Quantifiable Service Differentiation
for
Packet Networks

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Computer Science

by

Nicolas Christin

August 2003

Approvals

This dissertation is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Computer Science

Nicolas Christin

Approved:

Jörg Liebeherr (Advisor)	John A. Stankovic (Chair)
Tarek F. Abdelzaher	Stephen D. Patek (Minor Representative)
Victor Firoiu	Alfred C. Weaver

Accepted by the School of Engineering and Applied Science:

Richard W. Miksad (Dean)

August 2003

Abstract

In this dissertation, we present a novel service architecture for the Internet, which reconciles application demand for strong service guarantees with the need for low computational overhead in network routers. The main contribution of this dissertation is the definition and realization of a new service, called Quantitative Assured Forwarding, which can offer absolute and relative differentiation of loss, service rates, and packet delays to classes of traffic. We devise and analyze mechanisms that implement the proposed service, and demonstrate the effectiveness of the approach through analysis, simulation and measurement experiments in a testbed network.

To enable the new service, we introduce a set of new traffic control algorithms for network routers. The main mechanism proposed in this dissertation uses a novel technique that performs active buffer management (through dropping of traffic) and rate allocation (for scheduling) in a single step. This is different from prior work which views dropping and scheduling as orthogonal tasks. We propose several solutions for rate allocation and buffer management, through solutions to an optimization problem, approximations of such a solution, and through a closed-loop control theoretical approach. Measurement results from a testbed of PC-routers on an Ethernet network indicate that our proposed service architecture is suitable for networks with high data rates.

We extend the service guarantees of Quantitative Assured Forwarding to TCP traffic by integrating our buffer management and rate allocation algorithms with the feedback capabilities of TCP, and regulate the sending rate of TCP traffic sources at the microflow level. The presented techniques show, for the first time, that it is feasible to give service guarantees to TCP traffic flows, without per-flow reservations in the network.

Acknowledgments

First, I would like to thank my advisor, Jörg Liebeherr, for his continued support over the last five years. Jörg has not only been a great advisor, who fostered my creativity, and helped me stay focused even in difficult times, but he has also been an excellent and patient teacher. His clarity of thought and his insights are evidenced throughout this dissertation, and I can only wish that, in the years to come, I will manage to be as inspirational to others as he has been to me.

In 2002-2003, I spent a year at Nortel Networks, working under the guidance of Victor Firoiu. This internship gave me some perspective regarding the research challenges one can face in industry, and I will keep fond memories of the year I spent in Massachusetts. I hope that I will have more opportunities to collaborate with Victor in the future.

I also would like to thank the members of my thesis committee, Jack Stankovic, Tarek Abdelzaher, Alf Weaver, and Steve Patek, for the feedback they provided regarding my work and their continued interest in my progress. In particular, Tarek helped tremendously in the design of the feedback-based algorithm of Chapter 5, which is a central component of this dissertation.

I deeply enjoyed the stimulating discussions I had with my fellow members in the Multimedia Networks Group over the years, and want to thank all past and present members for their help and support. I especially owe a lot of gratitude to Jianping Wang for everything he has done for me over the years.

Being a foreign student can be a challenge, especially in the first few months, when one has to adjust to a different culture and a new language. My thanks go to all the friends I made here, and who immediately made me feel at home. In particular, thanks to Richard Ruble for helping me

having an easy transition when I first came here, and to my classmates at the time, notably Brian White and Mike Lack, for the much needed comic relief when we were swamped with coursework and research work.

Finally, I am forever in debt to my family for their love and support in every moment in my life. Without them, this dissertation would have certainly never been completed. The moral support of my parents, Pierre and Christiane, my sister, Carole, and her husband, Stephan, as well as my two nephews, Andreas and Sven, has always been the single most important thing in my life. I also want to thank my brother Olivier, who has been helping in his very own way. This dissertation is dedicated to all of them.

