provide hard delay guarantees for the SCORE architecture. In VTRS, a stateless core scheduling algorithm, Core Stateless Fair Queuing (CSFQ) [9], is deployed to provide the same capability as a stateful scheduling algorithm like Weighted Fair Queuing (WFQ) [10] by using *virtual time stamps* embedded into each packet's header. It reconciled the granularity of IntServ's guarantees with the simplicity of SCORE architecture by elevating state information needed to provide IntServ's guarantees. VTRS computes and embeds a *virtual time stamp* in every packet of each flow. This *virtual time stamp* is referenced and updated by each core routers as the packet traverses through the core network. Thus, VTRS provides a scalable support for guaranteed services.

1.2 Main Contribution

The next step in network QoS provisioning becomes tougher when interactive applications become popular. Interactive applications generate traffic that comprises diverse requirements on the network. For example, the Tele-Immersion [11], a collaborative virtual reality application, generates audio stream, video stream, control data and bulk data. These streams have different requirements in bandwidth, latency sensitivity, jitter sensitivity and error sensitivity. Most applications assume that networking problems are solved by making QoS guarantees. However, current Network QoS solutions are based only on bandwidth provisioning needed by applications.

For DiffServ, the usage of stateful scheduler like WFQ provides strong bandwidth allocation and protection between service classes. In addition, the delay of a service class is fixed based on its predetermined rate. However, this delay bound is based on statistics. When more traffic is admitted into the same class, the average delay also increases. Due to the limitation of buffers, packets will be dropped. This bounds the worst-case delay but loss of data becomes unpredictable, as the schedulers have no detailed information on its traffic. In addition, jitter cannot be controlled or minimised as packets can be accelerated or delayed by the core routers due to variations of traffic load. Although DiffServ is scalable, it provides coarse QoS guarantees.

Although the VTRS provides a hard delay guarantee for each flow, the delay guaranteed is still coupled to the allocated rate as it emulates a stateful scheduler - WFQ. This coupled effect will reduce the utilisation of the network with flows that have intolerable delay and low arrival rate. Network game traffic [12][13][14], a typical example of a traffic that has a latency bound, is not coupled to its arrival rate. This traffic is highly interactive and real-time critical. A popular client/server network game "Counter Strike" can generate network traffic of small average packet sizes (127 *bytes* from server and 82 *bytes* from client) in large bursts (30% of packets arriving in a single burst). However, it has low bandwidth

requirements. The game server generates data on an average of 16 *kbps* and at a peak rate of 18 *kbps*. Each client generates a more deterministic traffic at 15.7 *kbps*.

The contribution of this dissertation is to provide a solution for the SCORE architectures which gives hard delay guarantees for flows using a reservation rate that is coupled or decoupled from the latency requirement; minimises jitter for flows that are sensitive to jitter and delay; and provides throughput guarantees for all flows.

To achieve this objective, we introduced an edge-based QoS architecture based on the SCORE architecture which includes three components. First, we proposed an edge-based virtual time reference system that defines and computes state information that are embedded into each packet in a *packet train*. We envision that edge routers, in which per-flow states are kept, have the ability to aggregate flows into a single *packet train* fairly; whereas core routers ensure a predetermined QoS for each *packet train* using the embedded *packet train* properties. In addition, it also provides a scheme to maintain the properties of a *packet train* when a *packet train* is split into child *packet trains* due to multiple output links at a router. After that, we proposed a multi-service scheduling discipline that can provide delay guarantees with reservation rate that is coupled or decoupled from the latency requirement, throughput guarantees and jitter minimisation. Finally, based on our assumption of using a centralised Oracle to manage a SCORE domain, we proposed a reactive signalling protocol that records the route taken by a QoS-sensitive flow and also the neighbouring domain Oracles. The latter facilitates the communication between neighbouring domain Oracles when negotiating for a service contract during a reservation setup.

1.3 Organisation

This dissertation is organised as follows: Chapter 2 provides an overview of Network QoS and describes three prominent architectures proposed to provide guaranteed services for the Internet, namely, Integrated Services, Differentiated Services and Virtual

Time Reference System. In addition, this chapter also gives an overview of QoS control components used in these architectures.

Chapter 3 describes the architecture proposed in this dissertation - Edge-based QoS architecture. It describes the three main components used to achieve QoS guarantees for network applications, namely, Edge-based Virtual Time Reference System, Weighted Deficit Earliest Departure First Scheduler and Path and Oracle Discovery Protocol. We also demonstrate, using simulation, that this architecture is able to provide QoS-sensitive applications with their required network performance.

Chapter 4 introduces Edge-based Virtual Time Reference System, a virtual time reference system, that provides aggregation information and flow information of a *packet train* for core routers. It also includes an updating method to maintain properties of *packet trains*. We show that our edge-based virtual time reference system converges mathematically and is observable using work examples.

Chapter 5 presents the Weighted Deficit Earliest Departure First scheduling discipline used in our architecture. Through analytical results and simulations, we illustrate that this scheduling discipline is able to provide flow protection to each queue; weighted bandwidth allocation to queues; delay differentiation between queues; and support delay guarantees, minimise jitter and rate guarantees for multi-services.

Chapter 6 details our generic signalling protocol - Path and Oracle Discovery Protocol, which is designed to facilitate the management of networks with centralised bandwidth reservations. Through numerical examples, we show that this reactive protocol is able to respond to network path changes at minimum network overhead and storage requirements as compared to other proposed methods for centralised bandwidth reservations.

Finally, Chapter 7 concludes this dissertation and ends with directions for future work.

Chapter 2

Background

Packet switched networks (e.g. the Internet) started off with only Best-Effort (BE) service for users. With the lifting of the regulation against commercial use of the Internet Backbone in 1991, the commercial networks and the Internet merged; and demands for QoS on the Internet evolves as user pays for Internet usage and some users are willing to pay extra for better performance to support real-time applications like video conferencing and voice over IP. To address this demand, the IETF introduces QoS architecture standards for the Internet. A QoS architecture introduces tools to treat packets differently, thus one flow can receive better performance at the expense of other flows.

This chapter provides an overview of QoS architectures and their components for the Internet. Section 2.1 starts with an introduction to Network QoS. Section 2.2 describes QoS architectures currently proposed for the Internet while Section 2.3 describes briefly the common QoS control mechanisms used to support QoS.

2.1 Internet Quality-of-Service Overview

In the Internet, applications generate flows. Each flow consists of a stream of packet from a source to a destination. In a connection-oriented network, all the packets of

a flow follow the same route. Network QoS requirements of a flow can be categorised by a diverse set of service requirements such as performance guarantees, reliability and security. Typical metrics for Network QoS performance guarantees include rate, delay, jitter and packet loss rate. The QoS requirements for a flow depend on the application needs in network performance. For example, real-time applications, such as telephony and video conferencing have strict delay requirements; and playing video and audio files from a server are sensitive to jitter.

An easy technique for achieving QoS is to provide more than sufficient link capacity and buffer space so that all the flows meet their QoS requirements. Besides over-provisioning of network resources, providing QoS support in packet switched networks requires collaboration of different QoS control mechanisms.

At packet level, QoS control mechanisms include traffic conditioning devices (e.g. traffic classifiers, markers, policers, and shapers), packet schedulers, and active queue management. In particular, scheduler decides the packet to be transmitted next on the output link, hence, significantly impacting QoS guarantees for different flows. Scheduling affects delay, jitter and loss rate; and allows protection against misbehaving flows. Congestion and flow controls are feedback-based QoS control whereas admission control and QoS routing are session-based QoS control. Admission control limits the number of flows admitted into the network so that each individual flow obtains its desired QoS; and congestion control prevents, handles and recovers from network congestion scenarios.

2.2 Quality-of-Service Architectures

A QoS architecture uses QoS control mechanisms operating at different location of the network (e.g. end hosts, edge routers, core routers and etc) to treat packets differently so that QoS-sensitive packets get premium services over non-QoS-sensitive packets.

QoS architectures also maintain control information (i.e., control state) for making

control decisions and exerting controls. Control information can be maintained with different granularity, for example, the finest granularity of control information is the per-flow state information which can be used to enforce QoS for individual flows; whereas coarser-grain control information like per-service class, per-edge pair, can be maintained for different level of flow aggregations. This control information can be stored in packet headers, or network elements like routers, bandwidth brokers and etc.

2.2.1 Integrated Services

Figure 2.1: An Integrated Services Model [1].

The IntServ Architecture is designed to provide a set of extensions to the BE traffic delivery model for the Internet. This framework is set up to enable special handling for QoS sensitive traffic to support of multimedia applications.

IntServ requires prior network resource reservation over the connection path for each flow. It defines reservation specifications that are delivered hop by hop from source host to destination host. ResSerVation Protocol (RSVP) [15] is a signalling protocol for IntServ to facilitate delivery of reservation specifications and set up the right parameters

in the routers, such that reserved flows receive their desired QoS. It is a feedback-based protocol that requires routers to perform reverse path route pinning (i.e., keeping reverse path information for each reservation). More precisely, a sender starts sending a path message to a receiver. As this message traverses through the network, a path state is created on each router, storing reverse path information so that the reservation message from the receiver can be forwarded back to the sender. In return, the receiver sends reservation message which specifies a flow descriptor that indicates the desired QoS. Reservation is made on each router based on this flow descriptor. As this message is propagated upstream along the reverse path towards the sender, appropriate reservations of resources are done at the routers. Once the sender receives the reservation message, it knows that the desired amount of resources has been reserved on each router along the path to the receiver and the network is ready to provide the desired QoS.

Using RSVP, both the host and the router influence the actual data flow by setting the parameters for QoS controls like packet classifier and scheduler. Fig. 2.1 depicts an IntServ model. IntServ uses a multi-field packet classifier that classifies incoming packets into flows based on the parameters such as IP Addresses, Port and etc; and queues them appropriately in the scheduler. The scheduler schedules the packets according to their flow requirements so that each flow receives its desired QoS. Intuitively, IntServ adopts a distributed reservation method whereby admission control of reservations is performed by routers independently based on the routers resource availability.

IntServ employs per-flow QoS control and stores QoS states at every router on a per-flow basis. It requires a sophisticated control plane at every router that complicates its operations and management, thus limiting its scalability. A major drawback of RSVP is that the amount of state information required increases with the number of per-flow reservations. As hundreds of thousands of real-time flows can exist in the Internet backbone at any time, state information on per-flow granularity is considered a non-scalable solution for Internet

backbone. In fact, Mankin et al. [16] suggest that smaller and private networks are most likely to benefit from the deployment of RSVP. The inadequacies of RSVP scaling and the lack of policy control may be more manageable within the confines of a smaller, more controlled network environment than in the global Internet.

To cater for different QoS service requirements, IntServ defines two types of services: Guaranteed Service and Controlled-Load Service.

Guaranteed Service

Guaranteed Service (GS) [17] aims at assuring a level of bandwidth for an application such that if the traffic is policed, the end-to-end delay will be bounded and there will be no loss of conforming packets. However, this mechanism assumes that the routers responsible for providing the service do not fail or change (due to a change in routing) during the lifetime of the application. This service is well suited for playback and real-time applications. Playback applications use a jitter buffer to offset the delay variations of the packet arrivals to function well. This service guarantees the worst-case queuing delay and it aids in the estimation of required jitter buffer size. Moreover, real-time applications enjoy guaranteed bandwidth and delay service.

However, this is achieved with a significant increase in complexity. Each router needs to maintain and manage per-flow state on both the data and control paths. On the data path, a router must perform per-flow classification, buffer management and scheduling; whereas on the control path, a router must maintain per-flow forwarding state and admission control in which network resources like bandwidth and buffer space for flows are reserved.

Controlled-Load Service

Controlled-Load Service [18] attempts to provide end-to-end traffic behaviour that closely approximates a BE service under unloaded or lightly loaded conditions. This ser-

vice is applicable to a broad class of applications running on the Internet today that are sensitive to network overloaded conditions. These applications work well on lightly loaded network, but they degrade quickly under overloaded traffic conditions. Controlled-Load Service ensures that the packet loss rate is not significantly larger than the error rate of the link; and the end-to-end delay experienced by a very large percentage of packets does not exceed the end-to-end propagation delay. Examples of such applications are File Transfer Protocol (FTP) [19] and adaptive real-time applications.

Controlled-Load Service also requires routers to perform per-flow admission control on the control path; and packet classification, buffer management and scheduling on the data path.

2.2.2 Stateless Core Architecture

Figure 2.2: A SCORE Domain which comprises edge routers and core routers

A SCORE network model, depicted in Fig. 2.2, is a conceptual network model that consists of a contiguous and trusted region of network which comprises edge routers and core routers. Edge routers maintain per-flow state so that they can perform admission control on incoming flows and aggregate them into the network core; while the core routers schedule and forward the core traffic appropriately without maintaining per-flow state. Scalability is maintained in SCORE as edge routers usually run at much lower speed and handle fewer flows than the core routers. DiffServ and VTRS are QoS architectures that are based on

the SCORE network model. These QoS architectures are discussed in Section 2.2.3 and Section 2.2.4 respectively.

The Dynamic Packet State Techniques

The idea behind a dynamic packet state technique is to store any required per-flow states in IP packets and let these states be carried over the domain instead of having core routers storing them. Such techniques provide the SCORE an inexpensive mechanism for information passing. State information are inserted into packets by ingress routers, which maintains per-flow states. Core routers use these states to determine their packet forwarding sequence. It also updates these states for downstream core routers whenever necessary.

Currently, there exist two dynamic packet state techniques. They are used in (1) CSFQ and (2) Multiprotocol Label Switching (MPLS) [20]. In CSFQ, information is fitted into the IP packet using unused bits whereas MPLS adds a new label stack to the existing protocol stack to embed required information to be carried across a network domain.

2.2.3 Differentiated Services

To alleviate the scalability problems with IntServ, the IETF proposed the DiffServ Architecture. DiffServ follows three basic approaches:

- Aggregate reservations for a number of data streams, which, for example, flow between two Internet Service Providers, instead of individual reservation for each data stream.
- Implicit signalling for QoS through the inclusion of the necessary information in the data units. This avoids dedicated signalling with its own independent complex signalling protocol and the status maintenance required in the intermediate systems.
- Provision QoS guarantees for different service classes.

Figure 2.3: A Differentiated Services Domain Model [3].

DiffServ divides the network into domains. Fig. 2.3 depicts a DiffServ domain model. Generally, packet sent from end systems are marked with a DiffServ CodePoint (DSCP) value (stored in the Differentiated Services Field (DS Field) [21] of the packet header) and forwarded across domains via edge routers. Edge routers maintain per-flow or per-aggregate state. They perform traffic conditioning on the flows or aggregated flows; classify them into service classes; and forward packets from each service class to the core routers.

The core routers do not maintain any fine grained state; they only maintain a small number of service classes. These routers identify the service class that a packet belongs to using the DSCP value and forward the packet accordingly. Eventually, a packet will arrive at its receiving host with the desired QoS. Edge routers specifically handle the traffic control for the transmission link carrying the aggregated reservation. A traffic regulator like Token

Bucket Filter (TBF) [22] can be used for this purpose. If a packet cannot be serviced with the existing resource, it is either dropped or re-marked as non-conforming packet on the edge router.

Core routers are responsible for triggering per-hop behaviours based on the DSCP of the packet in which packets are forwarded according to their classified DSCP in the packet header. The first router following the end-system that issued the packet is called a first-hop router. It can mark the packet to the assigned DSCP.

A PHB is a description of the externally observable forwarding behaviour of a core router applied to a set of packets with the same DSCP. PHB is defined in terms of its resource priority relative to other PHBs, or to some observable traffic service characteristics, such as packet delay, loss or jitter. In a DiffServ domain, BE behaviour may be viewed as the default PHB. DiffServ recommends specific DSCP value for each PHB, but a network provider can choose to use a different DSCP from the recommended values. The PHB of a specific service class depends on a number of factors:

- The arrival rate or load for a service class is controlled by the traffic conditioning at the network boundary.
- The resource allocation for a service class is controlled by the resource allocation on the core routers in the DiffServ domain.
- The traffic loss depends on the packet discard policy on the core routers in the DiffServ domain.

DiffServ Architecture is highly scalable as core routers can process packet efficiently as they have very small number of service classes to maintain. Currently, two PHBs, EF and AF, have been standardised. They are discussed in the following sections.

Per-hop Behaviour - Expedited Forwarding

The intention of EF is to support a behaviour aggregate of low loss, low latency, low jitter and assured bandwidth service through a DiffServ domain. This service is also referred to as Premium Service or Virtual Leased-Line (VLL) service of Virtual Private Network (VPN) [23] for bandwidth guarantees.

Constructing a premium service for this behaviour aggregate implies that traffic within this service class must be exposed to minimal, if any, queuing delays in transit. As long as the aggregates' arrival rate at any router is less than the sustainable departure rate, an EF service is supportable. This departure rate must be independent of any load imposed by all other traffic. The packet arrival and departure rates are typically measured at intervals equal to the time it takes for a link's Maximum Transmission Unit (MTU) [24] to be transmitted.

A router can allocate resources for a certain departure rate on an interface by using different EF functionality implementations. Packet scheduling techniques such as WFQ [10], provide this functionality whereby the EF traffic can be carried over a highly weighted queue using a weight that allocates a much higher rate to EF traffic than the actual EF traffic arrival rate. Traffic queues in EF are to be implemented with priority. This ensures that a busy EF priority queue does not potentially starve the remaining traffic queues beyond a certain configurable limit. This service targets at applications such as high assurance voice or video channels (Voice over IP, video conferencing) and services that require VLL.

Per-hop Behaviour - Assured Forwarding

AF is suggested for applications that require a better reliability than the BE service, thus, it attempts to provide a scalable solution to the problem of service differentiation in the Internet. It groups (using DSCP) flows with similar QoS requirements into classes;

and provides a qualitative service differentiation between service classes. A low priority service class experiences higher loss rates and delays than a high priority class. AF neither requires admission control nor per-flow classification. Thus, it is scalable on both the control and data planes. However, it only provides qualitative differentiation between service classes, as some classes receive lower delays and loss rate than others. Service differentiation is not quantified and there is no absolute service bounds offered.

There are four classes of service in which each class is predefined with three drop precedences. This service is also known as Olympic Service. Packet classification and policing are done at the ingress routers of the networks. If a flow does not exceed the bit-rate specified, it is considered as “in-profile”; otherwise, the excess packets are considered as “out-of-profile”. However, this filtering is refrained from reordering packets from the same micro-flow. A Two Rate Three Color Marker (trTCM) [25] admission and marking scheme is proposed to overcome this constraint. In trTCM, a packet is marked “red” if it exceeds some peak information rate, “yellow” if it exceeds some committed information rate and “green” otherwise. This scheme is used to mark a flow where different decreasing levels of assurances are given to packets that are “green”, “yellow” or “red”. For example, a service can discard all “red” packets, because they exceeded the peak rate; forward yellow packets within the BE service class; and forward green packets with a low drop probability, preserving the order of all packets that are not dropped. One approach to implement AF service is to use Class-Based Queuing (CBQ) [26] across the AF service classes, with the weighting adjusted to reflect the relative bandwidth allocations made to each service class. Another approach is to use the same fixed number of service classes and implements WFQ as the scheduling mechanism, to ensure a higher level of accuracy in the per-service class bandwidth allocations. To undertake the queue management within each service class, weighted Random Early Detection (RED) Queue Management [27] can be used. The RED weighting associated with each packet corresponds to the packet’s discard preference as

marked in the DSCP.

2.2.4 Virtual Time Reference System

Dynamic Packet State (DPS) [9] was first proposed by Stoica et al. [9], where control information is embedded into the IP packet header and carried across the network in which routers can update this information for scheduling purposes. DPS is demonstrated in core-stateless version of Jitter Virtual Clock (Jitter-VC) [28] which is proposed to attain the same end-to-end delay bound without per-flow management required by Jitter-VC.

The above scheme is generalised in VTRS by Zhang et al. [8]. VTRS is developed to provide scalable support for guaranteed services. The key construct in VTRS is the notion of *packet virtual time stamps*, which are referenced and updated as packets traverse through each core router. These *packet virtual time stamps* are computed using packet states that are embedded in the packet with DPS techniques and a couple of fixed parameters associated with core routers.

VTRS consists of three logical components, namely, (1) packet states carried by packet using DPS; (2) traffic conditioning at edge routers to ensure that flows will not inject packets at a rate exceeding their reserved rate into the network; and (3) *per-hop virtual time reference and update mechanism* at the core routers to maintain the continual progression of the virtual time embodied by the *packet virtual time stamps*.

Each packet, k , embeds three types of information: (1) QoS reservation expressed using a rate-delay pair $(r_k, \tilde{d}_k)$ where r_k is the reserved rate and $\tilde{d}_k$ is a virtual delay parameter which is the delay a packet expected to experience at a node; (2) *packet virtual time stamp*, $\widetilde{\omega}_i^{j,k}$, of the packet k from flow j associated with router, i . This term is updated at each node; and (3) a virtual time adjustment term, $\delta^{j,k}$, of the packet which relates to a cumulative delay experienced by the packet.

At the network edge, the rate-delay pair is determined by the Bandwidth Broker

(BB) of the domain and inserted into every packet of a flow. The *packet virtual time stamp* is initialised to the actual time the packet, k , of flow j leaves the edge conditioner and enters the first core router; while the virtual adjustment term is set to $\frac{\Delta_{j,k}}{q}$, where q is the number of rate-based schedulers along the flow's path and $\Delta_{j,k}$ is the cumulative delay experienced by the packet in an ideal dedicated per-flow system in which packets of the flow are serviced by q schedulers in tandem with constant rate r_j .

The *packet virtual time stamp* represents the virtual arrival time of the k -th packet of flow j at the i^{th} core router; and is referenced and updated in each core routers. It satisfies two important properties (virtual spacing property and reality check property) which ensure that the end-to-end delay experienced by packets of a flow across the network is bounded. The updating of the *packet virtual time stamp* depends on the schedulers deployed (rate-based or delay-based).

2.3 Quality-of-Service Control Mechanisms

In Fig. 2.1 and Fig. 2.3, we have shown how QoS architectures use QoS control mechanisms to ensure QoS for QoS-sensitive flows. In the following sections, we will describe well-known QoS control mechanisms proposed for the Internet.

2.3.1 Classifiers

The Classifier selects a packet in a traffic stream based on the content of some portion of the packet header. The most common way to classify traffic is based on the Type-of-Service (TOS) field in the IP header, but one may classify traffic based on the other fields in the packet headers. This function is used to identify IP packets for grouping or providing special services.

Type-of-Service Classifier

Figure 2.4: Reclaiming of IP header fields for DSCP field

The IP header contains TOS field (4 *bits*) that is designed for routers to identify and classify packets based on the combination of these bits. The DiffServ Architecture introduced a DSCP field which extends the TOS field to a 6 *bits* field by reclaiming an extra 3 *bits* from the IP header as shown in Fig. 2.4.

Multifield Classifier

The multifield classifier [29] extends the Type-of-Service classifier by looking at a more general set of classification conditions. These conditions can be based on other information in IP header which includes source IP address, source port, destination IP address, destination port; and Transmission Control Protocol (TCP) [30] or User Datagram Protocol (UDP) [31] packet header.

2.3.2 Shapers and Policers

Traffic shaping regulates the average rate or burstiness of input traffic. It controls the amount and volume of traffic being sent into the network. This reduces congestion so that routers can maintain QoS guarantees to its reserved flows.

Leaky Bucket Filter

Figure 2.5: A Leaky Bucket Filter.

The Leaky Bucket Filter (LBF) enforces a rigid output pattern at a specified rate, r , (see Fig. 2.5) no matter how bursty is the input traffic. It has a bucket size, M , which at maximum allows packets with the sum of their sizes not exceeding M to pass through.

Token Bucket Filter

Figure 2.6: A Token Bucket Filter.

TBF can be imagined as a LBF that holds tokens at a specified rate, r , as shown in Fig. 2.6. Thus, TBF allows saving of tokens up to maximum bucket size, σ , which means that some burstiness is allowed at the output of the TBF. In the TBF, extra tokens are discarded but extra packets are kept in its buffer.

TBF allows large bursts but it is necessary to reduce the peak rate of each burst. This can be resolved by adding a LBF after the TBF.

Dual Token Bucket Filter

Figure 2.7: A Dual Token Bucket Filter.

Figure 2.8: Arrival Curve of a Dual Token Bucket Filter with parameters (σ, r, ρ, M) .

A Dual Token Bucket Filter (Dual-TBF) [32] can be implemented using a two stage TBF configured as shown in Fig. 2.7. When a Dual-TBF is used as a traffic shaper for a flow, the first TBF has a depth of σ bytes that limits the maximum burst size of the

flow and a sustained rate of r bps, while the second TBF has a depth of M bytes that takes the size of the maximum packet size of the flow such that the condition ($\sigma \geq M$) is satisfied and a peak rate of ρ bps. This traffic shaper introduces an additional delay to the flow which should be accounted for when measuring the end-to-end delay of the flow.

As a traffic policer, there is no *Buffer* at the input of the first TBF. This guarantees that a flow will not send more than $\min(M + \rho\tau, \sigma + r\tau)$ bytes in any interval τ . The arrival curve (see Fig. 2.8) of the Dual-TBF comprises two linear segments with one of slope ρ and the other of slope r . This curve changes from one segment to the other at its intersection point at $T = \frac{\sigma - M}{\rho - r}$.

2.3.3 Packet Scheduling Algorithms

Packet switching networks allow users to share buffer and bandwidth. Given a number of flows multiplexed on a link, a packet scheduling algorithm is needed to determine the packet service order.

Numerous packet scheduling algorithms have been developed to support network services which can be generally classified as follows :

- **Best-Effort** A service that does not impose any commitments on QoS.
- **Guaranteed Throughput** A service that ensures each flow is allocated a predetermined bandwidth regardless of the behaviour of other flows sharing the same link.
- **Bounded Delay-Jitter** A service that bounds the maximum delay difference between two packets.
- **Bounded Delay** A service that guarantees an upper bound on observed packet delays.

The recent popularity of network games [12] [13] [14] that generate low bandwidth and intolerable delay traffic increases the challenge of guaranteeing delay bound for this traffic type while maintaining high utilisation.

Scheduling that approximates a Generalised Processor Sharing (GPS) [33] model has become the central issue in providing multimedia services with various QoS requirements for IP routers. GPS is based on a fluid flow model that is not practical for realisation. A class of packet fair queuing algorithms has been proposed to emulate this fluid model.

Weighted Round Robin and Deficit Round Robin

Weighted Round Robin (WRR) [34] scheduler is a quantum-based control scheduler that ensures differentiated fairness between queues. The scheduler polls each queue in a cyclic manner in proportion to a weight assigned to each queue. In each poll, the scheduler serves a packet from a non-empty queue encountered. Thus, a malicious flow only overflows its own queue.

Deficit Round Robin (DRR) [35] scheduler modifies WRR scheduler for handling of variable packet sizes without knowing the mean packet size of each flow in advance. Each queue is allocated a fixed quantum and is serviced in a round robin manner. When a queue is served, its allocated quantum reduces according to the size of the packet sent. This deduction continues until sending of the next packet would result in a deficit of quantum. The final balance of the quantum is added to the quantum assigned in the next service round. DRR scheduler provides near-perfect throughput fairness and isolation of flows.

Weighted Fair Queuing

WFQ [10] approximates the GPS [33] model such that it does not make GPS's infinitesimal packet size assumption and does not need to know a connection's mean packet size in advance. WFQ computes the time a packet would complete its service before serving them in order of their finishing times as if it was served by a GPS that serves its connections bit by bit. Instead of using finishing times as in the GPS model, WFQ uses a service tag (*finish number*) which indicates the relative order in which the packet is to be served. The

computation of the *finish number* depends on a variable - *round number*, R . Suppose that, in each round, WFQ serves a bit from each active connection, thus a R is equivalent to the number of rounds of service completed (i.e., a *round number* is a variable that increases at a rate inversely proportional to the number of currently active connections). Each round of service takes a variable amount of time. The more connections served in a round, the longer the round takes. The *finish number* of a packet $F_q^k(t)$, from connection q , at time t is

$$F_q^k(t) = \max(F_q^{k-1}(t), R(t)) + L_q^k$$

where L is the size of the packet and $R(t)$ is the number of rounds of service that has completed for all active connections at time t . A connection is active if the largest *finish number* of a packet either in its queue or last served from its queue is larger than the current *round number*. To determine the *finish number* for each packet, WFQ recomputes the *round number* on each packet arrival. This re-computation requires a WFQ to determine the current *round number* which leads to a problem called **iterated deletion** [36]. This problem is caused by the deletion of inactive connections from the list of active connections. A connection is deleted once it becomes empty. When a connection empties, its R increases at a faster rate (i.e., the value of R approaches the *finish number* of other packets, faster than originally evaluated). This expedition can cause some of the other flows to complete which leads to a further increase in R 's rate of change. To keep track of R , WFQ must recompute the value of R whenever a packet arrives or leaves. Self-clocked Fair Queuing (SCFQ) [37], a variant of WFQ addresses this problem, in which it proposed a way to speed up the R computation.