Contents

List of Figures

List of Tables

List of Symbols

A_i	Class-*i* arrival curve
$a_i(t)$	Class-*i* arrivals at time *t*
B	Maximum buffer size of the transmission queue
B_i	Class-*i* backlog
$\widetilde{B}_{i,s}(t)$	Prediction made at time *s* of the class-*i* backlog at time *t* $(t > s)$
C	Output link capacity
$D_i(t)$	Delay of the class-*i* packet in transmission at time *t*
$\widetilde{D}_{i,s}(t)$	Prediction made at time *s* of the class-*i* delay at time *t* $(t > s)$
$\overline{D}_{i,s}$	Class-*i* average predicted delay, averaged over the horizon $\widetilde{T}_{i,s}$
d_i	Class-*i* delay bound
e_i	Class-*i* delay differentiation error
e'_i	Class-*i* loss differentiation error
F	Objective function in the optimization problem
g_k	*k*-th equality constraint in the optimization problem
h_k^i	*k*-th inequality constraint applied to class *i* in the optimization problem
K	Proportional control
k_i	Proportional delay differentiation guarantee between classes *i* and $(i+1)$
k'_i	Proportional loss differentiation guarantee between classes *i* and $(i+1)$

L_i	Class-i loss rate bound
$l_i(t)$	Class-i losses at time t
MSS^j	Maximum segment size of TCP flow j
p_i	Class-i loss rate (averaged over current busy period)
Q	Number of different classes of traffic
R_i^{in}	Class-i input curve
$R_i^{in,j}$	Input curve of flow j in class i
$\widetilde{R}_{i,s}^{in}(t)$	Prediction made at time s of the class-i input curve at time t $(t > s)$
$\widetilde{R}_{i,s}^{in,j}(t)$	Prediction made at time s of the input curve at time t $(t > s)$ of the flow j in class i
R_i^{out}	Class-i output curve
$\widetilde{R}_{i,s}^{out}(t)$	Prediction made at time s of the class-i output curve at time t $(t > s)$
RTT^j	Round-trip time of flow j
$\widehat{RTT}^j$	Estimated round-trip time of flow j
r_i	Class-i service rate
$r_i^{\min}(t)$	Minimum class-i service rate needed to meet service guarantees at time t
$\widetilde{r}_{i,s}^{\min}(t)$	Prediction at time s of the minimum class-i service rate needed to meet service guarantees at time t
$s^j(t)$	Start time of the TCP round in progress at time t for flow j
$\widehat{s}^j(t)$	Estimated start time of the TCP round in progress at time t for flow j
$\widetilde{T}_{i,t}$	Predicted horizon for class i at time t
W^j	TCP window size of flow j
$\widehat{W}^j$	Estimated TCP window size of flow j
$\widetilde{W}_s^j(t)$	Prediction made at time s of the TCP window size of flow j at time t $(t > s)$
$\mathcal{X}$	Number of recorded flows
$Xmit_i$	Measured class-i transmissions
$\mathbf{x}_t$	Optimization vector at time t in the optimization problem

$\lambda_i(t)$	Instantaneous class-*i* arrival rate at time *t*
μ_i	Class-*i* throughput guarantee
$\xi_i(t)$	Instantaneous class-*i* drop rate at time *t*
$\bar{\Omega}$	Mean TCP window size of the recorded flows
σ	Multi-stage filter sampling interval
θ	Multi-stage filter byte threshold
τ^j	Estimated remaining time before the start of the next round for flow *j*
ζ	Correction factor to account for flow filtering in the prediction of the input curve

List of Acronyms

ABE	Alternative Best-Effort
ADC	Absolute Delay Constraint
ADD	Average Drop-Distance
AF	Assured Forwarding
ALC	Absolute Loss Constraint
ALTQ	Alternate Queueing
ARC	Absolute Rate Constraint
ATM	Asynchronous Transfer Mode
AVQ	Adaptive Virtual Queue
BPR	Backlog-Proportional Rate
BSD	Berkeley Software Distribution
CBP	Complete Buffer Partitioning
CBQ	Class-Based Queueing
CCF	Critical Cells First
C-DBP	Class Distance-Based Priority
CE	Congestion Experienced
CSFQ	Core-Stateless Fair Queueing
DiffServ	Differentiated Services

DPS Dynamic Packet State

DSCP Differentiated Services Code Point

DWFQ Dynamic Weighted Fair Queueing

ECN Early Congestion Notification

EF Expedited Forwarding

FIFO First In First Out

FOQ Feedback Output Queueing

FPU Floating Point Unit

FRED Flow-RED

FTP File Transfer Protocol

GPS Generalized Processor Sharing

HFSC Hierarchical Fair Service Curves

HPD Hybrid Proportional Delay

HTTP HyperText Transfer Protocol

IETF Internet Engineering Task Force

IntServ Integrated Services

IP Internet Protocol

ISP Internet Service Provider

JoBS Joint Buffer Management and Scheduling

LPF Lower Priority First

MDP Mean Delay Proportional

MPLS Multi-Protocol Label Switching

PBS Partial Buffer Sharing

PCC Processor Cycle Counter

PHB Per-Hop Behavior

PI	Proportional-Integral dropper
PLR	Proportional Loss Rate
PQCM	Proportional Queue Control Mechanism
QAF	Quantitative Assured Forwarding
QoS	Quality-of-Service
RDC	Relative Delay Constraint
RED	Random Early Detection
REM	Random Early (or Exponential) Marking
RIO	Random Early Detection with In and Out profiles
RLC	Relative Loss Constraint
RSVP	Resource Reservation Protocol
RTT	Round-Trip Time
SACK	Selective Acknowledgments
SCORE	Scalable (or Stateless) Core
SCTP	Stream Control Transmission Protocol
TCP	Transport (or Transmission) Control Protocol
TOS	Type of Service
TSC	Timestamp Counter
UDP	User Datagram Protocol
VPN	Virtual Private Network
WRED	Weighted Random Early Detection
WFQ	Weighted Fair Queueing
WTP	Waiting Time Priority

Chapter 1

Introduction

Since its creation in the early 1970s, the Internet has adopted a "best-effort" service, which relies on the following three principles: (1) No traffic is denied admission to the network, (2) all traffic is treated in the same manner, and (3) the only guarantee given by the network is that traffic will be transmitted in the best possible manner given the available resources, that is, no artificial delays will be generated, and no unnecessary losses will occur.

The best-effort service is adequate as long as the applications using the network are not sensitive to variations in losses and delays (e.g., electronic mail), the load on the network is small, and if pricing by network providers is not service-based. These conditions held in the early days of the Internet, when the Internet merely consisted of network connections between a handful of universities.

However, since the late 1980s, these conditions do not hold anymore, for two main reasons. First, an increasing number of different applications, such as real-time video [80], peer-to-peer networking (e.g., *napster* [7], *Gnutella* [8]), or the World-Wide Web [17], to name a few, have been using the Internet, as illustrated by several measurement studies, e.g., [58,123,130]. These different applications have different needs in the service they must receive from the network. Second, the Internet has switched from a government-supported research network to a commercial entity in 1994, thereby creating a need for service-based pricing schemes that can better recover cost and maximize revenue than a best-effort network [132]. These two factors have created a demand for different levels of service. In some ways, the Internet has been victim of its success, and finding a

solution to the problem of providing different levels of services in the network has become critical to ensure the long-term survival of the Internet.

Traffic control mechanisms to differentiate performance based on network-operator or application requirements are referred to as *Quality-of-Service* (QoS). Some have argued that increasing the capacity of the backbone network makes QoS obsolete [70]. Indeed, as reported by measurement studies of backbone links [149], the core of the Internet is currently over-provisioned and supports low latency and low loss service for almost all of its traffic [118]. On the other hand, increasing the capacity of the Internet backbone has merely shifted the capacity bottleneck to the edge of the backbone networks, and the service experienced by demanding applications remains inadequate. As a result, mechanisms for service differentiation are urgently needed in the access networks that connect end-users to the Internet backbone.

In fact, the explosion of link capacity in the network, instead of alleviating the need for service guarantees, has put more stringent requirements on QoS architectures. Routers at the edges of the Internet backbone now have to serve millions of concurrent flows at gigabit per second rates, which induces scalability requirements. First, the state information kept in the routers for providing QoS must be small. Second, the processing time for classifying and scheduling packets according to their QoS guarantees must be small as well, even with the advent of faster hardware. In addition to these two scalability requirements, the fact that the Internet is now mostly a commercial network requires to utilize the existing network resources as efficiently as possible, for instance, maximizing the utilization of the links.