WFQ requires per-flow or per-aggregate state to be kept. In addition, it requires explicit packet sorting of the output queue. Despite these, WFQ can be used to give per-flow

performance guarantees.

Virtual Clock

The Virtual Clock (VC) [38] scheduling algorithm stamps packets with a tag that is computed based on an emulation of time division multiplexing instead of the GPS model. When all queues are non-empty, VC and WFQ provide identical service and identical worst-case end-to-end delay bounds. However, the relative delay bound for BE traffic in VC is infinite. For two non-empty queues, one queue can obtain a throughput infinitely more than the other.

Earliest Deadline First

In the classic Earliest Deadline First (EDF) [39] scheduling algorithm, a packet is assigned a deadline and the scheduler serves packets in order of their deadlines. However, some packets will miss their deadline if the scheduler is over-committed.

Ferrari et al. present Delay Earliest Due Date (Delay-EDD) [40], a scheme for real-time channel establishment in wide area network whereby a deadline policy is used. Delay-EDD reserves bandwidth at a session's peak rate and sets packet's deadline for this session to a time at which this packet should have been received no faster than its peak rate. Therefore, every packet receives a hard delay bound if the session obeys the peak rate constraint. The delay bound of packet in a session is independent of the bandwidth reservation, i.e., a session that reserves a small bandwidth can still obtain a small delay bound.

To support delay-jitter bound, Verma et al. propose Jitter-Earliest Due Date (Jitter-EDD) [41] which extends Delay-EDD by incorporating delay-jitter regulator preceding a Delay-EDD scheduler. In Jitter-EDD, each packet is stamped with the difference between its deadline and the actual finishing time after being served by a scheduler. The

delay-jitter regulator at the entrance of the next scheduler holds the packet for this period before releasing it to the Delay-EDD scheduler. This removes the effect of queuing delay variability at the previous scheduler which in turn avoids the burstiness built up within the network.

Comparison of Packet Scheduling Algorithms

We summarise the packet scheduling algorithms (see Table 2.1) discussed in the previous sections by looking at several dimensions such as work-conserving¹ or non-work-conserving², bandwidth bound, delay bound, delay-jitter bound; and decoupled delay and bandwidth allocation. An extensive comparison of packet scheduling algorithms is presented by Zhang et al. [42, 43].

GPS emulation schedulers, like WRR, DRR, WFQ and SCFQ, provide bandwidth protection between sessions with a specific degree of fairness. In addition, they provide bandwidth and delay bounds. These schedulers are classified as Latency-Rate (LR) [44] schedulers whereby the delay bound for a session is coupled with the amount of bandwidth reserved. For LR schedulers, a session with small delay bound requires large bandwidth reservation.

VC provides per-session bandwidth and delay bound without computing complicated round numbers like WFQ. However, it has potentially unlimited relative unfairness between two sessions. Unlike those GPS emulation algorithms, both the Delay-EDD and Jitter-EDD algorithms provide bandwidth and delay bounds without coupling them. Jitter-EDD also provides delay-jitter bound by using separate delay-jitter regulator for each session. This increases the implementation complexity and the scheduler becomes non-work-conserving.

¹A scheduler is considered as work-conserving if and only if it is idle when there is no packet awaiting for its service.

²A scheduler is considered as non-work-conserving if it can be idling even when there is a packet awaiting for its service.

Table 2.1: Comparison of packet scheduling algorithms

Dimension	GPS Emulations (WRR,DRR,WFQ)	VC	Delay-EDD	Jitter-EDD
Work-conserving	✓	✓	✓	×
Bandwidth Bound	✓	✓	✓	✓
Delay Bound	✓	✓	✓	✓
Delay-Jitter Bound	×	×	×	✓
Decouple delay and bandwidth allocation	×	×	✓	✓

2.4 Summary

We provide an overview of Network QoS; and describe QoS architectures and the QoS control mechanism used in handling QoS-sensitive packets. These control mechanisms are used mainly for packet classification; traffic shaping and policing; and scheduling of packets. A router must be able to determine packets from a particular flow correctly in order to impose scheduling policy for these packets. This requires packet classifiers. To enable grouping of flows, packets can be marked after being classified. Marking involves writing or rewriting of information on IP packet. Packets are marked so that downstream routers can group them (by classifiers) for a pre-specified treatment (e.g. DiffServ groups packets with similar QoS requirements into a pre-defined PHB). Currently, marking is performed on the TOS or DSCP bits in IP header. To ensure that the network always has sufficient capacity to service all its reserved flows, the volume of traffic entering a network must be controlled at the reserved rate of each flow. This function requires traffic shaping and traffic policing. With traffic load controlled and packets classified into different queues, routers deploy a scheduling discipline to select the next packet to be serviced. This scheduling discipline also manages resources (i.e., buffers) allocated to its queues.

Using QoS control mechanisms, each QoS architecture provides QoS-sensitive flows with their specified network requirements. These architectures differ in their method of

reserving network resources for these flows.

IntServ consists of two service classes, namely, GS and Controlled-Load Service. GS provides an assured level of bandwidth, a tight bound on end-to-end delay and no queuing loss guarantees for flows that conform to their traffic specification. Controlled-Load Service offers a service similar to that of a BE flow in a lightly loaded network. RSVP is a signalling protocol used to signal QoS requirements to the routers. Routers propagate these requirements (with or without modifications) to other router en route to the receiver. This creates an end-to-end per-flow reservation state on the routers. RSVP is a soft state protocol whereby the reservation states in the routers are periodically refreshed and are automatically deleted if they are not used for some pre-specified duration of time. However, the state holding and management in the intermediate routers make RSVP non-scalable.

DiffServ avoids the disadvantages inherent in the IntServ. It provides QoS guarantees for aggregated data streams instead of per-flow reservation, thus making it a more scalable differentiation services for the Internet. In DiffServ, flows are assigned to different PHB using DSCP at the edge of the network. At the core of the network, packets are forwarded according to the PHB associated with each unique DSCP. Two PHBs have been formally defined, namely, EF and AF. EF provides low loss, low latency, low jitter and assured bandwidth service through a DiffServ network. AF allows service providers to offer different levels of forwarding assurances for IP packets. This service provides a minimisation of long-term local congestion events, while allowing short-term burst traffic. It is suitable for most TCP-based applications, in which individual packets within a micro-flow may be dropped, but packets reordering within the network may present false congestion signals to the sending TCP stack.

Finally, we also describe the SCORE architecture and introduce a virtual time referencing system for core-stateless virtual clock scheduler to provide a scalable support for guaranteed services that has the same capability as a stateful WFQ scheduler.

Chapter 3

Edge-based QoS Architecture

The main contribution of this dissertation is to provide a QoS solution for the SCORE network architecture. We look at a network whereby network traffic is generated at an edge router which accepts per-flow reservations and classifies them into a set of predetermined service class; while a core router ensures different QoS guarantees for each edge router's service classes. This chapter provides an overview of the three components (Edge-based Virtual Time Reference System, Weighted Deficit Earliest Departure First, and Path and Oracle Discovery Protocol) proposed and shows the performance of our edge-based QoS architecture through simulations. The detail description and discussion of these three components are presented in the subsequent chapters.

This chapter is organised as follows. We present our architecture in Section 3.1; and conduct simulation investigations of our architecture and compare it with DiffServ and IntServ architectures in Section 3.2. Finally, Section 3.3 summaries our findings.

3.1 Edge-based QoS Architecture

We adopted the SCORE network model that is used in DiffServ and VTRS. In this model, edge routers (usually have lower speed and handle fewer flows as compared

to core routers) maintain per-flow state so that they can perform admission control on incoming flows and aggregate them into the network core; while the core routers schedule and forward the core traffic appropriately without maintaining per-flow state. In addition, we also used the DPS technique that is proposed for the SCORE network model to embed state information into the core of the network.

A model of our architecture is depicted in Fig. 3.1. Our model is similar to the DiffServ Domain Model in which packets from each flow at an ingress router are classified into a predetermined service class whereby such a service class receives a pre-defined service level at the scheduler. In addition to marking each packet with an unique service class identifier (like the DSCP for DiffServ), our model marks each packet with a unique identifier of an ingress router. This allows core routers to group packets by their service class and the ingress router that they had traversed, i.e., our model does not aggregate the packets of the same service class from different ingress router at the core router.

Thus, our architecture builds on the foundation of the SCORE network architecture with the DPS technique that is described in Section 2.2.2. Based on SCORE, we proposed three components to realise our architecture. In Section 3.1.1, we give an overview of an edge-based virtual time reference system which describes how edge-based service classes are organised into *packet trains* and how the properties of *packet trains* are defined and maintained. After that, we briefly describe a scheduling discipline in Section 3.1.2 which is used to schedule *packet trains* for our architecture. Finally, we provide an overview of a generic signalling protocol in Section 3.1.3 for triggering any QoS reservation while it records the route taken by a flow and discovers neighbouring centralised Oracles which are used to maintain SCORE domains.

The three main components (edge-based virtual time reference system, scheduling discipline and generic signalling protocol) proposed for our architecture are discussed in details in Chapter 4, Chapter 5 and Chapter 6 respectively.

Figure 3.1: Edge-based QoS Domain Model that consists of two ingress routers. Ingress 1 router contains TC, JS, RB and BE service classes in which each service class contains two flows; and each core router ensures QoS for each edge router's service classes.

3.1.1 The Edge-based Virtual Time Reference System

In our architecture, edge routers aggregate flows into a single *packet train* for each service class; and core routers ensure a predetermined QoS for each *packet train* using the embedded *packet train* properties. In other words, flows from each ingress router are aggregated and forwarded down a tree of core routers to their designated egress router. A *packet train*, consisting of an aggregation of packets from different flows of the same service class, is formed implicitly at the output of the ingress router. As traversing down the tree of core routers splits this *packet train* into shorter *packet trains*, an updating mechanism is required to maintain the properties of the *packet trains*.

With this motivation, we introduce an Edge-based Virtual Time Reference System (EVTRS) [45] to maintain and update QoS information on per-flow and per-aggregate basis in a distributed manner. The ingress router classifies flows by their service class and delivers packets from these flows with the smallest expected departure time. These aggregation can be visualised as a single *packet train* whereby properties like *Inter-packet delay*, *Inter-flow packet delay* and *accumulated delay of each packet* can be defined. The aggregation of packets into a service class forms a *packet train* which becomes the input for the core router in which it uses the embedded *packet train* information to impose QoS guarantees for each *packet train*.

QoS guarantee using EVTRS can be achieved by reserving a specified rate (assuming that LR schedulers are used) for each edge queue. For example, an ingress router aggregates ten throughput-sensitive flows that require a total rate of 20 *Mbps*. Intuitively, assuming that all the flows share a common route, the next hop (core router) can reserve 20 *Mbps* for this ingress router. Likewise, over-provisioning of bandwidth for an edge queue can assure delay guarantee using any LR scheduler. Example 1 illustrates the provisioning for a delay-sensitive flow.

Example 1 *To make reservation for a flow that visits five hops to pass through a domain in 50 ms, the ingress router will delay each packet of this flow for $(50 - 4 \times 1)$ ms as each of the remaining four routers guarantees a 1 ms delay for each packet from the ingress router.*

Inter-flow packet delay is a crucial parameter for a *packet train* as it implicitly tells core router the inter-packet delay of the packet from different flows of the same service class. Accumulated *Inter-flow packet delay* with reference to the packet at the head of the *packet train* gives the ideal virtual time that a packet is expected to be delivered. In addition, the *accumulated delay of each packet* measures the extra delay or expedition incurred by a packet along its route. EVTRS allows core routers to shuffle its packet forwarding sequence between *packet trains* to minimise deviation of inter-packet delay due to cross traffic, thus

reducing jitter; and delay or expediting packets to minimise the end-to-end delay of packets.

With *Inter-packet delay*, EVTRS uses a *virtual clock technique* to filter packets of a specified flow from its *packet train*. This extraction of flow allows EVTRS to provide a granular drop for rate guarantee service class that can ensure a flow's loss rate¹ or loss size² using a proposed drop algorithm. In addition, the *virtual clock technique* also provides instantaneous service rate for a queue with a backlog, which can be useful for schedulers that use adaptive bandwidth allocation.

EVTRS provides core routers with these information without having to keep per-flow states thus maintaining the stateless core property of SCORE. Core routers only need to keep a couple of states to maintain this time reference system.

3.1.2 Multi-service Scheduling

Quantum-based schedulers like WRR and DRR schedulers are well-known for providing near-perfect throughput fairness and isolation of flows but they do not provide delay differentiation and the delay guarantees provided are bounded by the LR relation. We proposed Weighted Deficit Earliest Departure First (WDEDf) [46] scheduler, a quantum-based scheduler, that is able to provide both rate and delay differentiation. In addition, it is able to provide delay guarantee for both LR-coupled and LR-decoupled reservations; while maintaining rate guarantees for each flow. Delay guarantees are enabled with proper selection of quantum and passing a schedulability check.

In Chapter 5, we show that delay differentiation can be achieved using a Head-Of-Line (HOL) earliest departure time forwarding scheme. To support multi-services, we define a set of services and propose a forwarding scheme based on the HOL earliest departure time forwarding scheme to packet delivery. Particularly, we aim to support four basic QoS service classes. They are (1) Time-Critical (TC) class that maintains a specified delay that can

¹Loss rate refers to the maximum number of loss per time interval

²Loss size refers to the maximum number of consecutively lost packets

be coupled or decoupled from the reservation rate of each queue; (2) Jitter-Sensitive (JS) class that delivers packets as close as possible to their specified departure time; (3) Rate-Based (RB) class that maintains a specified rate for each queue and; (4) Best-Effort (BE) class that aggregates all BE packets into a single queue from all ingress routers. Each edge router's service class is managed as a queue at each core router (see Fig. 3.1). We proposed a forwarding sequence to support these service classes. In each service class (except BE class) the HOL packet with the earliest departure time is selected³ to compete with packets from other service classes. A TC packet or a JS packet that has missed or at its departure time is always forwarded first whereas a RB or BE packet is forwarded before a JS packet if and only if the expected completion time of the RB or BE packet is less than the expected departure time of the JS packet.

3.1.3 Signalling and Centralised Reservation

We assume that a centralised management control mechanism [47], like BB or Oracle, is used to make bandwidth reservation on the SCORE network architecture. Each SCORE domain has an assigned BB for making decisions on bandwidth reservation requests; interacting with BBs of neighbouring domains; and managing routers in its domain. We developed Path and Oracle Discovery Protocol (PODP) [48], a generic signalling protocol, to facilitate the management of networks with centralised bandwidth reservations. It is a generic reactive protocol that identifies routers and neighbouring BBs that are involved in provisioning any bandwidth reservations. The flow of PODP messages is shown in Fig. 3.2. A QoS-sensitive flow generates small non-fragmented datagram (*Discovery Message*) that triggers routers to inform their BB (using a *Reporting Message*) of the path state information, which also includes the IP Address of BB in the upstream domain that the datagram had traversed. A *Notifying Message* is used to inform the BB in the previous domain so

³When there exist more than one packet with the same departure time, a packet is chosen arbitrary.

that an inter-BB communication protocol can be established. PODP is adaptable to existing QoS architectures like DiffServ and IntServ. Centralised reservation decouples control plane from data plane. This further improves the scalability of the SCORE.

Figure 3.2: An illustration showing the flow of PODP messages in a single SCORE Domain.

3.2 Simulation

3.2.1 Simulation Settings

In this section, we conduct simulations using Network Simulator Version 2 (ns2) [49] to investigate the performance of EVTRS in supporting flows with diverse QoS requirements. The network topologies used for our simulations are depicted in Fig. 3.3 and Fig. 3.4, which are referred to as *parking-lot* and *chain* network, respectively. In these networks, we denote an arbitrary ingress router i as I_i , an arbitrary core router j as C_j and an arbitrary egress router k as E_k . Each ingress router contains two functional blocks: an edge conditioner and a scheduler that implements WEDF scheduling discipline (see Fig. 3.1); while each core router contains only a scheduler that implements WEDF scheduling discipline.

Figure 3.3: A parking-lot network

Figure 3.4: A chain network

These networks are used to show the effect of cross traffic on the QoS guaranteed flows. The *parking-lot* network injects traffic incrementally along the path undertaken by the set of flows (*Flow Set A*) that is used for evaluation; while the *chain* network injects and extracts traffic on each hop along the path undertaken by *Flow Set A*. For all

network configurations, each link’s capacity is set to 90% utilisation; buffer size is assumed infinite; and propagation delay is assumed negligible. In the ingress routers, each flow makes a reservation at their specified rate and can be regulated by a traffic conditioner (see Table 3.2). In the core routers, the reservation rate of each edge’s service class is equal to the sum of reservation rate of flows from the service class that traverse the core router.

Table 3.1: Traffic patterns for a flow used for simulation studies

Traffic name	Traffic Type	Bucket Size (σ)	Mean Rate (r)	Peak rate (ρ)	Minimum Packet Size	Maximum Packet Size (M)	Load
exp1	EXPOO	192	16	64	-	96	0.4
exp2	EXPOO	1152	128	512	-	1152	0.4
exp3	EXPOO	2560	256	1024	-	1024	0.4
poisson1	Poisson	-	16	-	64	1500	-
poisson2	Poisson	-	128	-	64	1500	-
poisson3	Poisson	-	256	-	64	1500	-
poisson4	Poisson	-	912	-	64	1500	-

EXPOO refers to Exponential On/Off traffic

Table 3.2: Traffic profiles for a flow set used for simulation studies

Flow Number	Traffic Type	Reserved Rate (R) in <i>kbps</i>	Edge-to-edge delay (<i>ms</i>)	Dual-token Bucket Filter	Comments
TC-1	exp1	16	52	(σ, r, ρ, M)	LR-decouple Flow
TC-2	exp2	512	58	-	EF/GS
TC-3	exp3	256	72	(σ, r, ρ, M)	Mean Rate
JS-1	exp1	16	52	(σ, r, ρ, M)	LR-decouple Flow
JS-2	exp2	512	58	-	EF/GS
JS-3	exp3	256	72	(σ, r, ρ, M)	Mean Rate
RB-1	poisson1	16	-	-	-
RB-2	poisson2	128	-	-	-
RB-3	poisson3	256	-	-	-
BE	poisson4	760	-	-	Malicious Flow

There are five similar flow sets labelled as *Flow Set A*, *Flow Set B*, *Flow Set C*, *Flow Set D* and *Flow Set E*. Flows are generated from a source node in a flow set to a sink

node in their respective egress router. Each flow set contains ten flows comprising a BE flow and three flows each from the other service classes (TC, JS and RB, respectively). The flow profiles for a flow set are tabulated in Table 3.2 and traffic pattern used by each flow is listed in Table 3.1. The fields marked as “-” means that the corresponding parameter setting is not applicable. The traffic type *EXPOO* is generated as *exponential on/off sources* and is regulated by Dual-TBF with an infinite buffer to hold any out-of-constraint traffic of the flows (i.e., out-of-constraint traffic is shaped instead of dropped). Poisson traffic is generated with varying packet sizes that are uniformly generated between 64 *bytes* to 1500 *bytes*.

The above simulation settings are used on both the parking-lot and chain network configurations. To provide some context for comparison, we conduct the same set of simulations with network configurations that follow the DiffServ and IntServ definitions. Although DiffServ and IntServ have been proposed, they are still not fully defined and quantified. Hereby, we set the following criteria for comparison purpose :

- Link’s capacity, buffer size and propagation delay; traffic conditioners and reservation rate of each flow remains the same for both DiffServ and IntServ network configuration.
- For IntServ network configuration, each flow makes a reservation of its specified rate on each router it traverses.
- For routers in the DiffServ network configuration, the reservation rate of each service class is equal to the sum of reservation rate of flows from the service class that traverse the router.
- The DiffServ network configuration defines four service classes, namely, TC, JS, RB and BE service class.

- All routers implement the WFQ scheduling discipline for both DiffServ and IntServ network configurations.

In particular, with reference to Table 3.2, Flow TC-1 and JS-1 make LR-decoupled reservations as they both require a delay of 12 *ms* with a reservation of 16 *kbps*; Flow TC-1 flow tests the LR-decoupled feature of the Time-Critical service of EVTRS and Flow JS-1 flow tests the LR-decoupled feature of the Jitter-Sensitive service of EVTRS which provides jitter minimisation. For IntServ, these flows are viewed as flows with a queuing delay that is bounded by their respective allocated rate. Likewise, for DiffServ, these flows use the EF PHB whereby they are queued into their respective service class.

Flow TC-2 and JS-2 make LR-coupled reservations at their peak rate. These flows can be described as GS flows for IntServ whereby over-provisioning of bandwidth is given to cater for traffic bursts. Likewise, these flows can be described by flows that use EF PHB which is designed to achieve small packet queuing delay throughout a DiffServ domain. In EVTRS, Flow TC-2 and JS-2 use the Time-Critical and Jitter-Sensitive service classes respectively.

Flow TC-3 and JS-3 make LR-coupled reservations at their mean rates. Both IntServ and DiffServ architectures will not provide a service guarantee for these flows as over-provisioning of bandwidth is required to cater for their peak rate. EVTRS provides Time-Critical service and Jitter-Sensitive service for Flow TC-3 and JS-3 respectively. They are considered as flows with a latency requirement that is coupled by their respective allocated rate.

3.2.2 Simulation Results

Using the criteria described in Section 3.2.1, we run a simulation and use a time frame of 200 *s* to compare the delay, jitter and rate achievable for each QoS-sensitive flow which we will describe in the following sections.

Edge-to-edge delay

Fig. 3.5 and Fig. 3.6 show the worst-case edge-to-edge delay for Flow Set A. The edge-to-edge delay is taken from the point where a packet enters the scheduler at the ingress router and exits the scheduler at the egress router. The delay incurred by the traffic regulators is excluded in this presentation as we focus on the delay incurred in the network domain with regulated traffic sources. However, the delay incurred by a traffic conditioner, if any, should be included in an end-to-end delay computation.

For the parking-lot network topology, all TC flows and JS flows maintain their edge-to-edge worst-case delay within their theoretical worst-case edge-to-edge delay as shown in Fig. 3.5. EVTRS provides Flow TC-A-1 and JS-A-1 with a delay that is decoupled from its reservation rate. This cannot be achieved in DiffServ and IntServ in this simulation as both architectures implement the WFQ scheduler which couples delay to reservation rate. Over-provisioning of bandwidth for Flow TC-A-1 and JS-A-1 can achieve their specified delay guarantee but it results in a wastage of resources.

As both Flow TC-A-2 and JS-A-2 made a reservation at their peak rate, the IntServ and DiffServ guaranteed their delay requirements. EVTRS also provides these flows a delay within their specified latency bound.

Finally, Fig. 3.5 shows that only Flow TC-A-3 managed to meet its delay requirement. Flow JS-A-3 failed to meet its delay requirement for both IntServ and DiffServ. This result is expected as both IntServ and DiffServ can only provide a bursty flow a guaranteed service if and only if the flow makes a reservation at its peak rate (i.e., over-provisioning of bandwidth). The same results hold for the chain network topology, which is presented in Fig. 3.6. This shows that EVTRS is able to continue providing its service guarantees even with increasing amount of cross traffic.

Figure 3.5: Edge-to-edge worst-case delay for TC and JS flows of Flow Set A in parking-lot network

Figure 3.6: Edge-to-edge worst-case delay for TC and JS flows of Flow Set A in chain network

Figure 3.7: Normalised Inter-packet Arrival Delay for JS flows of Flow Set A at node E1, plotted with 99% confidence interval in parking-lot network

Figure 3.8: Normalised Inter-packet Arrival Delay for JS flows of Flow Set A at node E1, plotted with 99% confidence interval in chain network

Jitter

The average inter-packet arrival delay is computed at node E1, i.e., at the egress of the network, for each JS flow. In addition, the variance of this delay is computed at 99% confidence interval. To ease the comparison for the amount of jitter for each flow, we normalise the average inter-packet arrival delay and its variances. Fig. 3.7 and Fig. 3.8 show this normalised result for each JS flow in both the parking-lot and chain network.

EVTRS shows that explicitly forwarding JS packets near their departure time reduces jitter for JS flows. On the other hand, both DiffServ and IntServ do not attempt to reduce jitter, thus, resulting in a higher variance as compared to EVTRS.

Throughput

Table 3.3: Throughput (kbps) of flows from Flow Set A at node E1

Parking-lot network										
Flow	TC1	TC2	TC3	JS1	JS2	JS3	RB1	RB2	RB3	BE
DiffServ	16.00	511.95	256.04	16.00	511.95	256	15.27	127.12	255.01	762.46
IntServ	16.00	511.95	256.04	16.00	511.95	256	15.27	127.12	255.01	762.46
EVTRS	16.01	511.95	256.04	16.00	511.94	256.04	15.60	128.12	255.33	760.77
Chain network										
Flow	TC1	TC2	TC3	JS1	JS2	JS3	RB1	RB2	RB3	BE
DiffServ	16.00	511.95	256.04	16.00	511.95	256	15.27	127.12	255.01	762.42
IntServ	16.00	511.95	256	16.00	511.94	256	15.27	127.10	255.01	762.46
EVTRS	16.01	511.95	256.04	16.00	511.90	256.04	15.60	128.12	255.33	760.73

Table 3.3 shows the throughput of each flow in Flow Set A for both the parking-lot and chain network topologies at node E1 (i.e., at egress of the network) within the 200 s time frame used for evaluation. From the table, we see that DiffServ, IntServ and EVTRS provide protection for each flow from the malicious BE traffic and the bursty TC1, TC3, JS1 and JS3 flows, thus achieving some degree of rate guarantees. There are some very small variations in the throughput obtained with DiffServ, IntServ and EVTRS. This can

be due to the aggressiveness of the bursty traffic which was delivered first before the RB (Poisson) flows.

Most importantly, this result shows that EVTRS provides throughput guarantees to RB flows even though it provides delay guarantees that are decoupled from bandwidth allocation to time-critical flows and minimises jitter for jitter-sensitive flows.

3.3 Summary

Our edge-based QoS architecture builds on the the SCORE network model and uses DPS technique to embed state information at edge routers for downstream core routers.

Our architecture defines a set of service class that includes TC, JS, RB and BE service classes. TC service class guarantees a delay that is suitable for GS model defined in IntServ. JS service class guarantees a delay with low jitter that is suitable for EF model defined in DiffServ. In addition, both TC and JS supports a delay guarantee that is decoupled from its reserved rate which can be used for real-time control applications and network games. RB service class provides rate guarantees which is are suitable for applications like bulk data transfer applications.

Per-flow reservations are made at the edge router and flows are classified into one of the service class in which packets from these flows are aggregated at the edge routers to form a *packet train*. In the core of the network, routers guarantee QoS for each edge router's service class whereby scheduling decisions are made based on state information embedded in each packet. Thus, core routers do not keep per-flow state information. This makes our architecture a core-stateless architecture.

In this chapter, we provide a brief description of the main components of our edge-based QoS architecture which includes PODP, a generic signalling protocol, that aids in triggering QoS reservation while recording routes taken by flow and discovers peer BBs; EVTRS, an edge-based virtual time reference system, that treats each service class at the

edge router as a *packet train* whereby properties of this *packet train* are embedded into each packet for core routers to make scheduling decisions and; WDEDF, a scheduling discipline, that is able to provide delay guarantee, minimise jitter, provide rate guarantee and ensure flow protection. For delay guarantee, it is able to make LR-coupled and LR-decoupled reservations.

Finally, using simulation, we show that our architecture is able to provide edge-to-edge delay guarantee for TC flows (LR-coupled and LR-decoupled), minimise jitter for JS flows and provide throughput guarantee for RB flows.

Chapter 4

Edge-based Virtual Time Reference System

We have introduced Edge-based QoS Architecture and showed that QoS can be achieved using EVTRS and WDEDF scheduler in Chapter 3. In this chapter, we present the detailed analysis of EVTRS and show that it is able to provide essential per-flow and per-aggregate state information for stateless core routers.

The rest of this chapter is organised as follows. In the next section, we provide the motivation behind the need for an edge-based information reference system for the Edge-based QoS Architecture. Section 4.2 outlines the information infrastructure of EVTRS. It provides details on how information is assembled in EVTRS. Section 4.3 details how information is extracted in EVTRS. It also shows a *virtual clock* technique using the extracted information to generate tools for extracting a flow and measuring instantaneous arrival rate for a queue. The mapping of QoS-sensitive flows into EVTRS is described in Section 4.4. We prove the convergence of EVTRS mathematically in Section 4.5; and illustrate the characteristics of the EVTRS algorithm using examples in Section 4.6. Finally, we discuss the scalability of EVTRS in Section 4.7; and summarise this chapter in Section 4.8.

4.1 Introduction

Current research shows that aggregating traffic into agreed service classes, like DiffServ, and reserving bandwidth for these classes are more scalable than performing and maintaining per-flow bandwidth reservation, like IntServ, on routers between two end points. However, traffic aggregation results in loss of important traffic information as packets in the core network are only identified by their service class. We introduce EVTRS for the Edge-based QoS Architecture to provide core schedulers with *per-flow*¹ and *per-queue*² information.

In our architecture, an edge router performs admission control on arriving QoS-sensitive flows. Admission control on these flows includes identifying packets, policing of flow rate and marking on packets (i.e., grouping flows into predefined service classes). In the core of the network, marked packets will receive their service according to their designated service class. This edge router also determines the delivery order of packets from its QoS flows and aggregates these packets into a single train of packets (packet train³) in ascending delivery order based on their departure time at its output queue. This packet train is the incoming traffic for the first core router and subsequent core routers, assuming that all flows take the same route. The core routers ensure a predefined QoS guarantee for incoming traffic from each edge router.

EVTRS maintains and updates *per-flow* and *per-queue* state information for each edge queue in a distributed manner. It provides core routers with this information without having to keep per-flow states. Routers only need to keep a fix number of states to maintain EVTRS. The *per-flow*, *per-queue* information can be derived using the embedded EVTRS states whenever necessary.

¹Here, a flow can be either an user flow generated by an Internet application or an aggregated traffic consisting of multiple user flows.

²Here, a queue refers to a backlog of packets aggregated from different flows but of the same service class.

³Jain et al. have used a packet train model [50] for traffic measurements on ring local area computer network. The EVTRS uses a packet train technique to embed information at edge of a network domain whereby these information are decoded in the core of the network domain.

In summary, EVTRS is a relative time reference system which works in a distributed manner whereby timing information of packets are embedded at the ingress router and this information is extracted by the core routers. The expediting or delaying (time deviation) of each packet caused by each core router is also embedded for adjustment of departure time in the next core router. A core router can pick on any packet to be the reference packet of a new packet train as information embedded in the next packet is sufficient to inform the scheduler the inter-packet delay between the newly arrived packet and the reference packet.

4.2 EVTRS Architecture

Figure 4.1: Packet trains in Edge and Core routers with multiple queues

This section outlines the basic reference structure of EVTRS. EVTRS takes a different viewpoint in providing time references. Edge routers have long been identified as the bottleneck of a domain while core routers usually have high capacity. Thus, in EVTRS, QoS provisioning is performed at each core router based on requirements from each edge router instead of a set of predetermined Class-of-Service. Flows from each ingress edge

router are aggregated and forwarded down a tree of core routers to their designated egress edge router. A *packet train* consisting of aggregated packets is formed implicitly at the output queue of the ingress router. The tree of core routers splits this *packet train* into shorter *packet trains* while maintaining the timing properties of the *packet train*. Therefore, in a network of N edges, each core router will maintain $(N \times M) + 1$ queues for each interface if each edge router maintains M queues and a Best-Effort queue.

Figure 4.2: Timing relations between per-flow, per-queue and per-link *packet trains* in which $\widehat{T}_D^{k-1} = \widehat{T}_D^{m-1}$ and $\widehat{T}_D^k = \widehat{T}_D^{n-1}$, where $\widehat{T}_D$ is the expected departure time, $\widetilde{T}_D$ is the actual departure time of a packet, m identifies a packet in Flow D1, n identifies a packet in Flow D2 and k identifies a packet in Queue q .

With this queuing concept, we analyse how QoS-sensitive flows are aggregated at the edge routers and traffic information are carried across the network domain to facilitate QoS provisioning. In our analysis, we have modelled EVTRS using a system model that consists of multiple queues. This model classifies packets into different queues. At the ingress of the network, packets are classified and aggregated into the different queues based on their QoS requirements. For example, a possible scheduling scheme (see Fig. 4.1) classifies packets into Time-Critical (TC), Jitter-Sensitive (JS), Rate-Based (RB) or Best-Effort (BE). All notations used are summarised in Table 4.1. We will use the following notations: i refers

to a flow i , q refers to a queue q on a node j , e refers to a referenced edge node.

Table 4.1: Table of notations used in EVTRS

Notation	Description
$\widehat{T}_{D,j,(e,q)}^k$	Expected Departure Time of packet k at node j from queue q of edge router e . It refers to the ideal departure time with reference to the edge router e .
$\tilde{T}_{D,j,(e,q)}^k$	Departure Time of packet k at node j from queue q of edge router e . It refers to the departure time that is used for scheduling in router. This timing is adjusted by the δ .
$\widetilde{T}_{D,j,(e,q)}^k$	Actual Departure Time of packet k at node j from queue q of edge e . It refers to the time that packet k gets its services at node j .
$\widetilde{T}_{A,j,(e,q)}^k$	Actual Arrival Time of packet k at node j from queue q of edge e . It refers to the time that packet k arrives at the node j .
$IPD^{i,k}$	Inter-packet Departure Time of packet k and packet $k - 1$ of flow i .
$IFD_{(e,q)}^k$	Inter-flow Departure Time of packet k and packet $k - 1$ of an aggregated queue q from edge router e .
$\delta_{(e,q)}^k$	Packet Lag Time of packet k of an aggregated queue q from edge router e .
$\chi_{(e,q)}^y$	An EVTRS update state for an aggregated queue q from edge router e at interface y .
D_{edge}^i	Delay to be imposed to flow i at edge router.
D_{domain}^i	Maximum delay, specified by flow i , for a domain.
$D_{j,(e,q)}$	Delay guaranteed for aggregated queue q from edge router e at node j .
$L^{MAX,i}$	Maximum packet size of flow i .
$L^{i,k}$	Size of packet k of flow i .
R_i	Reserved rate of flow i .

To analyse this scheme, we will first review the timing diagram of a *packet train* (Fig. 4.2) for packet relationship of a single flow, an aggregation of traffic containing flows that are classified into the same queue (i.e., same service class) and an aggregation of traffic from different queues on a link. In Fig. 4.2, each packet of a flow is given an eligible time that is computed by the scheduler based on the flow's QoS requirements. This eligible time represents the expected departure time, $\widehat{T}_D$, of the packet at the scheduler. The difference between the two packets' $\widehat{T}_D$ is the Inter-packet Departure Time (IPD). The IPD of packet k from flow i is computed using Eqn. 4.1.

$$\begin{aligned}
 IPD^{i,1} &= 0 \\
 IPD^{i,k} &= \widehat{T}_D^{i,k} - \widehat{T}_D^{i,k-1}
 \end{aligned} \tag{4.1}$$

A per-queue *packet train* is formed when flows are aggregated together into a queue in ascending order of packet's $\widehat{T}_D$. The difference between two packets' $\widehat{T}_D$ in this queue is the Inter-flow Departure Time (IFD). The IFD of packet k from queue q at edge e is computed using Eqn. 4.2.

$$\begin{aligned}
 IFD_{(e,q)}^1 &= 0 \\
 IFD_{(e,q)}^k &= \widehat{T}_{D(e,q)}^k - \widehat{T}_{D(e,q)}^{k-1}
 \end{aligned} \tag{4.2}$$

Since traffic from different queues is aggregated in ascending order of $\widehat{T}_D$, this forms an ordered per-link *packet train* for delivery to the next hop. Ideally, $\widehat{T}_D$ is the same as $\widehat{T}_D$ but this is not always true due to the various possible variances, e.g. due to cross traffic. Therefore, during transmission, each packet can be delayed or expedited. The difference between the actual departure time and the expected departure time of a packet is the Packet Lag Time (δ). δ of packet k in queue q from edge e at node j is computed using Eqn. 4.3.

$$\delta_{j,(e,q)}^k = \widetilde{T}_{Dj,(e,q)}^k - \widehat{T}_{Dj,(e,q)}^k \tag{4.3}$$

A positive δ signifies that the packet is serviced late; while a negative δ signifies that the packet is serviced earlier than its $\widehat{T}_D$. This δ can be used to adjust $\widehat{T}_D$ of a packet in a coordinated networking [51] environment. On the other hand, δ can also act as an input to a packet-dropping schemes. In some applications, packets that cannot meet their desired QoS are considered useless to the application.

The key contribution of EVTRS is to maintain per-queue *packet train* using *IFD* of packets. For per-flow *packet train*, the *IPD* of each packet is computed by the ingress router. These terms can be embedded into the packet if required by core routers to perform advance functions. For per-queue *packet train*, the *IFD* of each packet is computed by the ingress router and embedded into the packet. The δ term of a packet is computed by both edge and core routers and embedded into the packet. It serves as a service monitoring term and as an accumulative adjustment term for adjusting the $\widehat{T}_D$ of this packet at edge and core routers. Thus, EVTRS has, at most, three terms (*IFD*, *IPD* and δ) to be embedded into a packet for basic operations. In fact, only *IFD* is required to maintain the *packet train* property.

In a core router with multiple output links, a per-queue *packet train* will be split into child *packet trains*. Each child *packet train* has its own characteristics but it is correlated to its parent. To maintain this co-relationship (see Fig. 4.1), a χ state is kept for each edge queue for its interfaces. When packet k of edge queue q is designated for an interface y of its set of all interfaces, $\mathbb{Z}$, the $IFD_{(e,q)}^k$ is added to the χ of edge queue of all interfaces except for interface y (see Eqn. 4.4). The $IFD_{(e,q)}^{k+1}$ of the next packet ($k + 1$) is added by χ of the edge queue from the interface that it is routed and χ is reset to zero.

$$\begin{aligned}
 IFD_{(e,q)}^{k+1} &= IFD_{(e,q)}^{k+1} + \chi_{(e,q)}^y \\
 \chi_{(e,q)}^y &= 0 \\
 \forall x, x \in \mathbb{Z}, x \neq y, \chi_{(e,q)}^x &= \chi_{(e,q)}^x + IFD_{(e,q)}^k
 \end{aligned} \tag{4.4}$$

To remove packet k from a *packet train* when packet k is waiting in its edge queue or at the head of its edge queue, the $IFD_{(e,q)}^k$ is added to the $IFD_{(e,q)}^{k+1}$ of packet ($k + 1$). When packet k is at the tail of its edge queue or the edge queue is empty, the $IFD_{(e,q)}^k$ is added to the χ of the edge queue and this χ will be added to the $IFD_{(e,q)}^{k+1}$ of the next arrived

packet $(k + 1)$. In addition, other critical information like edge identifier, rate, delay, jitter variance, loss rate and loss size can also be embedded. Embedding information into packets can exploit the techniques used in DPS or MPLS. In DPS, information is fitted into the IP packet using unused bits. MPLS adds a new layer to the existing protocol stack to embed required information to be carried across its network domain.

Thus, ingress routers are responsible for computing $\widehat{T}_D$, IPD , IFD and δ of packets. A packet's $\widehat{T}_D$ is dependent on its QoS requirement of its flow (see Section 4.4). Core routers are responsible for computing a departure time $\tilde{T}_D$ using the embedded EVTRS information (see Section 4.3) and updating of the IFD and δ for each packet.

4.3 Extraction of EVTRS Information

The embedded EVTRS information can be used to reconstruct the *packet trains* at the core routers. In addition, per-flow and per-queue information can be retrieved for computing a departure time ($\tilde{T}_D$) for each packet at a core router. We will illustrate how the retrieved information can be processed to generate important parameters for designing advance functions in routers.

4.3.1 Computing the Departure Time of packet for Core Routers

The main task of the core router is to compute the $\tilde{T}_D$ (see Eqn. 4.5), of a packet using the previous packet's $\widehat{T}_D$ (see Eqn. 4.6) of the same edge queue, where $D_{j,(e,q)}$ is the delay guarantee on node j for queue q from edge e , and $IFD_{(e,q)}^k$ is the value embedded in packet k before the packet left the edge router e .

$$\begin{aligned} \tilde{T}_{D_{j,(e,q)}}^k &= \widehat{T}_{D_{j,(e,q)}}^k - \delta_{j-1,(e,q)}^k \\ \tilde{T}_{D_{j,(e,q)}}^1 &= \widehat{T}_{D_{j,(e,q)}}^1 \end{aligned} \tag{4.5}$$

$$\begin{aligned}
\widehat{T}_{Dj,(e,q)}^k &= \widehat{T}_{Dj,(e,q)}^{k-1} + IFD_{(e,q)}^k \\
\widehat{T}_{Dj,(e,q)}^1 &= \widehat{T}_{Aj,(e,q)}^1 + D_{j,(e,q)} - \delta_{j-1,(e,q)}^1
\end{aligned} \tag{4.6}$$

For each packet, the core router also needs to maintain the value for δ . This term measures the accumulated delay and expedition of a packet received on each hop; and it can be used to minimise delay deviation caused by previous hops as shown in Eqn. 4.5. To maintain the δ term, a core router must update this term as shown in Eqn. 4.7.

$$\begin{aligned}
\delta_{j,(e,q)}^k &= \widehat{T}_{Dj,(e,q)}^k - \widehat{T}_{Dj,(e,q)}^{k-1} + \delta_{j-1,(e,q)}^k \\
\delta_{j,(e,q)}^1 &= \widehat{T}_{Dj,(e,q)}^1 - \widehat{T}_{Dj,(e,q)}^0
\end{aligned} \tag{4.7}$$

4.3.2 Relative Virtual Clock

Based on the state information embedded in EVTRS, *packet filtering* for a flow and *arrival rate estimation* for a queue in a core router can be achieved using a method call Relative Virtual Clock (RVC). In addition, we will also illustrate another usage of RVC in a dropping algorithm in Section 4.4.4.

Core routers can keep a RVC for any edge queue at any moment with low computation cost. On the arrival of an arbitrary packet k , RVC is set to zero. The RVC updates on the arrival of a new packet for the edge queue. The RVC for edge queue q at node j is updated based on the *IFD* of the newly arrived packet k as shown in Eqn. 4.8.

$$RVC_{j,(e,q)} = RVC_{j,(e,q)} + IFD_{j,(e,q)}^k \tag{4.8}$$

Packet Filtering for a Flow

Using RVC, core routers can identify whether a newly arrived packet belongs to a particular flow. Upon receiving packet k of flow i , the RVC arrival time of packet

$(k+1)$, $A_{RVC}^{i,k+1}$, is computed using Eqn. 4.9. The RVC arrival time for each arrived packet is compared to $A_{RVC}^{i,k+1}$. If the comparison is equal then the arrived packet belongs to flow i .

$$A_{RVC}^{i,k+1} = A_{RVC}^{i,k} + IPD_{j,(e,q)}^{i,k+1} \quad (4.9)$$

Arrival Rate Estimation for a Queue

Another application of RVC is to compute a arrival rate, R^{RVC} , for any edge queue. This arrival rate is the sum of packet length over a RVC interval as shown in Eqn. 4.10, where L is the packet length of each arrived packet. This arrival rate is also the instantaneous service rate required by the edge queue.

$$R_{j,(e,q)}^{RVC} = \frac{\sum_{\eta=k}^{k+\tau-1} L_{j,(e,q)}^{\eta}}{RVC_{j,(e,q)}^{k+\tau} - RVC_{j,(e,q)}^k} \quad (4.10)$$

4.4 Mapping flows to EVTRS

In this section, we illustrate a possible mapping of EVTRS to QoS-sensitive flows based on the service classes defined in Section 4.3. Without losing generality, apart from BE flows, QoS-sensitive flows are classified into three QoS types, namely TC, JS and RB. The mapping of these flow types to the admission control of EVTRS is described in Section 4.4.1, Section 4.4.2 and Section 4.4.3 respectively. To support controlled packet losses, we also introduce a dropping scheme (see Section 4.4.4) using RVC that ensures *loss rate* and *loss size* of any QoS-sensitive flow.

4.4.1 Mapping Time-Critical flow

Real-time interactive applications, like Internet Telephony and video conferencing, are delay sensitive applications. They require strict or predictive packet delivery delay at the receiving hosts to an extent that the delay inherent in jitter buffer is not acceptable

by these applications. In addition, Park et al. [52] have found that the round trip latency threshold for human to detect performance degradation is approximately 200 *ms*. This shows that for interactive applications, each packet must be delivered to its destination in approximately 100 *ms* if it has made a bidirectional QoS reservation.

In EVTRS, we assume that per-domain delay, D_{domain}^i , for flow i can be computed using the end-to-end delay, D_{e2e}^i , specified by the flow. We denote $D_{j,(e,q)}$ as the delay guarantee for edge queue q at core router j . Using any signalling protocol, like PODP, Bandwidth Broker Discovery Protocol (BBDP) [53], RSVP or Yet another Sender Session Internet Reservations (YESSIR) [54] [55], we can determine the set of hops⁴, H , used by the flow to cross the domain and compute the sum of worst-case delay for flow i in the core of the network. Therefore, a delay of D_{edge}^i at the ingress router e for flow i can be approximated using Eqn. 4.11.

$$D_{edge}^i = D_{domain}^i - \sum_{j=1}^H D_{j,(e,q)} \quad (4.11)$$

$$\widehat{T}_{D(e,q)}^{i,k} = \widehat{T}_{A(e,q)}^{i,k} + D_{edge}^i \quad (4.12)$$

At the ingress router, the $\widehat{T}_D$ for packet k of flow i is computed using Eqn. 4.12. After that, the packet is scheduled for delivery and before it is forwarded to its next hop, its δ is computed using Eqn. 4.3. At a core router j , the $\widehat{T}_D$ of each packet k is computed using Eqn. 4.5.

4.4.2 Mapping Jitter-Sensitive flow

Jitter is the measure of the variation between packet arrival interval. Audio and video streaming are applications that are sensitive to jitter. These applications can have an

⁴The signalling protocols used here must provide refreshing mechanism to detect path variation due to changes in routing decision. When there is a change in the path used by the designated flow, a reestablishment of QoS must be made.

average delay of 20 *ms* or 30 *ms* as long as the transit time is constant. An example of large variation in inter-packet arrival time, is a video streaming with some packets taking 20 *ms* and others taking 30 *ms* to arrive at the receiver. This gives an uneven quality of sound or video. For some applications, like Video-On-Demand, the effect of jitter is minimised using a jitter-buffer which improves user-perceived QoS. Therefore, jitter effect can be bounded by maintaining the inter-arrival time between packets within a given variance.

In EVTRS, the ingress router computes the $\widehat{T}_D$ for each packet k of flow i using Eqn. 4.13. After that, the packet is scheduled for delivery and before it is forwarded to its next hop, its δ is computed using Eqn. 4.3.

$$\begin{aligned}\widehat{T}_D^{i,1} &= \widehat{T}_A^{i,1} + \frac{L^{MAX,i}}{R^i} \\ \widehat{T}_D^{i,k} &= \widehat{T}_A^{i,k-1} + \frac{L^{MAX,i}}{R^i}\end{aligned}\tag{4.13}$$

In addition, for application with a jitter-buffer, a maximum variance term, γ , can be embedded by the application whereby this term can act as a numerical check condition computed using Eqn. 4.14 for packet dropping.

$$\gamma_{j,(e,q)}^{i,k} \leq \widehat{T}_{Dj,(e,q)}^k + \delta_{j-1,(e,q)}^{i,k}\tag{4.14}$$

4.4.3 Mapping Rate-Based flow

Electronic mail and file transfer applications are neither sensitive to delay nor jitter. They can tolerate packets which are delayed by a few seconds. In the ideal situation, RB flows are rate controlled at the ingress of the network using token bucket or leaky bucket depending on whether the flow traffic is consistent or in sudden bursts. In addition, GPS emulated schedulers provide strict bandwidth bounds on its queues. In general, admitted packets can be given an eligible time as computed using Eqn. 4.15, since the arrival time of

the packet is rate-controlled by the admission control mechanisms.

$$\widehat{T}_D^{i,k} = \widetilde{T}_A^{i,k} + \frac{L^{i,k}}{R^i} \quad (4.15)$$

4.4.4 Ensuring Fine Granular Packet Loss Rate and Packet Loss Size

It has been shown that proper *marking* of packets in terms of drop level can achieve goodput ⁵ for multimedia streaming [56]. However, *marking* only provides a priority for dropping. It is unable to bound losses suffered by a flow.

We derive a packet dropping scheme (see Algorithm 1) based on *Loss Rate* and *Loss Size* using RVC. In this context, *Loss Rate* is defined as the maximum number of packet loss per time interval while *Loss Size* is defined as the maximum number of consecutively lost packets.

We can detect packets that belong to the same flow using the packet filtering technique shown in Section 4.3.2. Therefore, a preventive congestion avoidance scheme can be derived. A First In First Out (FIFO) *drop list* is created to track the number of packet drop for flows selected for dropping. This *drop list* stores the RVC arrival time for the last arrived packet and a drop count, sorted in ascending order by $\widehat{T}_D$. When a new packet is dropped, the RVC arrival time is replaced by the arrival time of the dropped packet and the drop count is increased by one. Instinctively, a router can pick an arbitrary flow and start dropping the first packet while maintaining the *drop list*. The *Drop_Condition_Not_Violated()* function in Algorithm 1 simply checks and ensures that dropping the current packet will not violate the specified dropping condition.

However, this can violate the dropping criteria as a downstream router can coincidentally pick the same flow for dropping. A roundabout way for this open loop control problem is to embed a drop indicator, *DI*, to inform downstream routers that this flow has

⁵Goodput is the effective throughput computed using the amount of data successfully received and decoded at the receiver over time.

Algorithm 1 Packet Dropping Algorithm

```

adds flow  $i$  into drop list
...
if arrived packet  $k$  belongs to flow  $i$  then
  if Authorised_To_Drop() and Drop_Condition_Not_Violated() then
    drop packet
    update drop list
  else
    mark packet using  $DI$ 
    forward packet
  end if
end if
...
remove flow  $i$  from drop list

```

been picked for dropping. In addition, the router must wait for an authorised condition as shown in Algorithm 1 (*Authorised_To_Drop()*) before it starts dropping packets.

For a loss rate sensitive flow, the router must wait for the start of the next time interval of this flow before it is authorised to drop any packets. For a loss size sensitive flow, the router must let the specified maximum number of packets from this flow to pass through the router before it is authorised to drop any packets.

4.5 Analytical Results

In this section, we prove the relationship between IPD of two packets and the sum of IFDs between these two packets in a *packet train*. This is an important property for the RVC introduced in Section 4.3.2. Using mathematical induction, we also show that our proposed algorithm is able to maintain the *packet train* property defined in Section 4.2 after the original *packet train* created at the edge router is split into child *packet trains*.

4.5.1 Inter-packet Departure Time

The packet filtering technique introduced in Section 4.3.2 uses the property that the IPD of two consecutive packets of the same flow is the the sum of the IFDs embedded

in the packets between these two packets. Theorem 1 concretes this property.

Theorem 1 *The IPD of any two packets is the sum of IFD between the two packets when they are aggregated with other flows into a queue in ascending order by their $\widehat{T}_D$.*

Proof: Let the number of packets between packet k and packet $(k + 1)$ of flow i be x . Thus, packet k is mapped onto the aggregated queue as packet a and packet $(k + 1)$ is mapped as packet $(a + x)$ in the aggregated queue. From Eqn. 4.2, we summed up the IFD from packet $(a + 1)$ to packet $(a + x)$ and solved it using telescoping series.

$$\begin{aligned}
 IPD^{a+x} &= \sum_{i=a+1}^{a+x} IFD^i \\
 &= \sum_{i=a}^{a+x-1} (\widehat{T}_D^{i+1} - \widehat{T}_D^i) \\
 &= (\widehat{T}_D^{a+1} - \widehat{T}_D^a) + (\widehat{T}_D^{a+2} - \widehat{T}_D^{a+1}) + \dots \\
 &\quad + (\widehat{T}_D^{a+x-1} - \widehat{T}_D^{a+x-2}) + \dots + (\widehat{T}_D^{a+x} - \widehat{T}_D^{a+x-1}) \\
 &= \widehat{T}_D^{a+x} - \widehat{T}_D^a
 \end{aligned}$$

The result of the series shows that the IPD between packet k and packet $(k + 1)$ is equal to the IPD defined in Eqn. 4.1. ■

4.5.2 EVTRS updating algorithm

In this section, we formally define a “desired packet train” property and show that EVTRS conforms to this property.

Definition 1 *A packet train is said to have a “desired packet train” property if and only if the following properties are true :*

Property A *For any two adjacent packet $P(i + 1)$ and $P(i)$ of a child packet train, the IFD of $P(i + 1)$ is the sum of IFD of all the packets between $P(i + 1)$ and $P(i)$ in the original packet train whereby these packets are designated to other child packet trains.*

We consider **Property A** is satisfied if there are less than two packets in the child packet train.

Property B For each state reference value χ^i for interface i , let A be the packet at the front of the current parent packet train and B is the front packet in the current child packet train i (for interface i). χ^i is the sum of IFD of all the packets between A and B in the original parent packet train before B moves to interface i . We consider the sum of IFD is zero for an interface if there are no packets between A and B . We also consider **Property B** is satisfied if the child packet train is empty.

These two properties define that a child packet, formed by splitting of packets randomly between two or more interfaces, does not lose its IFD information.

Theorem 2 EVTRS conforms to the “desired packet train” property as defined in Def. 1.

Proof: Initially, let $\chi^i = 0$ for all i with no child packet train. Clearly the system has the “desired packet train” property as it satisfies the initial conditions for **Property A** and **Property B**. Packets are moved to child packet trains one by one. For any system that has the “desired packet train” property, let us assume that the front of the queue is packet C destined to interface i . Now, we need to move C to child packet train i . The action is taken using Eqn. 4.4 and is re-stated below :

$$\begin{aligned} IFD_{(e,q)}^{k+1} &= IFD_{(e,q)}^{k+1} + \chi_{(e,q)}^y \\ \chi_{(e,q)}^y &= 0 \\ \forall x, x \in \mathbb{Z}, x \neq y, \chi_{(e,q)}^x &= \chi_{(e,q)}^x + IFD_{(e,q)}^k \end{aligned}$$

This proof is complete if we can show that after the move the system still has the “desired packet train” property.

For an interface i , consider,

Case 1 No packet in the child packet train i . Trivial case.

Case 2 At least one packet (call it B) in the child packet train i ; No packets between B and C in the original packet train. Trivial case.

Case 3 At least one packet (B being the last packet) in the child packet train i ; Some packets between B and C in the original packet train.

Before the move, the system has the “desired packet train” property, χ^i is the sum of IFD of packets between B and C . The way IFD of C is updated ensures **Property A** is satisfied. After the move, χ^i is updated to zero. The new last packet in the child packet train immediately follows **Property A**. Thus, **Property A** is satisfied.

For an interface j such that $j \neq i$, consider,

Case 1 No packet in the child packet train j . Trivial case.

Case 2 At least one packet (call it B) in the child packet train j ; No packets between B and C in the original packet train.

Before the move, the system has the “desired packet train” property, χ^j is the IFD of C . χ^j is updated so that **Property B** is satisfied. No change for **Property A** since the child packet train is not changed.

Case 3 At least one packet (B being the last packet) in the child packet train j ; Some packets between A and B in the original packet train.

Before the move, the system has the “desired packet train” property, χ^j is the sum of IFD of packets between B and C . χ^j is updated so that **Property B** is satisfied. No change for **Property A** since the child packet train is not changed.

Therefore, the new system has the “desired packet train” property. By mathematical induction, after every move (from beginning), the system maintains this property. ■

4.6 Numerical Examples

Figure 4.3: Timing Diagram at edge router e which aggregates Flow X , Flow Y and Flow Z .

To provide a deeper insight of EVTRS, we illustrate two numerical examples in this section. We will look at the behaviour of a queue on edge router e that aggregates three flows (Flow X , Flow Y and Flow Z). This queue is passed into a core router c whereby the embedded IFD is used to compute the $\tilde{T}_D$ and δ .

Fig. 4.3 shows the timing diagram of the three flows at edge router e . Each flow has its allocated D_{edge} computed using Eqn. 4.11. The figure shows the actual arrival time of the first packet from each of the three flows. Assume that δ starts at zero, the packet's $\tilde{T}_D$ is computed using Eqn. 4.5. These packets are scheduled and their $\tilde{T}_D$ are shown in the figure. In addition, the shaded area in the figure show the δ of packets that are computed using Eqn. 4.7.

Figure 4.4: Timing Diagram at core router c with $D_c = 2$ in an ideal situation whereby packet's arrival at core router c is equal to the expected departure time ($\tilde{T}_D$) at the edge router e and $\delta = 0$.

In an ideal situation, packets are sent at exactly their $\tilde{T}_D$ at their previous hop

(i.e., edge router e) and their ideal $\widetilde{T}_A$ at core router c is equal to their $\widetilde{T}_D$ at their previous hop. Likewise, the $\widetilde{T}_D$ of these packets at core router c are equal to their computed $\widehat{T}_D$ as shown in Fig. 4.4. This ideal situation serves as a reference model for the subsequent numerical examples.

Numerical example for a non-ideal case

Figure 4.5: Timing Diagram at core router c with $D_e = 2$ in a non-ideal situation whereby packet's arrival at core router c is equal to actual departure time ($\widetilde{T}_D$) at edge router e and δ is taken into account.

Fig. 4.5 shows the timing diagram of the aggregated *packet train* on core router c in a non-ideal situation whereby the arrival time of packets at core router c is equal to the actual departure time ($\widetilde{T}_D$) of packets at the edge router e ; and packets are scheduled in FIFO manner for each queue while the $\widetilde{T}_D$ of each packet is used for inter-queue scheduling. In this example, only the $\widetilde{T}_D$ of packet from Flow Y lags by 2, the delay of packets from Flow X and Flow Z incurred on edge router e are patched up by core router c .

In comparison with the ideal case in Fig. 4.4, the $\widehat{T}_D$ of each packet in the non-ideal case is the same as the $\widehat{T}_D$ in the ideal case.

Numerical example for split node case

Using the same scenario as in Fig. 4.3, let us assume that Flow X and Flow Z are routed to core router i ; and Flow Y is routed to core router j . As shown in Fig. 4.6,

χ^j of core router j is set to 7 according to Eqn. 4.4 upon which the packet from Flow Y is forwarded to core router i . When the packet from Flow Y is forwarded to core router j , the χ^i is set to 6. Finally, the value of χ^i is reset to zero once it is used to update the IFD of the packet arrived from Flow Z.

Figure 4.6: Timing Diagram at core router i and j with $D_e = 2$ in a non-ideal situation whereby packet's arrival at core router i is equal to actual departure time ($\hat{T}_D$) at edge router e and δ is taken into account.

4.7 Discussion

4.7.1 Packet Lost at Medium Access Control Layer

A packet lost at Medium Access Control (MAC) layer can cause inaccurate computations for EVTRS. To resolve this issue, a packet counter can be used to detect packet loss on a link between two routers (Router A and Router B). Router A can embed a packet count (generated in a cyclic manner, for example, 1 to 10), on each packet transmitted to Router B. Router B will be able to detect a packet lost if the packet count in the packet

received does not tally with the packet counting sequence. In such cases, Router B can re-synchronise the *packet train* by treating this packet as the head of a new *packet train*.

4.7.2 Scalability of Core Routers

In this section, we look at the scalability of the core routers. Prior to that, we will identify the state information that are maintained by EVTRS and review the number of queues that will be created on each core router.

To maintain EVTRS, a core router needs to keep a EVTRS update state - χ and the latest Expected Departure Time of the last packet served by the scheduler for each edge queue. For packet classification, each queue is identified by two identifier strings, namely, queue identifier and edge router identifier. Thus, there are four states that must be maintained for each edge queue.

As mentioned in Section 4.2, if there are N edges in the EVTRS network domain, each core router will maintain $N \times M$ queues for each interface if each edge maintains M number of service classes. These service classes are TC, JS, RB, excluding the BE queue. Thus, EVTRS will maintain $3N + 1$ queues whereby that the extra queue is the BE queue where BE traffic aggregates.

In terms of computation, router needs to update the χ value and computes the $\widetilde{T}_D$ and δ of each arrived packet. These operations only require simple mathematical functions. These mathematical functions are executed inline with the scheduling algorithm used in the core routers. The impact of executing these functions does not add to the algorithmic complexity⁶ of the scheduler used for a link.

EVTRS has a higher storage complexity as compared to DiffServ whereby only one queue of each service class is kept and packets are identified by using a single identifier

⁶Algorithmic complexity is generally written in a form known as *Big-O* notation, where the $O(N)$ represents the complexity of the algorithm and a value N represents the size of the set the algorithm is run against.

(DSCP). On the other hand, EVTRS has much lower storage complexity as compared to IntServ where core router creates a queue for each reserved flow and a flow is identified using a multi-field classifier.

The above analysis does not include the usage of RVC. It is clear that the usage of RVC increases the complexity of a core router, but, for each queue, it only adds an additional storage for one RVC state and the operation only involves addition and comparison on this RVC state.

4.7.3 Complexity of Packet Dropping Algorithm

Section 4.4.4 discussed a fine granular drop scheme which requires per-flow management. This per-flow management only includes flows that have been selected by the core routers for drop operation. Using the packet filtering technique described in Section 4.3.2 that triggers the dropping algorithm when the designated packet arrived, the dropping algorithm just needs to perform simple checks to ensure that the specified dropping conditions are not violated. From Algorithm 1, the algorithmic complexity of the packet dropping algorithm is $O(1)$ as there is no loop involved to perform the packet dropping condition checks. Additionally, the storage requirements to manage each chosen flow is low. It requires three states (arrival time, drop count, drop authorisation condition) to be kept.

4.8 Summary

In this chapter, we introduce EVTRS for our Edge-based QoS Architecture. It aims to provide aggregation information and flow information for core routers. EVTRS assumes that packets are classified into predetermined service classes and an expected departure time is computed for each packet from the same flow whereby the difference between the expected departure time of two consecutive packets indicates the inter-packet delay which is embedded into the latter packet.

Flows from the same service class are aggregated and delivered according to their expected departure time into the core of the network. On aggregation, the expected departure time of two consecutive packets indicates the inter-flow delay which is also embedded into the latter packet. In addition, a packet lag term is kept in each packet to reflect the accumulated delay of this packet at each router.

Thus, the aggregation of flows of the same service class forms a *packet train* for the core routers. Using these information embedded in each packet, we are able to extract per-queue and per-flow information from the aggregated packet train. Specifically, each queue is reconstructed at each core node with each packet having an expected departure time that is relative to the ingress router. We have shown that useful functions like flow extraction, instantaneous service rate of a queue and fine granular dropping scheme can be derived using a relative virtual clock technique in conjunction with the embedded information.

We showed through mathematical induction that our proposed algorithm to maintain EVTRS converges and the inter-packet delay is derived from the sum of inter-flow delay between two consecutive packets of the same flow. Most importantly, we also showed that EVTRS is observable using numerical examples. Finally, we addressed the pitfalls and scalability of EVTRS.

Chapter 5

Weighted Deficit Earliest Departure First Scheduler

Quantum-based schedulers like WRR and DRR schedulers are well-known for providing near-perfect throughput fairness and isolation of flows. In this chapter, we discuss the lack of assuring delay differentiation between queues with different reservation rate in traditional quantum-based schedulers. We proposed WDEDF scheduler, a new quantum-based scheduler that is able to provide rate differentiation, delay differentiation; and minimise delay-jitter bound. In addition, it is able to provide reservations with a delay bound that is decoupled from its bandwidth allocation (LR-decoupled¹) for regulated sources.

This chapter is structured as follows. Section 5.1 describes the background on quantum-based schedulers and EDF schedulers. Section 5.2 describes WDEDF scheduler and shows how it can provide QoS guarantees for flows using a schedulability check. Section 5.3 shows the analysis of WDEDF scheduler for its HOL deadline forwarding; fairness; and latency comparison with DRR scheduler. WDEDF scheduler is evaluated in Section 5.4 where we show that WDEDF scheduler can provide flow protection, delay differentiation,

¹LR-decoupled reservation refers to a reservation that has a delay bound that is not bounded by $\frac{L}{R}$ where L is the maximum packet size and R is the reserved rate.

guaranteed service for LR-coupled² and LR-decoupled reservation, jitter minimisation and throughput guarantees using simulations. Finally, Section 5.6 summarises the characteristics of the WEDDF scheduler.

5.1 Introduction

WRR scheduler can be considered as a pioneer quantum-based control scheduler that ensures differentiated fairness between queues. Packets, from different flows, are queued in separate queues and the scheduler polls each queue in a cyclic manner in proportion to a weight assigned to each queue. In each poll, the scheduler serves a packet from a non-empty queue encountered. Flows are protected from misbehaving flows as the latter only overflow their own queue. WRR scheduler performs well when all packets have the same size.

DRR scheduler is a modification of WRR scheduler for handling of variable packet sizes without knowing the mean packet size of each flow in advance. DRR scheduler provides near-perfect throughput fairness and isolation of flows at very low implementation cost of $O(1)$ packet processing time. In DRR scheduler, each queue is allocated a fixed quantum and is serviced in a round robin manner. When a queue is served, its allocated quantum is reduced according to the size of the packet sent. This deduction continues until sending of the next packet would result in a deficit of quantum. The final balance of the quantum is added to the quantum assigned in the next service round. This preservation method ensures fairness and allows packets with size greater than the assigned quantum to be sent.

Weighted Deficit Round Robin (WDRR) [57] scheduler is a direct application of DRR scheduler to provide weighted bandwidth allocation to different queues in the scheduler. Different queues are allocated a different quantum using a proportional weighted function.

²LR-coupled reservation refers to a reservation that has a delay bound that is bounded by $\frac{L}{R}$ where L is the maximum packet size and R is the reserved rate.

EDF scheduling algorithm is a deadline driven rule that assigns the highest priority to a task with immediate deadline. Priority assignment is dynamic as a task's priority changes as time passes. In addition, a priority can also be reassigned when a new task arrives. EDF scheduler can achieve full utilisation for an arbitrarily large task set [58]. It yields a feasible schedule for a set of n tasks whenever $\rho = \sum_{i=1}^n \rho_i \leq 1$, whereby ρ is the load factor (or density) of a periodic task i such that $\rho_i = \frac{C_i}{T_i}$ in which C is the processing time of the task and T is the period for the task.

Delay-jitter regulator preceding EDF scheduler for each jitter sensitive flow has been proposed in Jitter-EDD which extends EDF. The delay-jitter regulator holds a packet till its computed deadline period before it is released to scheduler. This removes the effect of queuing delay variability at the previous scheduler which in turn avoids the burstiness built up within the network but it causes this scheduler to be a non-work-conserving scheduler.

5.2 Weighted Deficit Earliest Departure First Scheduler

WRR, DRR and WDRR schedulers provide strong rate differentiation between flows but do not guarantee any delay differentiation. This is due to their insensitivity to packet's deadline and their packet forwarding sequence. WRR scheduler services a packet from each queue in a round robin manner and DRR scheduler services a series of packets from each queue till the quantum allocated to the queue is exhausted. On the other hand, EDF and Jitter-EDD schedulers are unable to provide flow protection and bandwidth guarantees. Although, Jitter-EDD scheduler provides a jitter guarantee to its flows, it is non-work-conserving.

WDEDF scheduler addresses these deficiencies. It has a quantum-based control scheduling algorithm that achieves delay differentiation while maintaining fairness between its queues in terms of rate differentiation and flow protection. In addition, it can provide delay guarantee for LR-coupled and LR-decoupled reservations; and minimise jitter for jitter

sensitive flows.

WDEDf scheduler consists of two main components. It selects a packet at the HOL of each queue of the same service class with the earliest deadline for its forwarding engine that uses a forwarding rule to determine the next packet to be forwarded, based on the packet's service class and deadline. WDEDf scheduler uses a schedulability check algorithm for admission of new flows and adjustment of quantum setting.

5.2.1 Forwarding Head-Of-Line Packet with the Earliest Deadline

In WDEDf scheduler, each queue is allocated a weighted quantum using a proportional weighted function similar to DRR scheduler. However, unlike DRR scheduler, WDEDf scheduler does not restrict the setting of minimum quantum size to be the MTU of a router. Each queue i reserves a bandwidth of rate R_i on the scheduler. Based on the reservation rates, each queue is allocated ω_i worth of bits in each service round. Let us define ω^{MIN} as the minimum quantum size of WDEDf scheduler and $R^{MIN} = \min_{j=1}^F(R_j)$ where F is the total number of flows. The quantum allocated to queue i is simply

$$\omega_i = \frac{R_i}{R^{MIN}} \times \omega^{MIN}, \quad (5.1)$$

with $\omega^{MIN} \geq MTU$. This quantum allocation method ensures that at least one packet from each queue can be transmitted per service round as $\omega^{MIN} \geq MTU$.

A deficit counter (DC) that increases by ω_i at the start of each service round is kept for each queue. WDEDf scheduler decreases the DC_i by the size of the packet forwarded for queue i . A queue is *blocked* for service in the current service round when it exceeds its DC by forwarding its next packet, i.e., WDEDf scheduler blocks a queue when its DC value is insufficient to allow the forwarding of its next packet.

WDEDf scheduler works in service rounds. A service round ends when all queues are either emptied or blocked. Empty queues are set as *inactive* and their DC set to

zero. For *blocked* queues, the final balance of their DC are preserved and summed with the quantum value to be added in the next round. All *blocked* queues are set as *active* queues at the start of a new service round. An *inactive* queue becomes *active* again with DC set to its allocated ω when it has a new packet.

DRR scheduler services packets in a queue until the DC of the queue is insufficient for forwarding the HOL packet. This criterion can implicitly minimise delay of packet from the queue in service at the expense of increasing the delay of other queues waiting to be served. An example of this phenomenon is illustrated in Example 1.

WDEDf scheduler improves on DRR scheduler to provide a lower queuing delay by forwarding the packet at the head of the queue with the earliest departure time. The departure time (T_D) (see Fig. 5.1) is the time a packet needs to leave the scheduler.

Figure 5.1: General Queue Timing Diagram.

For each queue in WDEDf scheduler, a simple departure time computation using Eqn. 5.2, where T_A is the arrival time of packet, L is the length of packet, R is the reservation rate of the packet and C^l is the service rate of the server, can be used to compute the expected departure time, $\widehat{T}_D$, of each arrived packet.

$$\widehat{T}_D = T_A + \frac{L}{R} - \frac{L}{C^l} \quad (5.2)$$

Example 1 There are two non-empty queues (Queue A and B) set up on a router with a 10 Mbps link and A_i represents the i packet of Queue A. Likewise for Queue B. Queue A

Table 5.1: Packet Departure Times used in Example 1

Packet	Packet Length (bits)	Queuing Time (μs)	Service Time (μs)	Departure Time (μs)
A_1	400	200	40	$t_A + 160$
B_1	500	166	50	$t_A + 216$ ($t_A + 100 + 116$)
A_2	400	200	40	$t_A + 320$ ($t_A + 160 + 160$)

Table 5.2: Packet Completion Times used in Example 1

Packet	Expected Completion Time (μs)	DRR Completion Time (μs)	WDEDF Completion Time (μs)
A_1	$t + 38$ ($t_A + 200$)	$t + 40$	$t + 40$
B_1	$t + 104$ ($t_A + 100 + 166$)	$t + 130$ ($t + 80 + 50$)	$t + 90$ ($t + 40 + 50$)
A_2	$t + 198$ ($t_A + 160 + 200$)	$t + 80$ ($t + 40 + 40$)	$t + 130$ ($t + 90 + 40$)

Expected Completion Time = $T_A + \frac{L}{C}$ and $t_A = t - 162$

reserved 2 Mbps and Queue B reserved 3 Mbps on the router. A_1 arrives at the router at time $(t_A) \mu s$. B_1 arrives at time $(t_A + 100) \mu s$. A_2 arrives at time $(t_A + 160) \mu s$. The departure time of each packet is tabulated in Table 5.1 and the quantum is set to 1000 bits. At time $t = (t_A + 162) \mu s$, the scheduler starts to service Queue A followed by Queue B. The router starts to service Queue A using DRR scheduler and it forwards the packets in the following sequence: A_1 , A_2 and B_1 . A_2 is serviced before B_1 as Queue A has not used up its quantum for the round. This provides A_2 a free expedited service at the expense of B_1 , which misses its deadline, as shown in Table 5.2. WDEDF scheduler forwards these packets using the packet's earliest departure time. This results in the following forwarding sequence: A_1 , B_1 and A_2 . Table 5.2 shows the tabulated results. Both B_1 and A_2 did not miss their deadline.

WDEDF scheduler also does not block queues infinitely till the next round when the last

packet of the queue is sent. In DRR scheduler, a queue is removed from its active list once the last packet is sent and its DC is reset. This might delay packets that arrived at the system late but still conforms to its service contract. In WDEDF scheduler, DC of queues are only reset when all queues are either empty or had exhausted its DC for the service round. An example of this phenomenon is illustrated in Example 2.

Example 2 Consider a situation where there are three queues (Queue A, B and C) set up on a router with a single 20 Mbps output link. Queue A reserved 3 Mbps with average packet length of 600 bits. Queue B reserved 7 Mbps with average packet length of 1500 bits and Queue C reserved 10 Mbps with average packet length of 800 bits. The system assigns 1500, 3450 and 4950 worth of quantum to queues A, B and C, respectively. Assume that there are backlogs for Queue B and C and there is a packet, A_1 , in Queue A at a time instance t when the system begins to service Queue A. The next packet of Queue A, A_2 , arrives at the system at $(t + 200)$ ms conforming to the reservation at 3 Mbps.

Using a DRR scheduler, A_1 is serviced at time t and it leaves the system at $(t + 30)$ ms. After that, Queue A will be blocked till the next round. The next round starts after $(t + 200 \text{ ms} + \frac{(1500 \times 2 + 800 \times 4)}{20 \text{ Mbps}}) = (t + 510) \text{ ms}$. The blocking of Queue A delays the delivery of A_2 and the balance of value 1100 in DC of Queue A is lost.

The WDEDF scheduling algorithm is depicted in Algorithm 2 and 3. Consider any output link for a given router, queues are numbered 0 to $(n - 1)$ where n is the maximum number of output queues. Let Queue i be the i^{th} queue that stores packets for Flow i , and ω_i is the allocated quantum for Queue i . The deficit counter for Queue i , DC_i , contains the instantaneous bytes that are available for Queue i in a service round. *Active List* stores the index of queues, sorted based on the $\widehat{T}_D$ of packet at the HOL. Similarly, *Blocked List* stores the index of blocked queues to be used in the next round. A *Waiting List* is kept for queues that are emptied but have not exhausted their DC. The algorithm uses the following modules : The *En-queue Module* and *De-queue Module* are standard queuing

operators; *Get_Packet Module* returns the queue that contains the HOL packet with the earliest expected departure time; The *Update_Queue_Status Module* inserts a queue into its appropriate list; and the *New_Round Module* prepares queues for a new service round; The *Reduce_DC Module* subtracts the *DC* by the specified packet size; and *Free_Buffer Module* frees the buffer from the flow with the longest queue using McKenney's buffer stealing [59]. The WDEDF scheduler is triggered by packet arrival and packet departure events.

Packet Arrival

When a packet arrives, the *En-queue Module* is invoked and the packet is en-queued into its designated queue, Queue i . Queue i joins the *Active List* and starts competing for service with its allocated quantum if it is currently in the *Inactive List*; or with its leftover quantum (which must be larger than or equal to the size of the arrived packet) if it is currently in the *Waiting List*. However, if the size of the arrived packet is greater than the leftover quantum when Queue i is in the *Waiting List*, Queue i will join the *Blocked List* as it will have insufficient quantum to service this packet during the current service round.

Packet Departure

When the scheduler is ready to service the next packet, the *De-queue Module* is invoked, which returns the HOL packet with the earliest expected departure time of its queues and updates the *DC* of the queue accordingly before returning the packet to the scheduler for transmission. The queue, which the packet is selected from, rejoins the *Active List* if it has sufficient quantum to clear its next packet; joins the *Waiting List* if it has no more packets; or joins the *Blocked List* if it does not have sufficient quantum to clear its next packet. The various states of a queue can be directly mapped into its assigned list kept by the scheduler.

Algorithm 2 WDEDF Scheduler, Part I

Initialise() module:

```

for all  $i$  such that  $0 < i < n$  do
     $DC_i = 0$ 
    // Normalise quantum ( $\omega$ ) for Queue  $i$  using Eqn. 5.1
     $\omega_i = \text{normalise}()$ 
end for

```

En-queue(Packet k) module:

```

 $i = \text{Extract\_Queue\_Identity}(k)$ 
// Compute the Expected Departure Time for Arrived Packet  $k$  of Queue  $i$  ( $\widehat{T}_{D_i,k}$ )
// using Eqn. 5.2
 $\widehat{T}_{D_i,k} = T_A + \frac{L_k}{R_i} - \frac{L_k}{C_i}$ 
if  $\text{Out\_of\_Buffer}() = \text{TRUE}$  then
     $\text{Free\_Buffer}()$ 
end if
if  $\text{Queue\_Empty}(i) = \text{TRUE}$  then
    if  $\text{Queue\_in\_Inactive\_List}(i) = \text{TRUE}$  then
         $DC_i = \omega_i$ 
    else
        // Queue  $i$  in Waiting List
    end if
     $\text{enqueue}(i, k)$ 
     $\text{Update\_Queue\_Status}(i)$ 
else
     $\text{enqueue}(i, k)$ 
end if

```

De-queue() module:

```

 $i = \text{Get\_Packet}()$ 
 $k = \text{dequeue}(i)$ 
 $\text{Reduce\_DC}(i, L_k)$ 
 $\text{Update\_Queue\_Status}(i)$ 
return  $k$ 

```

Update_Queue_Status(Queue i) module:

```

if  $\text{Queue\_Empty}(i) = \text{FALSE}$  then
     $k = \text{Front\_Packet\_of\_Queue}(i)$ 
    if  $L_k \leq DC_i$  then
         $\text{Add\_Active\_List}(i)$ 
    else
         $\text{Add\_Blocked\_List}(i)$ 
    end if
else
     $\text{Add\_Waiting\_List}(i)$ 
end if

```

Algorithm 3 WDEDF Scheduler, Part II**Get_Packet()** module:

```

while Found( $i$ ) do
  if List_Empty(Active_List)=FALSE then
     $i = \text{Get\_Front\_Queue}(\text{Active\_List})$ 
  else
    New_Round()
  end if
end while
return  $i$ 

```

New_Round() module:

```

for all Queue  $j$  in Waiting_List do
   $DC_j = 0$ 
end for
Add Queue in Waiting_List to Inactive_List
for all Queue  $j$  in Blocked_List do
   $DC_j += \omega_j$ 
end for
Active_List = Blocked_List
Blocked_List = Waiting_List = Empty List

```

Reduce_DC(Queue i , Packet_Size L_k) module:

```

 $DC_i = DC_i - L_k$ 

```

Fig. 5.2 shows the state diagram of a WDEDF queue. It details the states of a queue and events that trigger state changes. Specifically, a queue can be in one of the following states: *Inactive*, *Active*, *Waiting* and *Blocked*. When there is no backlog in a queue at the start of a service round, the queue is in its *inactive state*. On the arrival of the first packet, the queue enters its *active state*. A queue is in *waiting state* (added into the *Waiting List*) when it is emptied after the last packet has been serviced. When there are still packets in the queue waiting for service, the next packet will be examined. If servicing the next packet will create a deficit in its *DC*, the queue will enter the *blocked state* (added into the *Blocked List*), otherwise, it will rejoin the *active state* (added into the *Active List*).

A new round begins when the *Active List* is emptied, resulting in state change of a queue. A queue at *waiting state* will go into *inactive state* as there is no backlog; while a queue at *blocked state* will change its status to *active state* as it has new quantum to clear

its backlog upon a new service round.

Figure 5.2: State Diagram of a queue implementing the WEDDF Scheduling Algorithm.

From the WEDDF scheduling algorithm, the algorithmic complexity³ of WEDDF scheduler depends on the complexity of the sorting algorithm used to maintain the sorted lists. Therefore, the algorithmic complexity of WEDDF scheduler is $O(\log N)$ which is similar to that of EDF scheduling algorithm but WEDDF scheduling algorithm has a smaller size of N . The size of N for WEDDF's scheduling algorithm is bounded by the maximum number of queues while the size of N for EDF scheduling algorithm is bounded by the total number of packets in the scheduler. Thus, in the case of an EVTRS-enabled router that deploys WEDDF, the algorithm complexity will remain as $O(\log N)$ as the computation of timing information does not involve any loop for both ingress and core routers.

³Algorithmic complexity is generally written in a form known as *Big-O* notation, where the $O(N)$ represents the complexity of the algorithm and a value N represents the size of the set the algorithm is run against.

5.2.2 Quantum computation and QoS support

Quantum allocation for WDEDF scheduler is based on the proportional weighted function as shown in Eqn. 5.1. The selection of the ω^{MIN} parameter plays a crucial part in providing QoS support.

Zheng et al. [60] have shown that a deadline-ordered service can be used to establish real-time channels in a point-to-point packet switching network. The authors laid a mathematical basis for the problem of establishing real-time channels. The result checks if every packet will be delivered within a pre-specified delay bound for a given traffic characteristic that is specified as (T, C, d) whereby T is the minimum packet inter-arrival time, C ⁴ is the maximum packet transmission time over a link and d is the requested delay bound. Thus, $\frac{C}{T}$ defines the traffic load generated by the flow. This traffic characterisation is essential for guaranteeing bounded queuing delays. Using this traffic characterisation, Zheng et al. proved the schedulability of deadline-ordered service which is restated in Theorem 3 and Theorem 4.

Theorem 3 *A set of n queues $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \dots, n$, are schedulable over a link under a non-preemptive deadline scheduling policy if and only if*

$$\forall t \geq d_{min}, \sum_{j=1}^n \lceil \frac{t - d_j}{T_j} \rceil^+ C_j + C_p \leq t$$

where $d_{min} = \min(d_i : 1 \leq i \leq n)$; C_p ⁵ is the time needed to transmit a maximum size packet; and the function $\lceil x \rceil^+ = n$ if $n - 1 \leq x < n$, $n = 1, 2, \dots$, and $\lceil x \rceil^+ = 0$ for $x < 0$.

The practical realisation of Theorem 3 is restated in Theorem 4.

Theorem 4 *A set of real-time queue $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \dots, n$, are schedulable over*

⁴ $C = \frac{L^{MAX}}{C^l}$, where L^{MAX} is the maximum packet size of a flow and C^l is the transmission rate of the link.

⁵ $C_p = \frac{L^{MTU}}{C^l}$, where L^{MTU} is the MTU of the link and C^l is the transmission rate of the link.

a link under the non-preemptive deadline driven scheduling policy if and only if both of the following hold :

$$1. \sum_{j=1}^n \frac{C_j}{T_j} \leq 1$$

$$2. \forall t \in \mathbb{S}, \sum_{i=1}^n \lceil \frac{t-d_i}{T_i} \rceil^+ C_i + C_p \leq t,$$

$$\text{where } \mathbb{S} = \bigcup_{i=1}^n S_i, S_i = d_i + nT_i : n = 0, 1, \dots, \lfloor \frac{t_{max}-d_i}{T_i} \rfloor;$$

$$\text{and } t_{max} = \max(d_1, d_2, \dots, d_n, \frac{C_p + \sum_{i=1}^n (1 - \frac{d_i}{T_i}) C_i}{1 - \sum_{i=1}^n \frac{C_i}{T_i}}).$$

Using the approach proposed by Zheng et al., we establish a checking algorithm to determine if all real-time packets can be delivered within their pre-specified delay bound for a given set of flows and ω^{MIN} .

Each service round of WDEDF scheduler is controlled by the total amount of quantum allocated to it. In each service round, packets from each queue is serviced based on their $\widehat{T_D}$. Observing from the operation of WDEDF scheduler, we can show that the maximum frame size of a service round is

$$\Omega^{MAX} = \sum_{i=1}^n (\omega_i + L_i^{MAX})$$

(see Theorem 8 in Section 5.3). Thus, the maximum frame time of WDEDF scheduler is $T_\Omega = \frac{\Omega^{MAX}}{C}$. We extend Theorem 4 to ensure that the set of queue is schedulable in a time frame up to T_Ω with a condition that $\forall i \in \tau, T_\Omega \geq d_i$.

Theorem 5 A set of real-time queue $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \dots, n$. are schedulable over a link using WDEDF if and only if all of the followings hold :

$$1. \sum_{j=1}^n \frac{C_j}{T_j} \leq 1$$

$$2. \forall i \in \tau, T_\Omega \geq d_i$$

$$3. \forall t \in \mathbb{S}, \sum_{i=1}^n \lceil \frac{t-d_i}{T_i} \rceil^+ C_i + C_p \leq t,$$

$$\text{where } \mathbb{S} = \bigcup_{i=1}^n S_i, \quad S_i = d_i + nT_i : n = 0, 1, \dots, \lfloor \frac{t_{max} - d_i}{T_i} \rfloor;$$

$$\text{and } t_{max} = \max(\frac{C_p + \sum_{i=1}^n (1 - \frac{d_i}{T_i}) C_i}{1 - \sum_{i=1}^n \frac{C_i}{T_i}}, T_\Omega).$$

Assuring the schedulability of the set of queues, we can now compute the expected quantum, $\hat{\omega}$, required by each queue using Eqn. 5.3. Each pre-computed quantum, ω , must be greater than or equal to the $\hat{\omega}$ without exceeding the $\Omega^{MAX} = t_{max} \times L^{MTU}$.

Theorem 6 *The expected quantum required by a real-time queue i is*

$$\hat{\omega}_i = (1 + \frac{t_{max} - d_i}{T_i}) L_i^{MAX}. \quad (5.3)$$

Proof: Since the minimal packet inter-arrival time of a queue i is T_i , there is at most $\lceil \frac{t - d_i}{T_i} \rceil +$ packets arrived during $[0, t]$ with deadlines $\leq t$ [60]. Thus, for $t \geq \max\{d_i : i = 1, \dots, n\}$,

$$\lceil \frac{t - d_i}{T_i} \rceil + \leq 1 + \frac{t - d_i}{T_i}$$

Therefore, the number of packet arrivals from queue i in the period t_{max} is at most $1 + \frac{t_{max} - d_i}{T_i}$ which can generate a maximum of $(1 + \frac{t_{max} - d_i}{T_i}) L_i^{MAX}$ bytes. ■

5.2.3 QoS Forwarding Rule

To support multi-services that require different network requirements, we classify QoS reservation into four classes, namely, (1) Time-Critical (TC) class for flow that requires a specific delay bound. This class also supports flows which require a low queuing delay that is decoupled from its reservation rate; (2) Jitter-Sensitive (JS) class for flow that requires a queuing delay that is closer to its expected queuing delay as expediting or delaying a JS packet causes jitter; (3) Rate-Based (RB) class for flow that requires a specific amount of bandwidth as queuing delay is not critical; (4) Best-Effort (BE) class for normal Internet traffic.

Building on the WEDDF scheduler, we focus on a packet forwarding sequence for the quantum-based scheduling that supports TC, JS, RB and BE reservations. Our proposed forwarding sequence provides non-preemptive⁶ priority service to TC queues, followed by a “Completion time” priority service to JS queues over RB queues and BE queue.

Figure 5.3: Block Diagram of WEDDF for multi-services support.

To provide an analogy, a service round is divided into four parts: TC part, JS part, RB part and BE part. In each part, packets are sorted in ascending order using the $\widehat{T}_D$ of the HOL packets and packets in the same part are delivered in a FIFO manner. In each service round, all packets in the TC part contend with the JS packet at the head of

⁶Non-preemptive service refers a service that does not preempt a packet of any priority that is already in service when a packet of a higher priority arrives.

the JS part, having a $\widehat{T}_D$ that is eligible or has missed its deadline, for a service. These packets are forwarded before packets from other parts; and all RB packets are forwarded before BE packets. A RB or BE packet is forwarded only when the TC and JS parts are cleared or the $\widehat{T}_C$ of the RB or BE packet is less than the $\widehat{T}_D$ of the TC and JS packets.

Algorithm 4 Overriding module for multi-service support

Get_Packet() module:

```

while Found(i) do
  if List_Empty(Active_List)=FALSE then
    if List_Empty(Active_TC_List)=TRUE then
      if List_Empty(Active_RB_List)=FALSE then
         $i = \text{Get\_Front\_Queue}(\text{Active\_RB\_List})$ 
      else
         $i = \text{Get\_Front\_Queue}(\text{BE})$ 
      end if
    if List_Empty(Active_JS_List)=FALSE then
       $j = \text{Get\_Front\_Queue}(\text{Active\_JS\_List})$ 
      if  $\widehat{T}_C(i) > \widehat{T}_D(j)$  then
         $i = j$ 
      end if
    end if
  else
     $i = \text{Get\_Front\_Queue}(\text{Active\_TC\_List})$ 
    if List_Empty(Active_JS_List)=FALSE then
       $j = \text{Get\_Front\_Queue}(\text{Active\_JS\_List})$ 
      if  $(\widehat{T}_D(j) \leq \widehat{T}_D(i))$  and  $(\widehat{T}_D(j) \leq \text{current time})$  then
         $i = j$ 
      end if
    end if
  end if
  if NOT(Found(i)) then
    New_Round()
  end if
else
  New_Round()
end if
end while
return  $i$ 

```

The block diagram of WDEDF scheduler is depicted in Fig. 5.3. WDEDF scheduler classifies queues into TC, JS and RB classes. An *Active List* of HOL packets is created for

each class. This *Active List* is ordered by the $\widehat{T}_D$ of packets. The HOL packet from each queue of the same class will be queued in the *Active List* if the length of the packet does not exceed the instantaneous DC value of its queue. The packet selector selects a packet for service based on the forwarding rule. The algorithm of the forwarding rule is presented in Algorithm 4. In the algorithm, starvation of BE traffic is not taken into account as WEDF works in service round whereby in each service round, a fixed amount of quantum is allocated to each queue. Thus, no queue will encounter any starvation.

Thus, WEDF scheduler is a work-conserving scheduler that provides differential scheduling for a predefined set of classes targeted for sessions that require low bandwidth and intolerable delay bound, delay bound, delay-jitter bound and bandwidth bound.

5.3 Analytical Results

We study the WEDF scheduling algorithm in this section and prove its fairness and its delay bound is statistically lower than DRR scheduler when WEDF scheduler acts as a scheduler without supporting any delay bound guarantees and differentiation of packet by their service class.

5.3.1 Fairness of WEDF Scheduler

Shreedhar et al. [35] have proven that DRR scheduler provides a throughput fairness measure (FM). In the following theorems, we show that WEDF scheduler maintains the service rate allocated to each queue even though it rearranges the forwarding sequence of packets; the FM theorem is used to show fairness in DRR scheduler also holds for WEDF scheduler; and WEDF scheduler gives a bound on the service offered to each queue over an arbitrary number of service rounds.

For WEDF scheduler, the sum of quantum available for any time interval is the same as DRR scheduler if both scheduling disciplines used the same ω^{MIN} .

Lemma 1 *In each service round, at least one packet from each active queue, q , is serviced as, for any queue q , its allocated quantum, ω_q , always satisfies the condition: $\omega_q \geq L_q^{MAX}$.*

Proof: In WDEDF scheduling algorithm, each queue is assigned a quantum value, ω , that represents its service reservation on a scheduler. Let us define L^{MTU} as the MTU for a router such that $\omega^{MIN} \geq L^{MTU} \geq \max_{i=1}^F(L_{q,i}^{MAX})$ where L^{MAX} is the maximum packet size of each flow in queue q , and F is the total number of flows. Quantum is normalised so that the smallest quantum is at least greater than or equal to ω^{MIN} to ensure that at least one packet in each active queue is serviced in each service round. Consider each queue q has made a reservation of R_q bps on the scheduler. Let R^{MIN} be the minimum value of all reserved rates from the set of queues supported by the scheduler. The quantum value, ω_q , allocated to queue q is simply $\frac{R_q}{R^{MIN}} \times \omega^{MIN}$. Therefore, for any q , the quantum, ω_q , allocated to queue q is always greater than or equals to L_q^{MAX} . ■

Similar to the approach used in the analysis of DRR scheduling algorithm, we begin with an invariant that is true for all executions of the WDEDF scheduling algorithm. An invariant is a constant that is meant to be true after every program action.

Lemma 2 *For each queue q , the following invariant holds for every execution of the WDEDF scheduling algorithm: $0 \leq DC_q < L_q^{MAX}$ where DC_q and L_q^{MAX} are the deficit counter and maximum packet size of queue q , respectively.*

Proof: Initially, DC_q begins with zero (inactive state) and approaches to $\omega_q - L_{q,k}$ after servicing the first packet k with the size of $L_{q,k}$ of queue q . The value of DC_q only changes after queue q is serviced. After the end of each service round, if a packet is left in queue q , it must be of size strictly greater than DC_q and no more than L_q^{MAX} . Therefore, DC_q is strictly less than L_q^{MAX} and the algorithm guarantees that $DC_q \geq 0$. If there is no packet in queue q , the algorithm resets the value of DC_q to zero on the start of the next service round. ■

Lemma 3 The service offered (bytes sent), S , for queue q in service round η is

$$S_q^\eta = \omega_q + DC_q^{\eta-1} - DC_q^\eta \quad (5.4)$$

where ω_q is the quantum value allocated to queue q and DC_q^η is the deficit counter of queue q at the end of service round η .

Proof: Following the WEDDF scheduling algorithm, for a service round η , a queue q is allocated $\omega_q + DC_q^{\eta-1}$ bytes. At the end of service round η , if queue q sent S_q^S bytes then the remaining bytes will be stored in DC_q^η . Thus, $S_q^\eta + DC_q^\eta = \omega_q + DC_q^{\eta-1}$. ■

Theorem 7 Over any time interval of service rounds, the service rate of a queue is maintained despite the fact that packet service order used by WEDDF scheduler interleaved packets from other queues in each service round.

Proof: Consider queue q being allocated ω_q of the total allocated quantum size, $\Omega = \sum_{i=1}^Q \omega_i$, where Q is the total number of queues in a scheduler. Let $t_{\eta-1}$ and t_η indicate the time that service round η starts and finishes respectively, in each service round interval $(t_{\eta-1}, t_\eta]$.

Using Lemma 3, the service offered, S_q , to queue q is bounded by Eqn. 5.5 and the total service offered to all queues, S , is bounded by Eqn. 5.6, where DC_q is the deficit counter of queue q .

$$S_q(t_{\eta-1}, t_\eta) = \omega_q + DC_q^{\eta-1} - DC_q^\eta \quad (5.5)$$

$$\begin{aligned} S_Q(t_{\eta-1}, t_\eta) &= \sum_{i=1}^Q S_i(t_{\eta-1}, t_\eta) \\ &= \Omega + \sum_{i=1}^Q (DC_i^{\eta-1} - DC_i^\eta) \end{aligned} \quad (5.6)$$

$$S_q(t_0, t_\nu) = \nu\omega_q + DC_q^0 - DC_q^\nu \quad (5.7)$$

Thus, in any service round, the service offered to queue q is constrained by $\frac{S_q(t_{\eta-1}, t_\eta)}{S_Q(t_{\eta-1}, t_\eta)}$ despite

the service order of packets from queue q being interleaved with packets from other queues.

This is also true for service offered to queue q over ν service rounds. The total service offered to queue q over ν service rounds is a telescoping series in Eqn. 5.7 where t_0 marks the time that begins a service round and t_ν is the time at end of the ν th service round after t_0 .

$$\begin{aligned}
 S_q(t_0, t_\nu) &= \sum_{\eta=1}^{\nu} \eta_j(t_{\eta-1}, t_\eta) \\
 &= \nu\omega_q + \sum_{\eta=1}^{\nu} (DC_q^{\eta-1} - DC_q^\eta) \\
 &= \nu\omega_q + DC_q^0 + \sum_{\eta=2}^{\nu} DC_q^{\eta-1} - \sum_{\eta=1}^{\nu-1} DC_q^\eta - DC_q^\nu \\
 &= \nu\omega_q + DC_q^0 - DC_q^\nu
 \end{aligned}$$

The service rate offered to any queue q over ν service rounds is $\frac{S_q(t_0, t_\nu)}{(t_\nu - t_0)}$ bps. Therefore, over any time interval of service rounds, the service rate of a queue can be maintained despite the packet service order used by WDEDF scheduler when interleaved with packets from other queues in each service round. ■

Theorem 8 The service offered, S_q , to queue q is bounded by

$$\nu \times \omega_q - L_q^{MAX} \leq S_q \leq \nu \times \omega_q + L_q^{MAX} \quad (5.8)$$

where ν be the number of service rounds received by queue q during the interval (t_0, t_ν) .

Proof: As shown in Eqn. 5.7, $S_q(t_0, t_\nu) = \nu\omega_q + DC_q^0 - DC_q^\nu$. This theorem follows using Lemma 2 which states that DC_q is always non-negative and greater than L_q^{MAX} . ■

Theorem 9 For any service round interval $(t_{\eta-1}, t_\eta)$ during an execution of the WDEDF

scheduling algorithm the fairness measure is

$$FM(t_{\eta-1}, t_{\eta}) \leq 2L^{MAX} \omega^{MIN}$$

where ω^{MIN} is the minimum quantum of all queues and

$$FM(t_{\eta-1}, t_{\eta}) = \max_{(i,j) \in \mathbb{B}} \left(\frac{S_i(t_{\eta-1}, t_{\eta})}{f_i} - \frac{S_j(t_{\eta-1}, t_{\eta})}{f_j} \right)$$

where $\mathbb{B}$ is the set of sessions with backlog packets in the interval $t_{\eta-1}, t_{\eta}$ and the quantity f expresses the ideal share to be used by queue q . The proof is analogous to Theorem 3 in DRR scheduler [35] based on Eqn. 5.7.

5.3.2 Delay bound comparison with DRR scheduler

The following analysis shows that the latency bound of each packet in WDEF scheduler is statistically less than the latency bound of DRR scheduler when WDEF scheduler acts as a scheduler without supporting any delay bound guarantees and differentiation of packets by their service class. In addition, we also derive a delay bound for WDEF scheduler when it is used as a LR scheduler.

Lemma 4 Let ω_q be the total quantum available for queue q during a service round, C^l be the link capacity and Q be the total number of queues. Assume that queue q is the last queue to receive its service in the round robin service sequence. Shreedhar et al. [35] show that the latency bound provided by DRR scheduler is

$$D_q^{DRR} = \sum_{\forall i: i \neq q, i=1}^Q \frac{\omega_i}{C^l}. \quad (5.9)$$

In other words, a small packet can be delayed by the sum of quantum from all other flows in the worst case.

Lemma 5 For a packet k at the head of its queue, q during a service round interval, the latency of packet k is bounded by the total number of packets that are eligible for transmission at the current service round with their $\widehat{T}_D$ earlier than the $\widehat{T}_D$ of packet k . Let C^l be the link speed of the router; K be the total number of packets in the set of packet $\mathbb{K}$ that is eligible for transmission⁷ at the current round; ε_k be a conditional variable that is set when the $\widehat{T}_D$ of a packet in $\mathbb{K}$ is smaller than the $\widehat{T}_D$ of k ; and L_k be the length of packet k . Then, the queuing delay of packet, k , is

$$D_{q,k}^{WDEF} = \sum_{i=1}^K \frac{\varepsilon_i L_i}{C^l}, \text{ where } \varepsilon_i \in \{0, 1\}. \quad (5.10)$$

Proof: WDEF scheduler forwards the packet at the head of a queue having the smallest $\widehat{T}_D$. The DC value of the queue will be decreased by the size of the packet. The DC value of a queue, q , is sufficient to allow more than one packet to be transmitted when $DC_q \geq L_k + L_{k+1}$. ■

WDEF scheduler repeats this forwarding rule until all deficit counters are exhausted. Intuitively, all the eligible packets in a service round can be regarded as a single prioritised queue based on the $\widehat{T}_D$ of each packet in ascending order. Therefore, the delay of packet k will be the sum of service time required to transmit all the packets in front of packet k in the single prioritised queue.

Theorem 10 For any packet k of queue, q , the latency bound of WDEF scheduler is statistically less than the latency bound of DRR scheduler.

Proof: From Lemma 4, the DRR scheduler's latency bound of queue q is

$$D_q^{DRR} = \sum_{\forall i: i \neq q, i=1}^Q \frac{\omega_i}{C^l}.$$

⁷A packet is eligible for transmission in WDEF scheduler if and only if the DC of the queue that the packet resides is greater than or equal to the size of the packet.

To open this summation, let γ_q be the number of packets eligible for transmission in queue q and $L_{i,j}$ be the packet length of packet i of queue j . Then, $\sum_j^Q \sum_i^{\gamma_j} \frac{L_{i,j}}{C^l}$. To map DRR scheduler's forwarding sequence into a single queue, each packet that is eligible for transmission is numbered using a variable j in ascending order. Thus, by generalising the summations in terms of j , the latency bound of a packet i in DRR scheduler is $\sum_{i=1}^K \frac{L_i}{C^l}$, where $K = \sum_j^Q \gamma_j$. Therefore, the inequality in Eqn. 5.11 shows that for any packet i of a queue, the latency bound of WEDF scheduler is less than or equal to the latency bound of DRR scheduler.

$$\sum_{i=1}^K \frac{\varepsilon_i L_i}{C^l} \leq \sum_{i=1}^K \frac{L_i}{C^l} \quad (5.11)$$

In WEDF scheduler, a small packet can be delayed by a sum of quantum by every other packet with a shorter departure time. ■

Although WEDF scheduler forwards the HOL packet with the earliest $\widehat{T}_D$, in an extreme case, the worst-case delay bound of a WEDF scheduler is similar to that of a WDRR scheduler.

Assume a well-behaved queue q contends for bandwidth with $(Q - 1)$ number of queues whereby the $\widehat{T}_D$ of all packets from the $(Q - 1)$ number of queues always have $\widehat{T}_D$ that are earlier than that of queue q . Thus, packets from Flow q are always the last packets to be serviced by the scheduler in a service round as compared to packets from other queues.

This worst-case scenario is analogous to WDRR scheduler. To our knowledge, only the worst-case delay bound of DRR scheduler has been derived. We derived the worst-case delay of WEDF scheduler when it is used as a LR scheduler in Appendix A. The result is restated in Corollary 1.

Corollary 1 *The worst-case delay bound of queue q in the WEDF scheduler when used*

as a LR scheduler is

$$\frac{\Omega^{MAX} - \omega_q - L_q^{MAX}}{C^l} + \frac{L_q^{MAX}}{R_q} \quad (5.12)$$

where $\Omega^{MAX} = \Omega + \sum_i^Q L_i^{MAX}$ in which Q is the total number of queues.

5.4 Simulation Results

This section presents simulation results of WDEDF scheduler in comparison with WFQ, WDRR and EDF schedulers. The following simulations show the behaviour of WDEDF scheduler in terms of

- flow protection and bandwidth sharing;
- delay bound when used as a LR scheduler;
- delay guarantee for LR-coupled and LR-decoupled reservations and;
- support for a set of service classes - TC, JS, RB and BE (see Section 5.2.3).

Fair bandwidth sharing and weighted bandwidth sharing are evaluated with scenarios when one flow misbehaves. Average and worst-case delay of packets from well-behaved flows are simulated using Constant Bit Rate (CBR), Variable Bit Rate (VBR) and an aggregation of both CBR and VBR traffic sources. To illustrate WDEDF scheduler's support for QoS, we used a series of simulation containing TC, JS, RB, and BE flows. In addition, we also show that the support of delivering TC flows with low rate and tight delay requirements. All simulations were performed using ns2. The implementation of DRR and WFQ schedulers for ns2 are taken from [49] and [61], respectively. In all simulations, infinite buffer is used to filter out any effects due to the lack of buffers. Unless stated, the $\widehat{T}_D$ of packets from an arbitrary flow are computed using in Eqn. 5.2 for EDF and WDEDF schedulers. A

single hop configuration, as shown in Fig. 5.4, is used to evaluate the behaviour of WDEDF scheduler.

Figure 5.4: A single hop configuration used to evaluate WDEDF scheduler.

5.4.1 Fair and Weighted Bandwidth Sharing and Flow Protection

Figure 5.5: Throughput of each VBR flow when router implements WFQ, DRR, EDF and WDEDF schedulers. All flows generate traffic at the same rate except for the malicious Flow 9.

This simulation shows the fair sharing property of WDEDF scheduler with respect to WFQ, DRR and EDF schedulers. It contains 20 flows (numbered 1 to 20) sending packets at 200 kbps using Poisson packet arrivals and random packet sizes (uniformly distributed

between 64 to 500 *bytes*). The malicious flow (Flow 9) generates traffic at a data rate of 300 *kbps*. The link is set at 4 *Mbps*, therefore each flow gets a fair share of 200 *kbps* each.

Fig. 5.5 shows the bandwidth offered to different flows using different scheduling disciplines. EDF scheduler allows Flow 9 to grab an arbitrary share of the bandwidth at the expense of other flows. On the other hand, DRR, WFQ and WDEDF schedulers ensure near perfection in fairness for bandwidth sharing even when there is a misbehaving flow, thus, providing flow protection for well-behaved flows.

Figure 5.6: Throughput of each VBR flow when router implements WFQ, WDRR, EDF and WDEDF schedulers. All flows generate traffic at their reserved rate (weighted) except for the malicious Flow 9.

We use another simulation to show the weighted bandwidth sharing property of WDEDF scheduler. The simulation settings remain the same as above except that each flow reserves a different rate. The reservation rate of a flow is equal to the flow number multiplied by 1 *kbps*. The sum of the reservation rate is 21 *kbps*.

Once again, the results in Fig. 5.6 show that EDF scheduler allows the malicious flow to grab an arbitrary share of the bandwidth at the expense of other flows while WDRR,

WFQ and WDEDF schedulers ensure near perfection in fairness for weighted bandwidth sharing even when there is a misbehaving flow.

5.4.2 Performances of WDEDF scheduler with CBR and VBR Traffic.

We observe the average delay and worst-case delay of packets from a set of flows using different schedulers in three different simulations where CBR traffic, VBR traffic and an aggregation of both CBR and VBR traffic are injected, respectively.

In the first simulation, Flow 1 to Flow 20 sent packets at the rate of 1 *Mbps* to 20 *Mbps*, respectively. These flows use CBR arrivals with constant packet size of 1000 *bytes*. The link is set to 252 *Mbps* for 80% utilisation. Similarly, in the second simulation, the 20 VBR traffic sources use Poisson packet arrivals instead of CBR arrivals. Each source generates packets with random packet sizes that are uniformly distributed between 64 to 500 *bytes*. The third simulation contains twenty flows of which ten flows inject CBR traffic and the other ten flows inject VBR traffic. Flow 1 to Flow 10 generate CBR traffic with constant packet size of 500 *bytes* at 1 *Mbps* to 10 *Mbps*, respectively; while Flow 11 to Flow 20 generate VBR traffic using Poisson packet arrivals with random packet sizes uniformly distributed between 64 to 500 *bytes* at 1 *Mbps* to 10 *Mbps*, respectively.

Packet Delay

We observe the average delay of packets from each flows scheduled using different schedulers in the three simulations. Fig. 5.7, Fig. 5.8 and Fig. 5.9 show that WDRR scheduler is unable to provide delay differentiation between flows with different bandwidth reservation. WDEDF scheduler, like WFQ and EDF schedulers, is able to provide a delay differentiation between flows with different bandwidth reservation in the presence of CBR traffic, VBR traffic and an aggregation of both traffic types. Most importantly, it has comparable performance to EDF scheduler in its ability to provide delay differentiation.

The worst-case delay of the three simulations are plotted in Fig. 5.10, Fig. 5.11 and Fig. 5.12, respectively.

Figure 5.7: Average delay of each CBR flow when router implements WFQ, EDF, WDRR and WDEDf schedulers.

Figure 5.8: Average delay of each VBR flow when router implements WFQ, EDF, WDRR and WDEDf schedulers.

Figure 5.9: Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. Flow 1 to 10 are CBR flows, and Flow 11 to 20 are VBR flows.

Figure 5.10: Worst-case delay of each CBR flow when router implements WFQ, EDF, WDRR and WDEDF schedulers.

Figure 5.11: Worst-case delay of each VBR flow when router implements WFQ, EDF, WDRR and WDEDF schedulers.

Figure 5.12: Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. Flow 1 to 10 are CBR flows, and Flow 11 to 20 are VBR flows.

The simulated results show that apart from being unable to provide average delay differentiation, WDRR scheduler is unable to provide worst-case delay differentiation for flows with either CBR traffic or VBR traffic or both. WDEDf scheduler, on the other hand, provides flows with a worst-case packet delay that is somewhere in between WFQ scheduler and EDF scheduler.

Deadline Misses

This section illustrates the improvements on deadline misses for WDEDf scheduler over WDRR scheduler. A packet misses its deadline if it is serviced by the scheduler after its $\widehat{T}_D$. We computed the $\widehat{T}_D$ for each packet, based on Eqn. 5.2 for the third simulation.

Achieving low percentage of deadline misses is important in providing QoS. The percentage of packet delivered within its $\widehat{T}_D$ is tabulated in Table 5.3. It shows that WDEDf scheduler achieves over 99% of packet deliveries within their specified deadline.

Table 5.3: Percentage (%) of packets delivered within specified deadline

CBR Flow 1 to 10										
WDRR	38.767	37.677	38.443	38.207	38.253	38.239	37.824	38.082	38.267	38.025
WDEDf	99.989	99.889	99.712	99.675	99.591	99.561	99.529	99.58	99.545	99.515
VBR Flow 11 to 20										
WDRR	38.025	38.471	38.935	38.356	38.256	38.098	38.548	38.52	38.421	38.367
WDEDf	99.945	99.842	99.696	99.583	99.606	99.589	99.534	99.542	99.545	99.503

5.4.3 Delay bound of WDEDf scheduler as a Latency-Rate scheduler

To illustrate the delay bound of WDEDf scheduler (see Corollary 1) when used as LR scheduler, we show the simulated worst-case delay of flows using CBR traffic sources; and the simulated worst-case delay of flows using VBR traffic sources. Six sources (numbered *RB1* to *RB6*) are used whereby each flow reserves a different rate. The reservation rates increase linearly from 1 *kbps* by a factor of 1 *kbps*; and the link is set to 210 *kbps*.

Figure 5.13: Worst-case delay of each flow when router implements WDRR and WDEDf schedulers. Flow RB 1 to RB6 are CBR flows.

Figure 5.14: Worst-case delay of each flow when router implements WDRR and WDEDf schedulers. Flow RB 1 to RB6 are VBR flows.

For CBR sources, the packet size is fixed at 1500 *bytes*; and for VBR sources, packet size is generated with an uniform distribution between 64 to 1500 *bytes*. Flow RB6 (the malicious flow) injects traffic at 1.2 times its reserved rate to keep the schedulers busy; while the other flows are leaky bucket regulated at their reserved rate.

Fig. 5.13 and Fig. 5.14 show the worst-case delay of each flow using CBR traffic and VBR traffic, respectively, against the computed expected delay bound using Eqn. 5.12. The plots show that the simulated worst-case delay of each flow is within the computed worst-case delay bound. To further illustrate, we plotted the worst-case delay of Flow RB5 (see plots “WDRR - only RB5 is leaky bucket regulated” and “WDEDF - only RB5 is leaky bucket regulated”) when all flows (except Flow RB5) are malicious and not leaky bucket regulated. The worst-case delays of Flow RB5 are still within the computed delay bound. In addition, from all the plots presented, WDEDF scheduler has a smaller worst-case delay than WDRR scheduler, which demonstrates Theorem 10.

5.4.4 Performance of WDEDF scheduler for Regulated Traffic with QoS

Table 5.4: Traffic Patterns for flows used in simulation study.

Flow	Traffic type	Reserved Rate (R) in <i>kbps</i>	Leaky Bucket rate (r) in <i>kbps</i>	Leaky Bucket depth (σ) in <i>bytes</i>
TC1	CBR	2	2	900
TC2	CBR	3	3	800
TC3	CBR	4	4	700
RB4	CBR	5	-	-
RB5	CBR	6	-	-
RB6	CBR	7	-	-
BE	CBR	5	-	-

Using the schedulability criteria developed in Section 5.2.2, we evaluate the performance of WDEDF scheduler in providing QoS support for TC flows, JS flows and RB flows. In addition, we also show that WDEDF scheduler is able to provide a delay bound that is

decoupled from bandwidth reservation. Unless stated, all simulations in this section use the patterns tabulated in Table 5.4. The link rate is set exactly at the sum of the reservation rates from all flows. TC and BE flows generate CBR traffic at thrice their reserved rate; and RB flows generate CBR traffic at their reserved rate. The ω^{MIN} of WDEDF scheduler is set at $30L^{MTU}$; and the $\widehat{T}_D$ of each packet from RB and BE flows are stamped with a large value when EDF scheduler is used.

Delivering HOL packets with the earliest deadline

This simulation follows the default setting except that ω^{MIN} of WDEDF is set at $1000L^{MTU}$; and the $\widehat{T}_D$ of RB and BE flows for both EDF and WDEDF are stamped with a large value. To illustrate the effect of HOL forwarding described in Section 5.2.1, this simulation does not classify flows into their respective service classes.

Figure 5.15: Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF (without QoS forwarding rule) schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. WDEDF scheduler forwards any HOL packet with the earliest $\widehat{T}_D$.

Figure 5.16: Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF (without QoS forwarding rule) schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. WDEDF scheduler forwards any HOL packet with the earliest $\widehat{T}_D$.

Fig. 5.15 and Fig. 5.16 show the average delay and worst-case delay of packet of each flow using WFQ, WDRR, EDF and WDEDF schedulers. The delay is measured from the time a packet enters the scheduler (not the leaky bucket) to the time before it starts being serviced by the scheduler. As shown in the figures, except for WDRR scheduler, all other schedulers are able to keep the delay of TC packets within the specified delay. Owing to the $\widehat{T}_D$ of the packets, the delay of packet from RB flows and BE flow using the EDF and WDEDF schedulers are large as they are scheduled only when there are no packets from TC flows. The worst-case delay of packet from RB flows using WDEDF scheduler is lower than using EDF scheduler as WDEDF scheduler limits the number of packet deliveries for packets from the malicious BE flow. This is a result of the flow protection property of WDEDF scheduler.

Delivery of Time-Critical Flow with low rate and intolerable delay

EDF scheduler can provide a reservation with a specified delay that is decoupled from its bandwidth reservation. This is not possible for LR schedulers like WFQ and WDRR schedulers. This type of traffic that needs a LR-decoupled reservation is known to have low rate and intolerable delay. We show that WDEDF scheduler can achieve a LR-decoupled reservation by using the same simulation settings except that TC 3 makes a bandwidth reservation of 0.5 *kbps* but still requests for a delay of $\frac{700 \text{ bytes}}{4 \text{ kbps}} = 1.4 \text{ s}$; and the link capacity is set to 28.5 *kbps* instead of 32 *kbps*. Fig. 5.17 and Fig. 5.18 show the average delay and worst-case delay of packet for each queue in this simulation. WDEDF and EDF schedulers maintain a delay below the expected delay for all TC flows which includes TC 3 that makes a LR-decoupled reservation whereas WFQ scheduler fails to provide a delay bound for a LR-decoupled reservation without over-provisioning.

Figure 5.17: Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation except TC3. TC3 requested a LR-decoupled reservation of 0.5 *kbps* with a maximum packet delay of 1.4 s. ω^{MIN} of WDEDF is $30L^{MTU}$

Figure 5.18: Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation except TC3. TC3 requested a LR-decoupled reservation of 0.5 *kbps* with a maximum packet delay of 1.4 s. ω^{MIN} of WDEDF is $30L^{MTU}$

Delivery of Time-Critical Flows and Rate-Based Flows

In this section, we will show that delay differentiation of RB flows can be achieved with $\widehat{T}_D$ computed as in Eqn. 5.2 using our QoS forwarding rule.

This simulation is set to observe the QoS forwarding rule described in Section 5.2.3 for TC, RB and BE service classes. All simulation settings remain the same except that WDEDF scheduler deploys the QoS forwarding rule; ω^{MIN} set to $30L^{MTU}$; and the $\widehat{T}_D$ of packets from RB flows in WDEDF scheduler are computed using Eqn. 5.2.

Fig. 5.19 and Fig. 5.20 show the average delay and worst-case delay of packet from each flow respectively. WDEDF scheduler shows that it maintains the delay bound for TC flows while providing delay differentiation and flow protection (from the BE flow that generates at three times its reserved rate) for RB flows.

Figure 5.19: Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and $\omega^{MIN} = 30L^{MTU}$

Figure 5.20: Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and $\omega^{MIN} = 30L^{MTU}$

Figure 5.21: Throughput of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and all RB flows are well-behaved except the malicious flow - RB5.

Figure 5.22: Worst-case delay of each flow using WFQ, EDF, WDRR and WDEDF schedulers and $\omega^{MIN} = 30L^{MTU}$. All TC flows are leaky bucket regulated and requested a LR-coupled reservation; and all RB flows are well-behaved except the malicious flow-RB5.

It provides a lower delay for RB flows as compared to WDRR and WFQ schedulers owing to the priority given to RB service class in the QoS forwarding rule. WDEDF scheduler, like WDRR and WFQ schedulers, is able to provide tight bandwidth bound when the RB Flow 5 misbehaved, as shown in Fig. 5.21. This further illustrates the flow protection property of WDEDF scheduler. With this malicious RB flow, EDF scheduler still maintains tight delay bound for TC flows (see Fig. 5.22) because it stamps the $\widehat{T}_D$ of RB flow with a large number, thus, it treats TC flows with strict priority over RB and BE flows implicitly.

Fig. 5.22 also shows that delay differentiation of RB4 and RB6 are still maintained. RB4 has a worst-case delay at 150.5 s and RB6 has a worst-case delay at 87.2 s. RB5 floods its own queue which results in long delay.

Delivery of Jitter-Sensitive Flows

We had shown that WDEDF scheduler is able to provide delay bound for TC flows for both LR-coupled and LR-decoupled reservation while maintaining bandwidth bound and providing delay differentiation for RB flows.

In Jitter-EDD scheduler, a delay jitter regulator is used to explicitly delay packets from JS flows so that packets are allowed to enter the scheduler only at their $\widehat{T}_D$. This caused Jitter-EDD scheduler to be a non-work-conserving scheduler. WDEDF scheduler's QoS forwarding rule delays packets from JS flows by forwarding packets from TC, RB and BE flows whenever possible (i.e., not causing packets from JS flows to miss their $\widehat{T}_D$). This minimises the delay-jitter of JS flows as compared to Delay-EDD scheduler.

To create arrival variance for JS flows, a token bucket with a bucket size, σ , of $3L^{MAX}$ is used for each JS flow. This causes misbehaviour of JS flows. In addition, Flow TC3 remains its reservation of 0.5 kbps with a delay of 1.4 s. Flows JS1, JS2 and JS3 reserve a rate of 2 kbps, 3 kbps and 4 kbps with a L^{MAX} of 900 bytes, 800 bytes and 700 bytes, respectively. The TC flow and BE flow settings remain as in the previous simulations.

Figure 5.23: Average delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. To create packet arrival variances at the scheduler, JS flows are token bucket regulated with $\sigma = 3L^{MAX}$.

Figure 5.24: Worst-case delay of each flow when router implements WFQ, EDF, WDRR and WDEDF schedulers. All TC flows are leaky bucket regulated and requested a LR-coupled reservation. To create packet arrival variances at the scheduler, JS flows are token bucket regulated with $\sigma = 3L^{MAX}$.

WDEDf and EDF schedulers keep the previous arrival time, T_A^{k-1} , of packet for each flow to compute the $\widehat{T}_D^k$ of the currently arrived packet k . Thus, the $\widehat{T}_D^k$ of a JS packet is computed as

$$\widehat{T}_D^k = T_A^{k-1} + d. \quad (5.13)$$

Each flow generates exponential on/off traffic with a peak rate of 1.5 times the reserved rate, R , having an “on” time of $\frac{4}{0.5R}$ at a load of 0.35 for JS flows and 0.95 for all other flows.

Figure 5.25: Average Inter-packet Arrival Delay of each JS flow when router implements EDF and WDEDf schedulers, plotted with a variance at 99% confidence.

The average delay and worst-case delay of packet for TC and JS flows are shown in Fig. 5.23 and Fig. 5.24, respectively. Both EDF and WDEDf schedulers still managed to provide a delay bound for JS flows despite the misbehaviour of the flows. More importantly, WDEDf scheduler shows that it can provide less variances in inter-arrival packet delay for JS flows as shown in Fig. 5.25 which plots the inter-arrival packet delay of the JS flows with

a variance at 99% confidence.

Bandwidth guarantees on WDEDF

To illustrate the bandwidth guarantees of WDEDF, we tabulated, in Table 5.5, the bandwidth allocated to each flow during a 200 s interval. The figures show that WDEDF, like WFQ and WDRR, provides bandwidth protection to all flows, including the lowest priority BE flow. In WDEDF, low priority flows will not be starved as it works in service rounds whereby in each service round, a fixed amount of quantum is assigned to each flow. EDF is unable to provide flow protection as shown in the table. It allocated more bandwidth to the malicious flow (BE). The bandwidth allocated to the TC and JS flows are not affected as EDF gave these flows a higher priority.

Table 5.5: Bandwidth of each flows

Scheduler	TC1	TC2	TC3	JS1	JS2	JS3	RB1	RB2	RB3	BE
WFQ	2.0096	3.01298	0.513022	1.0472	1.56267	2.07138	5.82378	6.98844	8.15378	9.81444
WDRR	2.0088	3.01156	0.512711	1.0472	1.56053	2.07138	5.82333	6.99333	8.16133	9.80733
EDF	2.0096	3.01333	0.513333	1.0472	1.56373	2.07169	4.41267	5.27	6.16933	14.9267
WDEDF	2.0096	3.01333	0.513333	1.0472	1.56373	2.07169	5.84133	6.97867	8.18067	9.778

Effect of Different Quantum Allocation Methods

We used $\omega^{MIN} = 30L^{MTU}$ for all simulations in the above sections as it closely sums up the required reservation rate by all flows as shown in Fig. 5.26.

In this section, we investigate the use of smaller ω^{MIN} at the expense of reducing bandwidth allocation for BE flow. Using the default simulation setting, we ran simulations using WDEDF scheduler with different ω^{MIN} (1 MTU, 5 MTU, 10 MTU and 30 MTU). The bandwidth allocated for each flow is plotted in Fig. 5.26.

Figure 5.26: Bandwidth allocation for each flow when router implements WEDDF with different ω^{MIN} value.

Figure 5.27: Average delay of each flow when router implements WEDDF scheduler with different ω^{MIN} value. All TC flows are leaky bucket regulated and requested a LR-coupled reservation.

Figure 5.28: Worst-case delay of each flow when router implements WDEF scheduler with different ω^{MIN} value. All TC flows are leaky bucket regulated and requested a LR-coupled reservation.

As shown in the average delay and worst-case delay plots in Fig. 5.27 and Fig. 5.28, respectively, using a small ω^{MIN} results in lower delay bound for RB flows. A smaller ω^{MIN} results in a shorter service round (i.e., smaller frame size) for WDEF scheduler which can be verified from the mathematical analysis of DRR scheduler (see Appendix A) which shows that a shorter service round gives a smaller delay bound.

5.5 Related Work

Numerous schedulers have been proposed in the literature (see Chapter 2). Generally, schedulers can be categorised into LR scheduler and delay-based scheduler.

WRR and DRR schedulers are popular LR schedulers that use quantum to control fairness and provide protection between flows. WFQ scheduler is a well-known LR scheduler that provides a good emulation of the GPS. It guarantees both differentiated fairness and

delay between its queues. WFQ scheduler is comparable to DRR in terms of the ability to provide weighted sharing of bandwidth. However, it has a higher algorithmic complexity than DRR scheduler. With regulated traffic source, it is known that LR schedulers can provide a delay bound for a flow.

WDEDF scheduler does not approximate the GPS model. Like WRR and DRR schedulers, WDEDF scheduler uses quantum to control fairness and provide protection between flows. We have shown that when traffic sources are regulated, WDEDF scheduler can provide a delay bound that is analogous to WDRR scheduler but due to the forwarding rule of WDEDF scheduler, it has a delay bound that is statistically lower than WDRR scheduler. The forwarding rule imposes a higher algorithmic complexity than WDRR scheduler but lower algorithmic complexity than WFQ scheduler. The processing complexity of WDRR, WFQ and WDEDF schedulers are $O(1)$, $O(N)$ and $O(\log N)$, respectively, where N is equal to the total number of competing flows in the scheduler. Nevertheless, WFQ scheduler provides a lower delay bound as compared to WDEDF scheduler when the latter is used as a LR scheduler.

EDF scheduler is one of the best-known algorithms for real-time processing that has been applied to solve packet-scheduling problems. It has a algorithmic complexity of $O(\log N)$, where the size of N is equal to the total number of packets in the scheduler. Delay-EDD scheduler shows that delay guarantees can be achieved for delivery of real-time flows. Although EDF scheduler does not assure any differential fairness or flow protection, it is an optimal dynamic algorithm that assigns highest priority to active packet with the most immediate deadline. The assignment is dynamic because packet priority changes as time passes.

WDEDF scheduler can provide a delay guarantee like Delay-EDD scheduler with proper allocation of quantum for each flow. Like, Delay-EDD scheduler, it supports LR-coupled and LR-decoupled reservations. Even when used as delay-based scheduler, WDEDF

scheduler still supports delay differentiation, fairness, and flow protection for non-real-time flows. In addition, WDEDF scheduler attempts to minimise jitter for Jitter-Sensitive flows but it does not guarantee jitter bound like Jitter-EDD scheduler in which a jitter regulators that cause the scheduler to be non-work-conserving are used. Thus, WDEDF scheduler remains as a work-conserving scheduler.

Service Curve based Earliest Deadline First policy (SCED) [62] scheduler is another delay-based scheduler that computes deadline of packets based on service curves. It supports both LR-coupled and LR-decoupled reservations. Based on deadline allocation, SCED scheduler is able to provide fairness and flow protection. However, its algorithmic complexity, like Delay-EDD and Jitter-EDD schedulers are higher than WDEDF scheduler in which only the HOL packets are sorted as compared to the former whereby sorting involves all packets in the scheduler.

5.6 Summary

WDEDF scheduler is a non-preemptive, quantum-based scheduler that provides flow protection; throughput guarantees; and delay bound guarantee for flows that requires LR-coupled and LR-decoupled types of reservations. Flow that has low rate and intolerable delay requires LR-decoupled reservation.

LR schedulers, like WFQ and DRR schedulers, control delay bound of a queue by controlling only the bandwidth reservation. Thus, over-provisioning of resource is required by LR schedulers to support delay guarantees specified by LR-decoupled reservations.

WDEDF scheduler also classifies flows into Time-Critical, Jitter-Sensitive and Rate-Based classes; and uses a QoS forwarding rule to determine the next packet to be serviced by the scheduler. It provides non-preemptive priority service to Time-Critical queues. Packet from Jitter-Sensitive queues can be delayed by other packet if forwarding the latter will not result in the Jitter-Sensitive packet missing its deadline. This implicitly shifts the

delay of Jitter-Sensitive packet nearer to its expected departure time. As a quantum-based scheduler, WDEDF scheduler provides throughput guarantees for Rate-Based queues.

WDEDF scheduler provides throughput and delay bound guarantees at the cost of a higher algorithmic complexity as compared to DRR scheduler owing to the rearrangement of service order according to the deadline of the packets at the head of each queue. This rearrangement of service order is bounded by the maximum number of queues in the system. As compared to EDF scheduler, algorithmic complexity of WDEDF scheduler is much smaller as the algorithmic complexity of EDF scheduler is bounded by the number of packets in its queue. On top of that, WDEDF scheduler avoids the iterated deletion problem [36] that occurred in WFQ scheduler for computing a packet's finish number and unlike EDF scheduler, WDEDF scheduler provides throughput fairness among its queues. Last but not least, WDEDF scheduler can also be used in any QoS architecture whereby packets are delivered based on their deadline.

Chapter 6

Path and Oracle Discovery Protocol

In the previous chapters, we had discussed our solution for the data plane. We had shown the operation of EVTRS and the ability of WDEDF scheduler scheduling packet trains generated by edge routers. This chapter describes the control plane of our architecture. We developed PODP, a generic signalling protocol, for centralised management control. Centralised management control is first introduced by Nichols et al. [47] in the context of DiffServ. Our protocol includes route and Oracle discovery for flows that require reservation. It has four signalling options to cater to the different signalling requirements for different QoS architectures that adopt this centralised management control mechanism. For example, IntServ requires the discovery of each router along the route for a flow but DiffServ requires the pair of edge routers through which a flow enters and exits a domain.

This chapter is structured as follows. Section 6.2 provides an overview of PODP. Section 6.3 illustrates the operation of PODP. Section 6.4 highlights the essentials in implementing PODP messages and the detection and recovery of lost PODP messages. The format of PODP messages is presented in the Appendix B. Section 6.5 describes the im-

plementation of a PODP-enabled router. In Section 6.6, we formulate the storage and traffic requirements of PODP. Issues concerning the processing of PODP messages on the fly, trade-off and scalability will be addressed in this section. Finally, in Section 6.7 we compare PODP with the current signalling protocols and Section 6.8 summarises PODP.

6.1 Introduction

The IntServ architecture introduced an end-to-end signalling protocol - RSVP. It is a per-flow (hop-by-hop) out-of-band signalling which provides each affected routers with per-flow reservation information for admission control. It is a distributed resource allocation mechanism in which each RSVP-aware router makes admission control decisions, based on its own resource availability. Distributed resource allocation mechanism has scalability issue to address as it requires routers to perform additional control plane functions like making admission decisions for each reservation request; and keep and maintain state information for reservation details and signalling. Thus, the merging of control plane functions and data plane on the router is not scalable solution.

The DiffServ architecture, introduced an Oracle that makes admission control decisions and interacts with edge routers [47]. This Oracle is an agent that has sufficient knowledge of resource availability and network topology to make admission control decisions. A BB [63] can be classified as an Oracle. It can be considered as a centralised resource allocation mechanism that makes provisions for QoS reservations in a network domain. This mechanism requires a Network-to-Network Interface (NNI) signalling protocol for inter-communications between Oracles when a flow traverses more than one domain. Simple Inter-Bandwidth Broker Signaling (SIBBS) [64] [65] is an NNI signalling protocol designed for neighbouring BBs to exchange information or parameters. In addition, a Host-to-Network Interface (HNI) signalling protocol is required for Host to communicate with BB. This protocol is a subset of SIBBS. Centralised resource allocation mechanism moves the QoS

control from the data plane. This decoupling of the data plane and QoS control plane increases the scalability of QoS architectures.

A BB is required to have up-to-date knowledge on route paths in its domain and to identify neighbouring BBs [66][63] so that it can make correct decisions on admission of reserved flows. However, obtaining and maintaining knowledge of network topology in a domain is costly and most BB [63][67][68][69] implementations did not address this issue. Some BB implementations are static, e.g. [70], it is assumed that there is no change in the network link/path once the data is entered into the system. A more dynamic approach is for the BB to probe the router's forwarding tables and calculate the different paths. The Telia Bandwidth Broker [71] learns the topology from the Open Shortest Path First (OSPF) [72]. This BB interacts with the routers using Simple Network Management Protocol (SNMP) [73] which will generate massive traffic between BB and its routers during BB initialisation and updating of network topology. Hence, this increases the recovery delay after a BB is down. Thus, scalability is the major concern for the proposed schemes in identifying network topology of a domain for a centralised bandwidth reservation.

We developed PODP, a generic protocol, identifying routers and neighbouring Oracles that are involved in making provisions of any reservations on demand. This has the advantage of higher accuracy in determining the path as well as adapting to any changes. The purpose of this protocol is to facilitate the management of networks with centralised bandwidth reservations.

6.2 PODP Overview

The goal of the protocol is to discover the route path and neighbouring Oracles for each domain that a flow passes through so that resources can be appropriately allocated and accounted. The protocol is designed to be:

- Simple to implement;
- Minimal in overhead;
- Minimal modification to routers;
- Able to capture traffic dynamics.

Presently, some BB systems use static entry [70] where the network links and routes are manually entered. When a user requests for network resources, the BB would compute the path that the new flow would traverse and make the appropriate allocation. The problems with such systems are:

- Incorrect network information entry;
- Inability to capture network link changes;
- High computation complexity for BB to compute the path.

Another option is to periodically access the router routing tables in the domain. Based on the information gathered, the BB would be able to compute the links and paths in the network. This method ensures that the BB is up-to-date on the network topology. However, the overhead in transporting the information depends on the frequency of probing a router and the size of the domain, which could be high in a large domain.

The approach we have adopted is a reactive mechanism, using special packets, which are recognised by the routers. When the special packet passes through the routers, it would append its IP onto this special packet. If the packet size of this special packet reaches the minimum MTU, it would force the router to report the list of IP addresses to the BB. Thus, the BB would be able to know the path taken by the flow. Furthermore, this would enable a BB to capture the BB addresses in the neighbouring domain and negotiate between the BBs where necessary. The protocol provides options for either capturing the IP addresses of two adjacent oracles and a list of IP addresses of routers (forwarding path)

or the IP addresses of the ingress router and egress router as the path state of a reserved flow in a domain. Thus, PODP needs to support the following communication:

- **Router to Router** : Discovery packet that routers could identify and act accordingly.
- **Router to Oracle** : This is necessary for the router to inform the oracle of the path taken by a discovery packet. It should have the IP address and special identifier to make it unique.
- **Oracle to Oracle** : Need to tell neighbouring domain Oracle that a discovery packet from their domain has arrived in its domain.

Based on the requirements we have proposed a set of 4 message types :

- **Discovery Message (DM)**. Sent by the Source Host to activate the propagation of the other messages where appropriate.
- **Reporting Message (RM)**. Sent by appropriate routers to their domain Oracle so that the latter has a record of the path taken for the connection.
- **Notifying Message (NM)**. Used by an Oracle to notify the neighbouring domain Oracle that it has received a *DM* from the neighbour domain. It is only done when the Oracle receives a *RM* containing the IP address of the previous domain.
- **Recovery Message (RCM)**. Used when an erroneous condition occurs.

Fig. 6.1 shows a network diagram illustrating the flow of PODP messages across three domains. Note that both intra-domain and inter-domain signalling are involved in PODP. In each domain there is a number of components:

- **PODP-enabled routers**. These are ingress, egress and core routers in a domain that supports QoS service as well as PODP.

- **PODP hosts.** These are the hosts that use the PODP protocol to initiate access to network resources.
- **Oracle nodes.** These are the BBs which maintain the information on the path used for each connection as well as the control access to the network resources.

As shown in Fig. 6.1, the Source Host would send a *DM*. On reaching the PODP-enabled routers, the router will modify the appropriate field in the message and forward this message according to its routing table. At the same time a *RM* is sent to the domain Oracle. This process continues until the Destination Host receives the *DM* and sends a *RM* to its domain Oracle confirming its acceptance of the connection. The Oracle will then send a *NM* to its neighbouring Oracle, which will be forwarded back to the Source Domain Oracle. The PODP provides different options so that only specified routers are activated. The options available are:

- **Option 0 :** Activate only ingress and egress routers;
- **Option 1 :** Activate all routers;
- **Option 2 :** Activate only edge router at the specified domain;
- **Option 3 :** Activate all routers in the specified domain.

Option 0 will be the most commonly used option in a DiffServ environment as the BBs are only interested in identifying the ingress and egress routers. We have shown this in our BBDP. However there are occasions when one would like to get the entire path, in which case *option 1* would be used. EVTRS uses *option 1* as it requires the route information for each reserved flow. This information is used to adjust the bandwidth reservation on each core routers for each edge router. In addition, IntServ environment also fits into this scenario as it requires all affected routers to perform admission control for each reserved

flow. *Option 2* is used for locating reserved tunnels and *option 3* is used for finding the path in specific Autonomous System (AS) or domain.

Figure 6.1: PODP Reference Network

6.3 Operation of PODP

PODP consists of four messages namely: *DM*, *RM*, *NM* and *RCM*. The detailed formats of these messages are given in the Appendix B. The *DM* is sent to the Destination Host by the Source Host. The purpose of this message is to aid the Oracle to identify the path taken in each domain as well as the peer Oracles across network domains. The *RM* contains routing information that is required by the Oracle to update its path state table and to identify the IP address of the Oracle in the previous domain. This message is sent to the Oracle by routers in the same domain. The Oracle that receives the *first RM* from its router, sends a *NM* to the Oracle in the previous domain, informing it of the neighbouring Oracle through which the flow will traverse. Routers and BB react differently to these messages. Fig. 6.2 shows the flow chart of a router reacting to the *DM* and *RM*. Fig. 6.3 shows how BB reacts to *RM* and *NM*. The detailed flow sequence of the messages in PODP is given in Table 6.1.

Figure 6.2: A flow chart that shows router's response on receiving PODP messages.

Figure 6.3: A flow chart that shows BB 's response on receiving PODP messages

Table 6.1: Flow Sequence of PODP

Flow Number	
1	The Source Host in the Source Domain generates a <i>DM</i> and sends it to the Destination Host in the Sink Domain. The Source Host places its own IP address in the <i>SRL</i> field of the <i>DM</i> . The <i>PO</i> field of the <i>DM</i> is assigned to the IP of the Source Host's Oracle.
2	Assuming the <i>DM</i> 's packet size is smaller than the allowed MTU, the core router appends its IP address onto the <i>SRL</i> and updates the MTU field in the <i>DM</i> . After that, this <i>DM</i> is forwarded to the next hop.
3	The egress router overwrites the <i>SRL</i> and <i>PO</i> fields of the <i>DM</i> to its IP address and its Oracle's IP address, respectively. It also updates the MTU field before the message is forwarded to the ingress router of the Transit Domain.
3a	The egress router generates and sends a <i>RM</i> to its assigned Oracle upon receiving the <i>DM</i> .
4	The ingress router appends its IP address onto the <i>SRL</i> and updates the MTU field in the <i>DM</i> respectively. After that, this <i>DM</i> is forwarded to the next hop.
5	Assuming the <i>DM</i> 's packet size has reached the allowed MTU, the core router overwrites the <i>SRL</i> and <i>PO</i> fields of the <i>DM</i> with its IP address and its Oracle's IP address respectively. It also updates the MTU field in the <i>DM</i> before forwarding the message to the next hop.
5a	The core router generates and sends a <i>RM</i> to its assigned Oracle upon receiving the <i>DM</i> that has reached its MTU size.
6	The Oracle in the Transit Domain sends a <i>NM</i> to the Oracle in the Source Domain upon receiving the <i>RM</i> from the core router. This is triggered when the <i>PO</i> field in the <i>RM</i> is unequal to the IP address of the Oracle. The Oracle in the Source Domain therefore learns about the Oracle in the Transit Domain and updates its information base.
7	The egress router overwrites the <i>SRL</i> and <i>PO</i> fields of the <i>DM</i> with its IP address and its Oracle's IP address respectively. It also updates the MTU field before forwarding the message to the ingress router of the Sink Domain.
7a	The egress router generates and sends a <i>RM</i> to its assigned Oracle upon receiving the <i>DM</i> .
8	The ingress router appends its IP address onto the <i>SRL</i> and updates the MTU field in the <i>DM</i> , respectively. After that, this <i>DM</i> is forwarded to the next hop.
9	Assuming the <i>DM</i> 's packet size is smaller than the allowed MTU, the core router appends its IP address onto the <i>SRL</i> and updates the MTU field in the <i>DM</i> . After that, this <i>DM</i> is forwarded to the next hop.
10	The Destination Host generates and sends a <i>RM</i> to its assigned Oracle upon receiving the <i>DM</i> .
11	The Oracle in the Sink Domain sends a <i>NM</i> to the Oracle in the Transit Domain upon receiving the <i>RM</i> from the Destination Host. This is triggered when the <i>PO</i> field in the <i>RM</i> is different from the IP address of the Oracle. The Oracle in the Transit Domain therefore learns about the Oracle in the Sink Domain and updates its information base.

PODP provides BB with the source IP address, source port, destination IP address and destination port for identification of reservation flows. Source IP address and destination IP address are extracted in the IP header of *DM* while source port and destination port are extracted from the *DM* itself. These four tuples are used as a combined key to identify *RM*s and *NM*s. PODP does not restrict that all the four tuples must be used. Inter-BB protocol like SIBBS can just use the source IP address and destination IP address for the identity field (RARID) in its Receiver Acknowledgement Response (RAR) Message ¹.

6.3.1 Operation of Option 1

In the Source Domain (Fig. 6.4), the Source Host sends a *DM* to the Destination Host. The *DM* is sent with a IP Router Alert Option (RA) [74]. This option signals routers on the arrival of IP datagrams that need special processing. Therefore, routers that implement high-performance forwarding designs may be alerted on the need to process special packets. The core routers append its IP address onto the *Source Route List (SRL)* field. When its MTU size is smaller than that advertised in the *DM*, it will overwrite the MTU field and forward it to the next hop ². If the packet size of *DM* has reached the allowed MTU size, the core router sends a *RM* to the Oracle. After that, it replaces the *SRL* field and the *Previous Oracle (PO)* field in the *DM* with its IP address and the IP address of its Oracle respectively. The *DM* is then forwarded to the next hop.

In the case of the egress router, it performs similar tasks as a core router except that a *RM* must be sent to the Oracle. The Oracle, upon receiving a *RM*, creates a new entry in its routing table for the new reservation and updates the route path according to each *RM* it receives.

In the Transit Domain, the *DM* continues to trigger the routers to send *RM* to its domain's Oracle. The Transit Domain Oracle will send a *NM* to the Oracle in the Source

¹The RAR is used in SIBBS in response to a bandwidth reservation request.

²The next hop is determined by the underlying routing protocol.

Domain upon receiving the *first RM* from the ingress router in the Transit Domain. The Oracle in the Source Domain therefore learns about the Oracle in the Transit Domain. This information is updated in the information base of the Oracle in the Source Domain.

Likewise, in the Sink Domain, the *DM* continues to trigger the routers. The Oracle sends a *NM* to the Oracle in the Transit Domain upon receiving the *first RM* from the ingress router. The Destination Host also sends a *RM* to the Oracle to complete the updating of the Oracle's information base in the Sink Domain. Table 6.1 details the flow sequence of the PODP with reference to Fig. 6.4.

Figure 6.4: Operation of PODP

6.3.2 Operation of other options

The operation of *option 0* is similar to *option 1* except that core routers will not send *RM* to the Oracles when it receives a *DM*. Only ingress and egress routers will act on the *DM*. The ingress router will append its IP address onto the *SRL* of the *DM* and the egress router will generate a *RM* to the Oracle in its domain. This *RM* will trigger a *NM* from the Oracle.

Likewise, the operation of *options 2* and *option 3* are similar to the operation of *option 0* and *option 1*, respectively. The only difference lies in the IP filtering mechanism. It only focuses on the specific domain whose IP address matches that in the filter field of the *DM*. Routers will mask its own IP with the filters to determine if it should process the *DM*.

6.4 Discovery Message Essentials

The *DM* is a critical message of the protocol. To minimise overheads, the Source Host sends this raw socket message to the Destination Host with RA set and the single fragment (the second bit of the Fragment Flag in the IP packet) set to “1”, i.e., no fragmentation is allowed in the IP header. The following sections discuss the setting up of the *DM*.

6.4.1 Raw Socket / UDP Socket

The *DM* comprises a simple, single packet which uses raw socket or UDP socket for fast transmission. The raw socket is only one level of abstraction above the network frame which is very fast in forwarding, thus, minimising processing overheads. This allows raw packets to fly through the network at the physical network speed. Raw socket does not have any higher-level protocols overhead that can erode network throughput (e.g. TCP). UDP socket is a good alternative for raw socket as it is a “connectionless” protocol that allows fast forwarding. However, UDP sockets add an 8 *bytes* UDP header on the IP packet which will decrease the number of router’s IP addresses that can be stored in the *SRL* field (see Section 6.4.3).

6.4.2 IP Router Alert Option

Katz [74] describes an IP option type that alerts routers to examine the contents of an IP packet. This is useful for, but not limited to, new protocols with packets addressed to a destination and yet require relatively complex processing in routers along the path. The design goal of IP Router Alert Option (RA) [74] is to provide a mechanism in which routers can intercept packets not addressed to them directly, without incurring any significant performance penalty. Thus, by including the RA in the IP header of the *DM*, routers can intercept this message causing little or no performance penalty on the forwarding of normal data packets. We define the value “1” in the IP header’s RA field for *DM*. Thus, the router alert option for the *DM* is 10010100 00000100 0000000 00000001₂.

6.4.3 Maximum Transmission Unit

Every network has a certain characteristic that is specific to the medium in use on that network. One of the most important characteristics is the maximum amount of data that a network can carry in a single frame (called Maximum Transmission Unit- MTU). Postel [24] specifies that the maximum allowed MTU size is 65535 *bytes*, and that the minimum allowed MTU size is 68 *bytes*. No network should advertise or attempt to use a value that is greater than or less than those values.

The *DM* has to be examined by routers. This means that routers must not fragment the *DM*. Therefore, *DM* must be sent as a single IP packet. As the *DM* grows in its length at each hop due to the appending of IP address of router, there must be a maximum size and this is set at 68 *bytes* (i.e., 17 32-bit words). This value is the minimum MTU size that a network must support. This limitation on the *DM* therefore imposes a corresponding limitation on the number of router’s IP addresses that can be stored in the *SRL* field. With a MTU size of 68 *bytes*, using raw socket, and the *DM*, at maximum, can store IP addresses of eight routers for *options 0* and *option 1*. IP addresses of six routers can be stored for

options 2 and *option 3*. Likewise a MTU size of 68 *bytes*, using UDP socket, and the *DM*, at maximum, can store IP addresses of six routers for *options 0* and *option 1*. IP addresses of four routers can be stored for *options 2* and *option 3*. In any case, when an Internet Control Message Protocol (ICMP) [75] error message is received due to the oversized *DM*, the Source Host must reflect this error to its Oracle or take appropriate actions. It is to note that the size of MTU will not limit the discovery of the number of routers between a pair of edge routers as PODP-enabled core routers are designed to generate *RM* to their respective Oracle when the size of the *DM* reaches its MTU.

6.4.4 Refreshing

Similar to RSVP, the refreshing rate (R) of the *DM* is once every 30 *s*. Floyd and Jacobson [76] have shown that periodic messages generated by independent network nodes can become synchronised. This can lead to disruption in network services as the periodic messages contend with other network traffic for link and forwarding resources.

To avoid periodic message synchronisation, Source Host generates a *DM* randomly in the range of $[\frac{1}{2} \times R, \frac{3}{2} \times R]$. Each Oracle will set a down timer on every reservation. This timer will be reset upon receiving the *RM* used for refreshing. An Oracle must tear down reservations on its routers upon expiry of this timer. Refreshing is necessary to detect changes in the routing path for the reserved flow. These changes include changes of intra-routing in a domain that forward packets to another edge router in the same domain; broken link between edge routers; and malfunction of edge router and Oracles. In multicasting, refreshing is used as a mechanism to reserve bandwidth for newly joined branches, and tear down the reserved bandwidth on branches that have been pruned. Despite the reservation delay due to refreshing, the new Destination Host is able to receive the multicast packets at best effort rate until the required reservation is established.

6.4.5 Error Detection and Recovery of PODP

PODP uses IP raw sockets which are transmitted unreliably over the network. Packet losses can occur during the transmission of *DM*, *RM* or *NM* and destroy the entire signalling process. Oracle detects the loss of packets by setting a timeout on the updating of the path state for each reservation. If the Oracle is unable to gather a complete path state within the specified timeout, it will request the Source Host to retransmit a *DM* via the *RCM*.

6.5 PODP Prototype

Routers that support PODP are required to detect and manipulate *DM*. In addition, it is required to generate *RM* when necessary. As described in Section 6.3, these routers detect and process *DM* using RA. Routers will detect this option and process the *DM* in its software processing modules.

Figure 6.5: Functional Architecture of the PODP Daemon Modules

In our implementation, we used FreeBSD ³ as a PC router. This PC router is patched with an implementation of IP Option Packet Processing for FreeBSD Kernel [77]. The performances of this patch are detailed in its documentation [77].

³<http://www.freebsd.org>

Incoming IP packets that contain RA will trigger the PODP Daemon developed to handle the modification of *DM* and generation of *RM*. Fig. 6.5 shows the Functional Architecture of the PODP Daemon Modules interacting with Kernel Functions. The PODP Daemon Modules consist of the *Discovery Message Modifier* and the *Reporting Message Generator*. The *Discovery Message Modifier* decodes the *DM* received. If the *SRL* field is full, it sends the *DM* to the *Reporting Message Generator*. At the same instance, assuming this is an egress router and it has a smaller packet size than the MTU size of the *DM*, it computes the checksum of the *SRL* field, the *PO* field and the MTU field. After that, these three fields are overwritten with the IP address of the router, the IP address of its Oracle and the MTU of the router, respectively. Next, if UDP socket is used, the new checksum of the *SRL* field, the *PO* field and the MTU field are computed. The checksum of the IP packet is replaced as follows:

$$\begin{aligned}
 \text{Checksum of } IP \text{ Packet} = & \text{Old Checksum of IP Packet} - \text{Checksum (Old SRL)} \\
 & - \text{Checksum (Old PO)} - \text{Checksum (OLD MTU)} \\
 & + \text{Checksum (SRL)} + \text{Checksum (PO)} \\
 & + \text{Checksum (MTU)}
 \end{aligned} \tag{6.1}$$

Finally, the IP packet is placed back into the traffic and forwarded to the next hop. The *Reporting Message Generator* is responsible for generating *RM* to the Oracle in the same domain as the router. It contains a list of router's IP addresses, IP address of peer Oracle and MTU. This set of data is obtained from the *SRL* field, the *PO* field and the MTU field of the *DM*, respectively. The *RM Generator* generates the *RM* and forwards the packet to the Oracle via a UDP socket connection.

6.6 Evaluating PODP

In Section 6.6.1 and Section 6.6.2, we make computational comparison of PODP and a BB that continuously probes information from routers as described in Section 6.1. For naming, the latter method will be called *Method A* from this section onwards. Table 6.2 lists the terms used for computation. We also provide a computational performance of PODP in Section 6.6.3.

Table 6.2: Table of parameters used for evaluating PODP

Parameter	Abbreviation	Value Used for comparison
D_{Hop} is the average number of hops between a pair of edge routers in a domain.	D_{Hop}	3
R_{Hop} is the average number of hops from a router to the Oracle of a domain.	R_{Hop}	1
SR_{Hop} represents the interval a router has to send <i>Reporting Message</i> to Oracle.	SR_{Hop}	8
Length of <i>Discovery Message</i>	L_D	68 <i>bytes</i>
Length of <i>Reporting Message</i>	L_R	16 <i>bytes</i>
Length of <i>Notifying Message</i>	L_N	12 <i>bytes</i>
Number of entries of FIB of an core router ^a	L_{FT}	100000
Number of entries of FIB of an edge router ^b	$L_{BGP\ FT}$	103442
Length of each entry of the FIB	L_E	8 <i>bytes</i>
Average total number of Neighbouring Oracles	N_{Oracle}	2
Average total number of core routers in a domain	N_{IGP}	1
Average total number of edge routers in a domain	N_{BGP}	2
Length of Probing Message	L_{Probe}	8 <i>bytes</i>

^a This approximated value taken from [78] on Tuesday, 21st November 2000 at 23:04

^b This average value taken from [79] on Saturday, 27th October 2001 18:15:02 UTC+1000

6.6.1 Storage Requirements

PODP stores forwarding information of a reserved path on demand. Therefore, it stores: IP address of Source Host, two IP addresses of neighbouring Oracle and a list of

router's IP addresses. Approximately, it stores

$$(D_{Hop} + 3) \times 4 \text{ bytes} . \quad (6.2)$$

Method A stores Forwarding Information Bases (FIB) of all its routers and the IP addresses of neighbouring Oracles. Approximately, it stores

$$N_{IGP} \times (L_{FT} \times L_E) + N_{BGP} \times (L_{BGP\ FT} \times L_E) + (N_{Oracle} \times 4) \text{ bytes} . \quad (6.3)$$

Considering a simple domain consisting of two edge routers and one core router, *Method A* needs to store 2.34 Mbytes of data whereas PODP only stores 24 bytes of data. Therefore, the additional amount of data capacity required by *Method A* may be used to store 102,000 PODP reservations. Fig. 6.6 shows the linear growth of the storage capacity required by *Method A* against PODP with increasing number of core routers in the same forwarding path.

Figure 6.6: Storage requirements of *Method A* against PODP

6.6.2 Traffic generated at each Router

PODP *Option 1* is chosen for this comparison as it generates the most traffic among the four options and it is used by EVTRS.

$$Traffic_{PODP} \approx \sum^i L_D + \sum^j L_R + (L_N \times k) \quad (6.4)$$

where

$$i = 1 \text{ to } D_{Hop} + \min(D_{Hop}, SR_{Hop}) + 1$$

$$j = 1 \text{ to } \left\lceil \frac{D_{Hop}}{SR_{Hop}} \right\rceil + 1$$

$$k = R_{Hop} \text{ of Domain} + R_{Hop} \text{ of next Domain} + 1$$

$$\approx 2 \times R_{Hop} + 1 .$$

The first term includes traffic generated for *DM* in the domain itself and the initial hops in the next domain before a *RM* is sent. The second term represents the traffic generated for *RMs* and the third term represents the traffic generated for *NMs*.

On the other hand, the traffic incurred by *Method A* includes probing message and the transfer of forwarding table of each router in the domain.

$$Traffic_{Method A} \approx R_{Hop} \times N_{IGP} \times (L_{FT} \times L_E + L_{Probe}) \quad (6.5)$$

$$+ R_{Hop} \times N_{BGP} \times (L_{BGP FT} \times L_E + L_{Probe})$$

Using the values set out in Table 6.2, the total traffic generated for both methods is computed. *Method A* injects 4,055,112 bytes of traffic whereas PODP injects 528 bytes of traffic, which is significantly less. Furthermore the BB also requires less processing.

6.6.3 Discussion on the Performance of PODP

In this section, we will discuss on the computational cost of processing *DM* on the fly by giving numerical examples. We also discuss on the trade-off for refreshing the *DM*. Finally, we address the scalability of using BB.

Computation cost for processing *Discovery Message* on the fly

The computation cost for processing *DM* on the fly is of the same order of magnitude as the implementation of IP Router Alert Option [77]. Without losing generality, Pan and Schulzrinne [77] evaluated the performance of IP option implementation using an Intel Celeron 500 *MHz* PC with 10 *Mbps* Ethernet interfaces. Each IP alert packet took 124.64 *ms* for processing and non-intercepted packets took 44.59ms. Therefore, the system could forward a maximum of 8,023 IP option packets per second and 22,426 non-IP option packets per second.

To provide some insight on the computational cost of processing the messages on the fly, the experimental timing derived by Pan and Schulzrinne [77] is used to compute the processing load for 100,000 reserved flows at 30 *s* refresh rate. There will be 3,333 *DM* generated per second. These messages take up 0.4 *s* ($3,333 \times 124.64 \text{ ms}$) of the processing time while leaving the remaining 0.6 *s* for the routing of non-IP alert packets. Therefore, the *DM* took up 40% of the processing power. However, kernel performance is affected by CPU clock speed and caching architecture. The same experiment has been tested on an Intel Celeron 600 *MHz* PC and an IP alert packet took 60.28 *ms* for processing. Using a CPU that is 20% higher in clock speed reduces the processing load of *DM* from 40% to 20%.

In the worst case, the refresh interval is 15 *s* due to the de-synchronisation technique used, i.e., $\text{Refresh Rate} = \frac{1}{2} \times R$ (see Section 6.4.4). There will be 6,666 *DM* generated per second. This will take 40% of processing power for an Intel Celeron 600 *MHz*

PC. However, with the current advancement in processor speed, this overhead will become much less significant.

Trade-off for refreshing *Discovery Message*

The periodic update of *DM* poses an increase of network traffic. To give an insight of such overhead, consider a worst case condition where the packet size of the *DM* is at its maximum of 68 *bytes* (i.e., at a maximum of 8 routers between each pair of edge routers) and each *DM* is being sent at half of 30 *s*-intervals due to the de-synchronisation technique (see Section 6.4.4) used for 100,000 flows simultaneously.

Using the above assumption, 6,666 *DM* or $(6,666 \times 68 \times 8 = 3.6 \text{ Mb})$ of data will be generated per second. Let us also assume that all updates reach a router with a throughput of 155 *Mbps*, this worst-case overhead accounts for only about 2.33% of the router load. If Gigabit Routers are used for the core network, this overhead is negligible.

Scalability of Centralised Management Control

Throughout this chapter, BB has been discussed as a centralised management control mechanism. It increases the efficiency and scalability of the data plane as core routers are relieved from QoS control functions.

However, as a centralised mechanism, BB faces issues on reliability, robustness and scalability. To address these issues, BB can be viewed as a centralised mechanism conceptually. Distributed or hierarchical BB systems [80] can be designed to address the scalability issue of BB.

PODP on EVTRS

PODP is considered as a path-coupled signalling protocol in which signalling messages are routed only through the data path. As EVTRS addresses the QoS for the data

plane, control plane messages (PODP messages) can be treated in the following ways :

- (1) queued into the same class as the data packets; (2) treated as a best-effort packet; or
- (3) special priority class can be created to aggregate control plane messages. There are drawbacks for each method. When control plane messages are queued together with data packets, the reservation for the flow must take the extra bandwidth required by the control plane messages into considerations. When control plane packets are treated as BE packets, the response time for route changes and QoS re-establishment will be affected. If this response time is critical for the flow, a special priority class can be created to aggregate control plane messages at the expense of the utilisation of bandwidth.

6.7 Related Work

QoS signalling is part of QoS architecture when it is necessary to establish, maintain and remove reservation states in network nodes. RSVP is a per-flow (hop-by-hop) out-of-band signalling protocol for IntServ. IntServ deploys a distributed resource allocation mechanism. Each router makes admission control decisions based on its own resource availability. RSVP provides information on flow reservation for these routers. RSVP is a two-pass protocol where Source Host sends a *PATH* message to Destination Host. This *PATH* message creates a reverse path state on each router en route to the Destination Host. Conceptually, the reverse path states act like a “*route pin*” for a *RESV* message to en route back to the Source Host. The storing of reverse path states is not scalable. RSVP is categorised as receiver-initiated reservation as the routers based on the flow descriptor in the *RESV* message to set up the requested reservation.

YESSIR is another signalling protocol proposed for per-flow resource reservation. It promotes sender-initiated reservation, as many applications will not make full use of the benefits of receiver-initiated reservations. In addition, sender-initiated reservation fits better with policy and billing. It builds itself onto the Real-time Transport Protocol (RTP)

[81], using Real-time Control Protocol (RTCP) [82] as the in-band signalling for resource reservation.

PODP is designed for centralised reservation mechanism. It is a one-pass protocol for the data plane. It does not require any reverse path state to be kept on each router. The establishment of reservation is left to the BBs. For centralised management control, reservations can either be receiver or sender initiated. These are resolved in the control plane using protocols like SIBBS.

In the few of years after PODP was proposed and the writing of this thesis, researchers have made rapid contributed to signalling protocols. RSVP is extended for traffic engineering (ResSerVation Protocol - Traffic Enginnering (RSVPTE) [83]) that is used for MPLS. Most significantly, the IETF Working Group, Next Steps in Signalling (NSIS) , is currently working on a signalling framework. This work group is proposing an architectural framework for signalling protocols that contain signalling information about a data flow along its path in the network. This proposed framework decomposes the overall signalling protocol suite into a lower layer and upper layer. The lower layer is a generic layer that is responsible for moving signalling messages around and should be independent of any particular signalling applications. The behaviour of the upper layer is designed for adapting to specific signalling requirements. It contains functionalities such as message formats and sequences that are specific to a particular signalling application. Thus, the framework envisioned to support various signalling applications that need to install and/or manipulate state in the network. PODP may also be viewed as one of the signalling applications that can be adapted into the NSIS framework's upper layer where the behaviour of PODP is specified.

6.8 Summary

Centralised Oracles need to store and maintain knowledge of network topology and neighbouring oracles in a domain. This is a costly requirement as the amount of data in core and edge routers are dynamically changing and large. It requires large high capacity data storage and long setup time for oracles to obtain forwarding tables of all its routers. In addition, constant probing to obtain the forwarding tables of routers generates massive unnecessary traffic.

PODP is designed to facilitate the management of networks with centralised bandwidth reservations. It defines a generic reactive protocol that identifies routers and neighbouring Oracles that are involved in provisioning any bandwidth reservations for a centralised Oracle. It injects small non-fragmented datagram that triggers routers to inform their Oracle of the path state information. Reporting for new reservations information can rely on existing protocols like Common Open Policy Service (COPS) [84]. To reduce the interactions between routers and Oracle, a *source route list* is used, whereby routers append their IP addresses onto the datagram. Therefore routers and Oracles used for making provisions for reserved flows are stored on demand. Thus, it provides a more effective way of storing and maintaining forwarding paths for a centralised bandwidth reservation mechanism. PODP is able to respond to network path changes at minimum network overhead and storage requirements compared to other proposed methods. Apart from being the signalling protocol for EVTRS, PODP is adaptable to existing QoS architectures like DiffServ and IntServ. In addition, PODP can be used to discover routers at specified domain.

Lastly, recent network management shows that monitoring network topology between every edge pairs is a useful building block for managing DiffServ networks [85] and monitoring of QoS path for multicast networks [86] is also a necessity in supporting the deployment of multicast services on the Internet. PODP can be adapted for the discovery of these paths.

Chapter 7

Conclusion and Future Work

We conclude this dissertation by summarising the contributions made and charting future research directions for our architecture.

7.1 Contribution

The stateless nature of the Internet contributes to its success. It is scalable and robust as routers do not need to maintain fine grained state information. However, it only provides a minimalist service (Best-Effort). With the increasing range of network applications, like IP telephony, Video-On-Demand, Interactive TV and network games, a divergent set of network requirements is attached and this minimalist service is no longer adequate.

A stateful Quality-of-Service (QoS) architecture (e.g. Integrated Services) provides absolute QoS guarantees but it is not scalable and less robust as compared to a stateless QoS architecture (e.g. Differentiated Services). The latter, however, can only provide statistical QoS guarantees. In particular, a stateful network requires each router to maintain and manage per-flow state information in order to perform flow classification, scheduling and buffer management; while a stateless network increases its scalability by grouping and

classifying packets into service classes so that core routers service these packets according to their service classes (i.e., core routers have a small set of service classes to manage). CSFQ is a stateless QoS architecture that approximates services defined in Integrated Services.

The contribution of this dissertation is to bridge the provisioning of absolute delay guarantees in presence of throughput guarantees; and minimise jitter for the SCORE architecture that manages service classes using different queues as opposed to CSFQ which relies on a single queue.

To maintain the scalability and robustness of a QoS architecture, we adopt the SCORE network architecture and use DPS technique to embed state information in IP packets at edge routers, where the volume of traffic is smaller, for downstream core routers in which QoS control mechanisms are kept simple. We maintain the core router as stateless by using a *packet train* technique that handles an aggregation of packets from flows of a service class as a *packet train* whereby properties are defined for this *packet train* and embedded into each packet at the edge router. Core routers make use of the embedded *packet train* properties and the nature of the QoS guarantees needed by its service class to ensure delivery of all *packet trains* within their QoS specifications.

We illustrated the first Edge-based Virtual Time Reference System (EVTRS) for the SCORE network architecture. It maintains the properties of a *packet train* when this *packet train* is split into child *packet trains* on a router has multiple output links; computes state information to be embedded into each packets for a *packet train*; uses a virtual clock technique to enable flow extraction, computes instantaneous service rate and implements a granular packet dropping algorithm. Through mathematical induction and work examples, we showed that EVTRS converges and is observable. EVTRS can suffer from packet losses at the Medium Access Control layer which we have addressed using a detection and recovery scheme. We have also discussed the scalability of EVTRS by analysing the complexity of core routers to maintain states for each *packet train*; and the complexity of our granular

packet dropping algorithm.

To provide Network QoS performance guarantees, we defined a set of service class that includes Time-Critical, Jitter-Sensitive, Rate-Based and Best-Effort service classes; and put forward a new multi-service scheduling discipline, Weighted Deficit Earliest Departure First Scheduler (WDEDF), that provides delay guarantees (Latency-Rate-coupled and Latency-Rate-decoupled types of reservations) for Time-Critical and Jitter-Sensitive service classes, throughput guarantees for Rate-Based and Best-Effort service classes and jitter minimisation for Jitter-Sensitive service class. The Time-Critical service class provides absolute delay guarantees like the Guaranteed Service in Integrated Services; while the Jitter-Sensitive service class provides an absolute delay guarantees with jitter minimisation that can be used by Internet applications which require Expedited Forwarding service as proposed in Differentiated Services. The Rate-Based service class provides rate guarantees that suit Internet applications, like bulk data transfer applications, that require bandwidth guarantees. In addition, flow protection is imposed as our scheduling discipline uses a quantum control mechanism. We have presented WDEDF scheduler with both analytical and simulation results. In addition, we compared WDEDF scheduler with popular rate-based and delay-based scheduling disciplines, for example, Deficit Round Robin (DRR), Weighted Fair Queuing, Delay Earliest Due Date, Jitter-Earliest Due Date and Service Curve based Earliest Deadline First policy schedulers. WDEDF scheduler has a lower algorithmic complexity compared to these scheduling disciplines except for DRR scheduler. However, DRR scheduler cannot provide the various QoS guarantees as compared to WDEDF scheduler.

As a component of our architecture and a contribution to the development of centralised domain management, we developed the pioneer generic signalling protocol, Path and Oracle Discovery Protocol (PODP), that aids in triggering QoS reservation; and discovering route and peer Oracles for QoS-sensitive flows. The discovery of neighbouring Oracle facilitates the communication between neighbouring domain when negotiating for a service

contract during a reservation setup. We have shown numerically that PODP is able to respond to network path changes at minimum network overhead and storage requirements as compared to other proposed methods for Bandwidth Broker implementation. In addition, PODP can be adapted for existing QoS architectures.

7.2 Future Work

In the following sections, we identify several future research directions both for our architecture and Network QoS.

7.2.1 Scalability and Robustness

Although EVTRS is scalable and robust as compared to a stateful network domain, our solution is still affected by scalability and robustness issues. In EVTRS, edge router is required to perform per-flow classification and scheduling. This reduces the scalability of the edge routers. In addition, the maintenance of per-flow state reduces the robustness of edge routers. There is also a possibility that an edge or a core router may malfunction and inserts erroneous state information that can severely affect the delay guarantee of Time-Critical and Jitter-Sensitive service classes. This also reduces the robustness of our architectures.

A possible solution is to shift the complexity at the edge routers to the egress of a leaf domain, thus, relinquishing the formation of edge and core routers in a domain. However, this approach increases the complexity of the core routers as there are more *packet trains* to be managed. To counter this, it will be interesting to study how QoS can be provisioned with the classification and merging of *packet trains* generated by egress of leaf domains at edge routers.

7.2.2 Multicast

Our architecture presented in this dissertation is primarily designed for unicast traffic. In unicast environment, the amount of traffic generated is under the control of the traffic generator, however, in the multicast environment, the amount of traffic generated also depends on the number and relative location of the receivers. An extension of our architecture for multicast traffic will be challenging to the *packet train* maintenance as the multicast protocol adds a new packet to the existing *packet train* in the core routers. Our signalling protocol can be adapted for multicast as each refresh of the *Discovery Message* will inform the Oracle on the new multicast tree within the domain and new neighbouring Oracles. With a updated multicast tree, Oracle can inform affected core routers to supply ample bandwidth for the respective service classes.

7.2.3 Granular packet dropping and relative packet dropping

We have given an account on the proposal of a granular packet dropping algorithm in Chapter 4. An extension of our current work is to conduct simulations to study this algorithm and perform more analytical studies on the complexity of this algorithm.

In addition, our extension work includes adding a service class that provides isolation between different service classes, and qualitative loss differentiation between a *drop precedence* within each service class. This type of service class is proposed in the DiffServ architecture as AF service class.

7.2.4 Effect of EVTRS with flow control protocol

Our QoS architecture caters for UDP flows that is non-congestion-managed flows typically used in applications that use external conditions for its data rate. These flows rely on short queues in routers for low delay and jitter. On the other hand, a flow that uses TCP tends to derive its behaviour from a *network clock* where its sending rate is timed

against the instantaneous network load that is fed back from the receiving end. It sends packet bursts at a speed higher than the feedback rate and rely on the routers to absorb the burst. This caused large queues within routers if network efficiency is to be maintained. Generally, UDP traffic will have a negative effect on TCP throughput, likewise, TCP bursts may also affect UDP losses [87]. Some work is done to ensure fair bandwidth allocation between these two traffic types for the Differentiated Services [88] [89]; and to explore the effect of TCP traffic that has Expedited Forwarding service class in a Differentiated Services domain [90].

Separating UDP and TCP flows allows TCP queues to behave more predictably and it also increases the probability in avoiding congestion, thereby increasing the throughput efficiency. However, this increases the number of queues to be maintained by both edge and core routers.

7.2.5 Inter-operation with other QoS architectures

Diversity is always a factor for the Internet. Some networks will deploy a type of service technology while others not. In such a heterogeneous environment, inter-operation between neighbouring QoS architectures is essential to provide a seamless reservation at the control plane and data plane.

Inter-operation can take on three basic forms: (1) pre-programmed mapping of service classes between edge routers of neighbouring domains. This requires prior agreement of service contract between service providers. This method involves no hardware changes but changes to service mapping requires operator intervention. (2) inter-operation of signalling protocol whereby routers detect signalling messages from neighbouring domain and convert the request into the signalling protocol implemented in its domain. This method requires edge routers to be replaced with one that can recognise different signalling protocols. (3) use of an unified signalling for the control plane like Path and Oracle Discovery Protocol,

our proposed generic signalling protocol, in a QoS architecture that implements Integrated Services, Differentiated Services or other QoS architectures.

It will be useful to devise a service mapping scheme between different QoS architectures and study the implementation complexities for inter-operation of signalling protocols.

7.2.6 Other QoS requirements

Apart from network performance guarantees, there are other complementary QoS requirements like reliability, security and availability. Common security issues like data integrity; confidentiality; protection against denial of service attacks; and protection against service theft are security issues for QoS networks. In particular, protection against service theft is one of the factor that may contribute to the commercial deployment of QoS. In the context of data networks, there is not much studies in techniques for redundant provisioning of resources.

7.2.7 Bandwidth Broker

Our edge-based QoS architecture assumes the usage of a central bandwidth management mechanism - Bandwidth Broker(BB). However, there is much room for research work in this area. The pioneer work of BB started with Terzis et al. [47]. This work is carried out by the Internet2 community. They proposed the informal specification of a Bandwidth Broker [63] along with a Simple Inter-Bandwidth Broker Signaling [64]; and we added a signalling protocol, Path and Oracle Discovery Protocol [48] and a Secure Simple Inter-Bandwidth Broker Signaling [65].

However, there are still various issues to be resolved. These issues include: a Host-to-Network Interface for communication between end-users and BB, a policy based decision engine to make QoS provisioning decision, data storage definition for reservations and routing, intra-domain communication between routers and BB, QoS monitoring mechanisms

etc. A concrete finding for Bandwidth Broker may bring a big step to deployment of QoS. Without the realisation of BB, it is hard to promote dynamic service level agreement and per-user QoS request.

7.3 Final Remarks

In this dissertation, we developed another approach to achieve QoS guarantees for the SCORE network architecture. We show that our architecture is able to support absolute guarantees for Latency-Rate-coupled and Latency-Rate-decoupled reservations; throughput guarantees; jitter minimisation; and flow protection. Researchers have made vast contributions in QoS theories, architectures and schedulers over the past decade but these did not gain any popularity in the deployment of IP QoS. However, QoS is not dead as (1) “it is time for researches to reach out and include business, systems control and marketing expertise in the efforts to get IP QoS meaningfully deployed and used” [91]; and (2) with the fast evolution in technology, QoS will be a potential research area in other newly developed communication platform, for example, QoS for ad-hoc network is one of the current popular research area.

Bibliography

- [1] R. Braden, D. Clark, and S. Shenker, "Integrated services in the Internet architecture: an overview," *RFC 1633*, June 1994.
- [2] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism," in *Proc. ACM SIGCOMM*, Baltimore, MD, Aug. 17–20, 1992, pp. 14–26.
- [3] S. Blakem, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated service," *RFC 2475*, Dec. 1998.
- [4] B. Davie, A. Charny, J. Bennett, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis, "An expedited forwarding PHB (per-hop behaviour)," *RFC 3246*, Mar. 2002.
- [5] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding PHB group," *RFC 2597*, June 1999.
- [6] S. Brim, B. Carpenter, and F. L. Faucheur, "Per-Hop behavior identification codes," *RFC 2836*, May 2000.
- [7] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proc. ACM SIGCOMM*, Cambridge, MA, Aug. 30–03, 1999, pp. 81–94.
- [8] Z.-L. Zhang, Z. Duan, and Y. T. Hou, "Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services," *IEEE J. Select. Areas Commun. Special Issue on QoS*, vol. 18, no. 12, Dec. 2000.
- [9] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, pp. 33–46, Feb. 2003.
- [10] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344–357, June 1993.
- [11] J. Leigh, O. Yu, D. Schonfeld, R. Ansari, E. He, A. Nayak, J. Ge, N. Krishnaprasad, K. Park, Y. Cho, L. Hu, R. Fang, A. Verlo, L. Winkler, and T. A. DeFanti, "Adaptive networking for tele-immersion," in *Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop (IPT/EGVE)*, Stuttgart, Germany, May 16–18, 2001.

- [12] M. Claypool, D. LaPoint, and J. Winslow, "Network analysis of counter-strike and starcraft," in *Proc. IEEE International Performance, Computing, and Communications Conference*, Phoenix, Arizona, USA, Apr. 9–11, 2003.
- [13] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The effect of latency on user performance in Warcraft III," in *Proc. ACM SIGCOMM on NetGames*, Redwood City, CA, USA, May 2003.
- [14] J. Färber, "Network game traffic modelling," in *Proc. ACM SIGCOMM on NetGames*, Braunschweig, Germany, Apr. 16–17, 2002, pp. 53–57.
- [15] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP) - version 1 functional specification," *RFC 2205*, Sept. 1997.
- [16] E. A. Mankin, F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, and L. Zhang, "Resource reservation protocol (RSVP) version 1 applicability statement - some guidelines on deployment," *RFC 2208*, Sept. 1997.
- [17] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed Quality of Service," *RFC 2212*, Sept. 1997.
- [18] J. Wroclawski, "Specification of the controlled-load network element service," *RFC 2211*, Sept. 1997.
- [19] J. Postel and J. Reynolds, "File transfer protocol (FTP)," *RFC 959*, Oct. 1985.
- [20] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," *RFC 3031*, Jan. 2001.
- [21] N. K., B. S., B. F., and D. Black, "Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers," *RFC 2474*, Dec. 1998.
- [22] H. Ohnishi, T. Okada, and K. Noguchi, "Flow control schemes and delay/loss tradeoff in ATM networks," *IEEE J. Select. Areas Commun.*, vol. 6, no. 9, pp. 1609–1616, Dec. 1998.
- [23] B. Gleeson, A. Lin, J. Heinanen, and G. A. A. Malis, "A framework for IP based virtual private networks," *RFC 2764*, Feb. 2000.
- [24] J. Postel, "Internet protocol DARPA internet program protocol specification," *RFC 791*, Sept. 1981.
- [25] J. Heinanen and R. Guerin, "A two rate three color marker," *RFC 2698*, Sept. 1999.
- [26] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Networking*, vol. 3, no. 4, pp. 365–386, Aug. 1995.
- [27] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on queue management and congestion avoidance in the Internet," *RFC 2309*, Apr. 1998.

- [28] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proc. ACM SIGCOMM*, Cambridge, Massachusetts, United States, Aug. 30–3, 1999, pp. 81–94.
- [29] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Proc. ACM SIGCOMM*, Cambridge, MA, Aug. 30–03, 1999, pp. 147–160.
- [30] J. Postel, "Transmission control protocol," *RFC 793*, Sept. 1981.
- [31] —, "User datagram protocol," *RFC 768*, Aug. 1980.
- [32] K. Dovrolis, M. P. Vadam, and P. Ramanathan, "The selection of the token bucket parameters in the IETF guaranteed service class," University of Wisconsin-Madison, Tech. Rep., Nov. 1997.
- [33] L. Kleinrock, *Queueing Systems*. New York, Wiley, 1976, vol. 2.
- [34] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1265–1279, Oct. 1991.
- [35] M. Shreedhar and G. Varghese, "Efficient fair queuing using Deficit Round Robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 375–385, June 1996.
- [36] S. Keshav, "On the efficient implementation of fair queuing," *Journal of Internetworking Research and Experience*, vol. 2, no. 3, Sept. 1991.
- [37] S. Golestani, "A self clocked fair queuing scheme for broadband applications," in *Proc. IEEE INFOCOM*, vol. 2, Toronto, Ontario, Canada, June 12–16, 1994, pp. 636–646.
- [38] H. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proc. ACM SIGCOMM*, Philadelphia, Pennsylvania, United States, Sept. 26–28, 1990, pp. 19–29.
- [39] M. S. Fineberg and O. Serlin, "Multiprogramming for hybrid communication," in *Proc. AFIPS Fall Joint Computer Conference*, Thompson, Washington, D.C., 1967.
- [40] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area network," *IEEE J. Select. Areas Commun.*, vol. 8, no. 3, pp. 368–379, Apr. 1990.
- [41] D. C. Verma, H. Zhang, and D. Ferrari, "Delay jitter control for real-time communication in a packet switching network," in *Proc. TriCom*, Sept. 5–8, 1991, pp. 35–43.
- [42] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *Journal of High Speed Networks, Special Issue on QoS*, vol. 3, no. 4, Oct. 1994.
- [43] H. Zhang and S. Keshav, "Comparsion of rate-based service disciplines," in *Proc. ACM SIGCOMM*, Zurich, Switzerland, Sept. 3–6, 1991, pp. 113 – 121.
- [44] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Networking*, vol. 6, no. 5, pp. 611–624, Oct. 1998.

- [45] T. M. Lim, B. S. Lee, and C. K. Yeo, "EVTRS: Edge-based virtual time reference system," in *Proc. IASTED International Conference on Communications, Internet, & Information Technology*, Scottsdale, Az, USA, Nov. 17–19, 2003, pp. 66–71.
- [46] —, "Weighted deficit earliest departure first scheduling," in *Proc. IEEE International Conference on Networks*, Sydney, Sept. 28–1, 2003, pp. 437–442.
- [47] A. Terzis, L. Wang, J. Ogawa, and L. Zhang, "A two-bit differentiated services architecture for the internet," *RFC 2638*, July 1999.
- [48] T. M. Lim, B. S. Lee, and C. K. Yeo, "Path and oracle discovery protocol for centralized bandwidth reservation mechanisms," *Journal of Network and Systems Management*, vol. 11, no. 4, pp. 447–468, Dec. 2003.
- [49] R. Braden, D. Clark, and S. Shenker, "Network simulator - ns (version 2)," UCB/LBNL/VINT, <http://www.isi.edu/nsnam/ns>.
- [50] R. Jain and S. A. Routhier, "Packet trains - measurements and a new model for computer network traffic," *IEEE J. Select. Areas Commun. Special Issue on QoS*, vol. 4, no. 6, pp. 986–995, Sept. 1986.
- [51] C. Li and E. W. Knightly, "Coordinated multihop scheduling: A framework for end-to-end services," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, pp. 776–789, Dec. 2002.
- [52] K. S. Park and R. V. Kenyon, "Effects of network characteristics on human performance in a collaborative virtual environment," in *Proc. IEEE Virtual Reality*, Houston, Texas, Mar. 13–17, 1999, pp. 104–111.
- [53] T. M. Lim, B. S. Lee, and C. K. Yeo, "BBDP: Bandwidth broker discovery protocol," in *Proc. IEEE International Conference on Information, Communication & Signal Processing*, Singapore, Oct. 15–18, 2001.
- [54] P. Pan and H. Schulzrinne, "YESSIR: A simple reservation mechanism for the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, pp. 89–101, Apr. 1999.
- [55] —, "Lightweight resource reservation signaling: Design, performance and implementation," Bell Labs, Tech. Rep. 10009669-03, July 2000.
- [56] Z. Cao, Z. Wang, and E. Zegura, "Rainbow fair queueing: fair bandwidth sharing without per-flow state," in *Proc. IEEE INFOCOM*, vol. 2, Tel-Aviv, Israel, Mar. 26–30, 2000, pp. 922–931.
- [57] M. Y. L. Wong and C. K. Li, "Low computational complexity weighted queuing using weighted Deficit Round Robin," in *Proc. IASTED Application Information*, Innsbruck, Austria, Feb. 18–21, 2002.
- [58] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

- [59] P. McKenney, "Stochastic fairness queuing," *Journal of Internetworking Research and Experience*, vol. 2, pp. 113–131, Jan. 1991.
- [60] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched network," *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 1096–1105, Feb. 1994.
- [61] P. Losi, "ns2 WFQ module," <http://www.pao.lombardiacom.it/wfq.html>.
- [62] H. Sariowan, R. L. Cruz, and G. C. Polyzos, "SCED: a generalized scheduling policy for guaranteeing quality-of-service," *IEEE/ACM Trans. Networking*, vol. 7, no. 5, pp. 669–684, Oct. 1999.
- [63] B. Teitelbaum and P. Chimento, "QBone signalling design team final report," Internet2, Tech. Rep., July 2002, <http://qos.internet2.edu/wg/documents-informational/20020709-chimento-et-al-qbone-signaling>.
- [64] R. Geib, "SIBBS message codes," July 2002, <http://qos.internet2.edu/wg/documents-informational/20020709-chimento-et-al-signaling/draft-BB-ph0-codings-08.html>.
- [65] B. S. Lee, C. K. Yeo, W. K. Woo, T. M. Lim, B. H. Lim, Y. He, and J. Song, "Secure communications between bandwidth brokers," *ACM SIGOPS Operating Systems Review*, vol. 38, no. 1, pp. 43–57, Jan. 2004.
- [66] M. A. Bauer and H. A. Akhand, "Managing Quality-of-Service in Internet applications using differentiated services," *Journal of Network and Systems Management*, vol. 10, no. 1, pp. 39–62, 2002.
- [67] J. Arnold, "Merit bandwidth broker," Nov. 1999, http://www.merit.edu/working.groups/i2-qbone-bb/inter-op/bb_demo_Merit/indexhtm.
- [68] J. Evans and K. Dhananjaya, "Bandwidth broker implementation, version 1.0," Sept. 1999, http://www.merit.edu/working.groups/i2-qbone-bb/inter-op/bb_proposal_kuhtm.
- [69] G. Stattenberger and T. Braun, "Performance of a bandwidth broker for DiffServ networks," in *Proc. Kommunikation in verteilten Systemen*, Leipzig, Germany, Mar. 25–28, 2003.
- [70] M. Hashmani and M. Yoshida, "ENICOMs bandwidth broker," in *Sym. on Applications and the Internet Workshop (SAINT)*, San Diego, California, Jan. 2001, pp. 213–220.
- [71] O. Schelen and N. Borg, "Telia bandwidth broker," Nov. 1999, http://www.merit.edu/working.groups/i2-qbone-bb/inter-op/bb_proposal_teliahtm.
- [72] J. Moy, "Open shortest path first - OSPF version 2," *RFC 2328*, Apr. 1998.
- [73] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," *RFC 1157*, Sept. 1990.
- [74] D. Katz, "IP router alert option," *RFC 2113*, Feb. 1997.

- [75] J. Postel, "Internet control message protocol," *RFC 792*, Sept. 1981.
- [76] S. Floyd and V. Jacobson, "Synchronization of periodic routing messages," *IEEE/ACM Trans. Networking*, vol. 2, no. 2, pp. 112–136, Apr. 1994.
- [77] P. Pan and H. Schulzrinne, "PF_IPOPTION: A kernel extension for IP option packet processing," Bell Labs, Tech. Rep. 10009669-02, June 2000.
- [78] "Telstra - BGP table statistics," <http://www.telstra.net/ops/bgptable.html>.
- [79] "The london internet exchange," <http://www2.linx.net/info/combined.1.html>.
- [80] Z.-L. Zhang, Z. Duan, and Y. T. Hou, "On scalable design of bandwidth brokers," *IEICE Transaction on Communications*, vol. E84-B, no. 8, Aug. 2001.
- [81] H. Schulzrinne, G. Fokus, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," *RFC 1889*, Jan. 1996.
- [82] T. Turetti and C. Huitema, "RTP payload format for H.261 video streams," *RFC 2032*, Oct. 1996.
- [83] D. Awduche, L. Berger, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP tunnels," *RFC 3209*, Nov. 2001.
- [84] D. Durham, J. B. Ed, R. Cohen, S. H. R. Rajan, and A. Sastry, "The COPS (common open policy service) protocol," *RFC 2748*, Jan. 2000.
- [85] J.-Y. Kim, J.-K. Hong, and T.-S. Choi, "Monitoring edge-to-edge traffic aggregates in differentiated services networks," *Journal of Network and Systems Management*, vol. 9, no. 3, pp. 267–291, Sept. 2001.
- [86] E. Al-Shaer and Y. Tang, "QoS path monitoring for multicast networks," *Journal of Network and Systems Management*, vol. 10, no. 3, pp. 357–381, Sept. 2002.
- [87] H. Sawashima, Y. Hori, H. Sunahara, and Y. Oie, "Characteristics of UDP packet loss: Effect of TCP traffic," in *Proc. Internet Society Conference INET*, Kuala Lumpur, Malaysia, June 24–27, 1997.
- [88] P. Piedad, B. Nandy, and S. Nandy, "The dynamics of TCP and UDP interaction in a Differentiated Services IP-QoS network," in *Proc. Canadian Conference on Broadband Research CCBR*, Ottawa, Nov. 7 – 9, 1999.
- [89] S.-H. Lee, S.-J. Seok, S.-J. Lee, and C.-H. Kang, "Study of TCP and UDP flows in a differentiated services network using two markers system," in *Proc. IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, Chicago IL, USA, Oct. 29–1, 2001, pp. 198–203.
- [90] J. Kaur and H. M. Vin, "Effect of higher priority EF traffic on TCP throughput and fairness," University of Texas at Austin, Tech. Rep. TR2001-1, July 2000.
- [91] G. J. Armitage, "Revisiting IP QoS: why do we care, what have we learned? ACM SIGCOMM 2003 RIPQOS workshop report," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 5, pp. 81–88, Oct. 2003.

Appendix A

Performance bound of WDEDF when used as a LR scheduler

Stiliadis et al. [44] have classified DRR as a LR scheduler and have derived a worst-case delay bound for DRR. The delay bound shown is only applicable when queues make equal reservation rate on the DRR scheduler, according to assumptions made by the authors when deriving the delay bound. However, it is not applicable for WDRR. In the following, we derived the worst-case delay bound for WDEDF which is analogue to WDRR.

Corollary 2 *Let t_0 be the beginning of a backlogged period of queue q in the WDEDF scheduler. Then, at any time t , during this backlogged period, the service offered to*

$$S_q(t_0, t) \geq \max(0, R_q(t - t_0 - (\frac{\Omega^{MAX} - \omega_q - L_q^{MAX}}{C^l} + \frac{L_q^{MAX}}{R_q}))) \quad (A.1)$$

and the delay of queue q is equal to

$$\frac{\Omega^{MAX} - \omega_q - L_q^{MAX}}{C^l} + \frac{L_q^{MAX}}{R_q} \quad (A.2)$$

where $\Omega^{MAX} = \Omega + \sum_i^Q L_v^{MAX}$ in which Q is the total number of queues.

Proof: Let queue q becomes backlogged at time t_0 , and t_ν be the time that a service round finishes in the WDEDF algorithm. From Theorem 8, if a queue is continuously backlogged in the interval (t, t_ν) , then at the end of the ν th service round, The service offered to queue q is

$$S_q \geq \nu\omega_q - DC_q^\nu. \quad (A.3)$$

Let Q be the total number of queues in the WDEDF scheduler. For each service round interval $(t_{\nu-1}, t_\nu]$,

$$t_\nu - t_{\nu-1} \leq \frac{\Omega + \sum_{i=1}^Q DC_i^{\nu-1} - \sum_{i=1}^Q DC_i^\nu}{C^l}.$$

By summing over ν number of service rounds,

$$\begin{aligned} t_\nu - t_0 &\leq \frac{\nu(\Omega + \sum_{i=1}^Q DC_i^{\nu-1} - \sum_{i=1}^Q DC_i^\nu)}{C^l} \\ &\leq \frac{\nu\Omega}{C^l} + \frac{\sum_{i=1}^Q L_i^{MAX}}{C^l}. \end{aligned}$$

The above inequality holds as the value of DC is always non-negative and less than L^{MAX} as shown in Lemma 2. Let $L_\nu^{MAX} = \frac{\sum_{i=1}^Q L_i^{MAX}}{C^l}$ and make ν as the subject,

$$\begin{aligned} \nu &\geq \frac{(t_\nu - t_0 - L_\nu^{MAX})C^l}{\Omega} \\ &\geq \frac{(t_\nu - t_0)C^l}{\Omega} - \frac{L_\nu^{MAX}}{\Omega}. \end{aligned}$$

Substituting ν into Eqn. A.3,

$$\begin{aligned} S_q &\geq \nu\omega_q - DC_q^\nu \\ &\geq \omega_q \left(\frac{(t_\nu - t_0)C^l}{\Omega} - \frac{L_\nu^{MAX}}{\Omega} \right) - DC_q^\nu \\ &\geq R_q(t_\nu - t_0 - \frac{L_\nu^{MAX}}{C^l}) - DC_q^\nu \end{aligned}$$

In the worst-case, queue q is the last to receive service during the ν service round.

Case 1 Queue q transmits more than ω_q during the ν^{th} round. The maximum number of bytes sent is bounded by $\omega_q + L_q^{MAX} - DC_q$. Therefore, for any time from the beginning of the ν^{th} round until the HOL packet of queue q starts being serviced is,

$$\begin{aligned} S_q(t_0, t) &\geq \max(0, R_q(t_\nu - t_0 - \frac{L_\nu^{MAX}}{C^l}) - DC_q^\nu - (\omega_q + L_q^{MAX} - DC_q)) \\ &\geq \max(0, R_q(t_\nu - t_0 - \frac{L_\nu^{MAX}}{C^l} - \frac{\Omega}{C^l} - \frac{L_q^{MAX}}{R_q})) \\ &\geq \max(0, R_q(t - t_0 + \frac{\omega_q + DC_q^\nu}{C^l} - \frac{L_\nu^{MAX}}{C^l} - \frac{\Omega}{C^l} - \frac{L_q^{MAX}}{R_q})) \\ &\geq \max(0, R_q(t - t_0 - (\frac{\Omega^{MAX} - \omega_q - L_q^{MAX}}{C^l} + \frac{L_q^{MAX}}{R_q}))) \end{aligned}$$

where $\Omega^{MAX} = \Omega + L_\nu^{MAX}$. The above inequality holds from the fact that at least ω_q bytes are serviced from queue q in the ν^{th} service round. Therefore, $t \leq t_\nu - (\frac{\omega_q + DC_q^\nu}{C^l})$ receives service at a rate of C^l when they start being serviced.

Case 2 Queue q transmits less than or equal to ω_q during the ν^{th} round. The maximum number of bytes sent is bounded by $\omega_q - DC_q$. Therefore, for any time from the beginning of the ν^{th} round until the HOL packet of queue q starts being serviced is,

$$S_q(t_0, t) \geq \max(0, R_q(t_\nu - t_0 - \frac{L_\nu^{MAX}}{C^l}) - DC_q^\nu - (\omega_q - DC_q))$$

$$\begin{aligned}
&\geq \max(0, R_q(t_\nu - t_0 - \frac{L_\nu^{MAX}}{C^l} - \frac{\Omega}{C^l})) \\
&\geq \max(0, R_q(t - t_0 + \frac{\omega_q + DC_q^\nu}{C^l} - \frac{L_\nu^{MAX}}{C^l} - \frac{\Omega}{C^l})) \\
&\geq \max(0, R_q(t - t_0 - (\frac{\Omega^{MAX} - \omega_q - L_q^{MAX}}{C^l})))
\end{aligned}$$

The above inequality is smaller than that derived in Case 1. Thus, the case when queue q having transmitted less than ω_q bytes in the last service round is covered.

Case 3 (DRR case) When all queues make equal reservations, i.e., the quantum value of all queues are the same and $\omega^{MIN} = L^{MTU}$. From Eqn. A.1,

$$\begin{aligned}
S_q(t_0, t) &\geq \max(0, R_q(t - t_0 - (\frac{\Omega^{MAX} - \omega_q - L_q^{MAX}}{C^l} + \frac{L_q^{MAX}}{R_q}))) \\
&\geq \max(0, R_q(t - t_0 - (\frac{2\Omega - \omega_q - \omega_q}{C^l} + \frac{\omega_q}{R_q}))) \\
&\geq \max(0, R_q(t - t_0 - (\frac{2\Omega - 2\omega_q}{C^l} + \frac{\omega_q\Omega}{\omega_q C^l}))) \\
&\geq \max(0, R_q(t - t_0 - (\frac{3\Omega - 2\omega_q}{C^l})))
\end{aligned}$$

This worst-case delay bound is the same as the worst-case delay bound derived for DRR.

■

Appendix B

Specifications for PODP Messages

B.1 *Discovery Message*

The *DM* is sent to the Destination Host by the Source Host as shown in Fig. 6.4. The purpose of this message is to identify PODP-enabled routers along the path taken in each domain and peer Oracles across network domains for an end-to-end reservation. The size of the packet must not exceed the MTU size to prevent packet fragmentation, otherwise it will create false trigger in the router, thus mis-informing the BB of a multiple flow instead of a single flow. Table B.1 shows the *DM* format. Please note that some of the fields are optional. Each *DM* is uniquely identified by a combination of four partial keys. They are source IP address, source port, destination IP address and destination port.

Table B.1: *Discovery Message*

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
RT										Reserved										Source_Route_Count										MTU									
PREVIOUS_ORACLE																																							
SENDER_PORT																				DESTINATION_PORT																			
SOURCE_ROUTE_LIST																																							
SENDER_DOMAIN_FILTER (Optional)																																							
RECEIVER_DOMAIN_FILTER (Optional)																																							

Field	Explanation
RT	<p><i>Router Triggering</i></p> <ol style="list-style-type: none"> 0 Only Edge Routers will respond to the <i>DM</i>. 1 All Routers will respond to the <i>DM</i>. 2 Edge Routers in the domain matching the corresponding filter will respond to the <i>DM</i>. 3 All Routers in the domain matching the corresponding filter will respond to the <i>DM</i>.
Source_Route_Count	This field contains the total number of routers' IP address that appears in the SOURCE_ROUTE_LIST.
MTU	This field (in bytes) contains the MTU of the flow. Each router will advertise its MTU on this field if its MTU is smaller than the original MTU of the message i.e., $MTU = \min(Received\ MTU, Router's\ MTU)$.
PREVIOUS_ORACLE (PO)	This field contains the IP address of the Oracle of the previous host or router. It is altered by routers that are ready to send a <i>RM</i> before the <i>DM</i> is forwarded to the next hop.
SENDER_PORT	This field contains the port number of the Source Host and is used as a partial key for the <i>RCM</i> .
DESTINATION_PORT	This field contains the port number of the Destination Host and is used as a partial key for the <i>RCM</i> .
SOURCE_ROUTE_LIST (SRL)	This field contains a list of IP addresses of routers along the transmission path. It is altered by routers before the <i>DM</i> is forwarded to the next hop.
SENDER_DOMAIN_FILTER (Optional)	This field contains the IP filter for the Source Domain. It must be defined when RT is set to 2 or 3. It must not be defined when RT is set to 0 or 1.
RECEIVER_DOMAIN_FILTER (Optional)	This field contains the IP filter for the Sink Domain. It must be defined when RT is set to 2 or 3. It must not be defined when RT is set to 0 or 1.

B.2 Reporting Message

Table B.2: Reporting Message

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Reserved										Source_Route_Count										MTU																			
SENDER_IP																																							
DESTINATION_IP																																							
SENDER_PORT															DESTINATION_PORT																								
PREVIOUS_ORACLE																																							
SOURCE_ROUTE_LIST																																							

Field	Explanation
Source_Route_Count	This field contains the total number of routers' IP address that appears in the <i>SRL</i> .
MTU	This field (in bytes) contains the MTU of the flow. The content of this field is duplicated from the content of the MTU in the <i>DM</i> received by the router.
SENDER_IP	This field contains the IP address of the Source Host. It is used as a unique key for the <i>RM</i> . The content of this field is duplicated from the content of SENDER_IP in the <i>DM</i> received by the router.
DESTINATION_IP	This field contains the IP address of the Destination Host and is used as a partial key for the <i>RCM</i> .
SENDER_PORT	This field contains the port number of the Source Host and is used as a partial key for the <i>RCM</i> .
DESTINATION_PORT	This field contains the port number of the Destination Host and is used as a partial key for the <i>RCM</i> .
PREVIOUS_ORACLE (<i>PO</i>)	This field contains the IP address of the Oracle of the previous host or router. The content of this field is duplicated from the content of <i>PO</i> in the <i>DM</i> received by the router.
SOURCE_ROUTE_LIST (<i>SRL</i>)	This field contains a list of IP addresses of the routers along the transmission path. The content of this field is duplicated from the <i>SRL</i> of the <i>DM</i> received by the router.

The *RM* contains routing information that is required by an Oracle to update its new path table and identify the IP address of the Oracle in the previous domain. The *RM* is sent by routers to the Oracle in the same domain as shown in Fig. 6.4. Table B.2 shows the format of the *RM*. The *RM* format serves as a template that may be merged with existing protocol like COPS, for more reliable transfer.

B.3 Notifying Message

Table B.3: Notifying Message

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
RT		Reserved								Source_Route_Count								MTU																					
PREVIOUS_ORACLE																																							
SENDER_PORT																DESTINATION_PORT																							
SOURCE_ROUTE_LIST																																							
-																																							
SENDER_DOMAIN_FILTER (Optional)																																							
RECEIVER_DOMAIN_FILTER (Optional)																																							

Field	Explanation
SENDER_IP	This field contains the IP address of the Source Host. It is used as a unique key for the <i>NM</i> . The content of this field is duplicated from the content of SENDER_IP in the <i>RM</i> received by the router.
DESTINATION_IP	This field contains the IP address of the Destination Host and is used as a partial key for the <i>RCM</i> .
SENDER_PORT	This field contains the port number of the Source Host and is used as a partial key for the <i>RCM</i> .
DESTINATION_PORT	This field contains the port number of the Destination Host and is used as a partial key for the <i>RCM</i> .
PREVIOUS_ORACLE	This field contains the IP address of the Oracle of the previous host or router. The content of this field is duplicated from the content of PO in the <i>RM</i> received by the Oracle.
ORACLE_IP	This field contains the IP address of the domain's Oracle.

The Oracle receives a *RM* from its router and sends a *NM* to the Oracle in the previous domain. The *NM* contains information that is required by the Oracle in the previous domain to identify the IP address of the Oracle in the next domain. Table B.3 outlines the format of the *NM*. It is necessary for an Oracle to know the neighbouring domain that a flow would traverse. Thus, when an Oracle receives a *RM* with a *PO* field value which is different from its IP address, it will send a *NM* to the Oracle specified in the *PO* field. In this manner, the Oracle from which the flow originated would be able to find the appropriate neighbour Oracle, by looking at this Oracle IP field, to which it needs to negotiate for access.

The *NM* format serves as a template that can be merged with existing NNI protocol like SIBBS for more reliable transfer.

B.4 Recovery Message

Table B.4: Recovery Message

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
SENDER_IP										DESTINATION_IP										PREVIOUS_ORACLE											
SENDER_PORT										DESTINATION_PORT										ORACLE_IP											

Field	Explanation
Recovery type	List of Recovery types: 1 Retransmission of DM. 2-255 Reserved.
SENDER_IP	This field contains the IP address of the Source Host and is used as a unique key for the RCM.
ORACLE_IP	This field contains the IP address of Oracle.

This message is sent to the host that requested for the reservation when the Oracle is unable to complete its routing table within a specified time. The IP address of this host is found on the SENDER_IP field in the RM. This may be a result of transmission error in the DM, RM or NM. Table B.4 outlines the format of the RCM.

The RCM format serves as a template that may be merged with existing HNI protocol for more reliable transfer. In addition, an incomplete path state may be added as an exception or reservation failure state in an existing NNI.