A number of QoS architectures have been proposed to address the above requirements. QoS architectures can be distinguished according to two criteria. The first criterion is whether guarantees are expressed for individual traffic flows (*per-flow guarantees*), or for groups of flows with the same service requirements (*per-class guarantees*). Per-flow guarantees generally require to perform per-flow resource reservations in routers. That is, each flow has to reserve resources at all routers from source to destination before starting to send data. When data is transmitted, each incoming packet has to be inspected at each router to determine to which flow the packet belongs. Then, the packet is mapped to the per-flow reservations in the router. These two operations constitute

what is called per-flow classification. In a per-flow architecture, the classification overhead grows linearly with the number of flows present in the network. Per-class guarantees usually do not rely on reservations. Here, flows are grouped in classes of traffic. Each packet entering the network is marked with the class of traffic to which it belongs. Routers in the network classify and transmit packets according to the service guarantees offered to classes of traffic. Since there are only a few classes of traffic in the network, the overhead incurred with per-class guarantees is smaller than that of per-flow guarantees. As a disadvantage, per-class service guarantees do not immediately translate into per-flow guarantees.

The second criterion to distinguish service architectures is whether guarantees are expressed with reference to guarantees given to other flows or classes (*relative guarantees*), or if guarantees are expressed as absolute bounds (*absolute guarantees*). As an example, absolute guarantees are of the form "Class-2 Delay ≤ 5 *ms*", or "Flow-2 Throughput ≥ 3 Mbps". Such absolute bounds define strong service guarantees. Relative service guarantees are weaker than absolute guarantees, and can be further discriminated between *qualitative guarantees* and *proportional guarantees*. Qualitative guarantees impose an ordering between classes of traffic without quantifying the differentiation, as in

$$\text{Class-2 Delay} \leq \text{Class-1 Delay} .$$

Proportional guarantees quantify the differentiation between classes of traffic by ensuring the ratios of the QoS metrics of two classes is roughly constant, and held equal to a proportional differentiation factor. For two priority classes, proportional service differentiation could specify that the delays of packets from the higher-priority class be half of the delays from the lower-priority class, e.g.,

$$\frac{\text{Class-2 Delay}}{\text{Class-1 Delay}} \approx 2 ,$$

but without specifying an upper bound on the delays. Likewise, loss differentiation is defined in terms of ratios of loss rates, such as

$$\frac{\text{Class-2 Loss Rate}}{\text{Class-1 Loss Rate}} \approx 5 .$$

The fundamental contribution of this dissertation is to explore the limits on the strength of service differentiation that can be obtained by per-class QoS. To that effect, we consider the design and implementation of a per-hop service architecture that provides absolute and proportional service guarantees to classes of traffic, while avoiding resource reservations [33].

The remainder of this chapter motivates our research and is structured as follows. In Section 1.1, we describe proposals for service architectures in packet networks that have tried, over the past decade, to provide a solution to the service differentiation problem. These proposals seem to imply the existence of a trade-off between strength of service differentiation and complexity of the service architecture. In Section 1.2, we present our thesis statement and describe the contributions of this dissertation. In Section 1.3, we give an overview of the service architecture we propose as a solution to the service differentiation problem, by introducing the different components of our service architecture. We outline the structure of the dissertation in Section 1.4.

1.1 History of Internet QoS

The need for service differentiation and QoS for the Internet became a topic of interest in the late 1980s and early 1990s, with the advent of networked multimedia applications (e.g., [84, 85, 86, 150]). The first solution for service differentiation in packet-switched networks was the Tenet protocol suite (see for instance [63, 64, 156]), developed by the Tenet group at UC Berkeley. At approximately the same time, Clark, Shenker and Zhang proposed a service architecture designed to provide QoS to real-time applications, and introduced the mechanisms associated with their proposed service model in [38].

1.1.1 Integrated Services

Building on the initial work by the Tenet group and the work by Clark, Shenker and Zhang, the IETF proposed the Integrated Services (*IntServ*) architecture [23] as a QoS architecture for IP networks. IntServ, developed in the early and mid-1990s, provides the ability to give individual flows absolute QoS guarantees on end-to-end packet delays (delay bounds) [133], and packet losses (no loss) [153],

as long as the traffic of each flow conforms to a pre-specified set of parameters, e.g., peak sending rate, or maximum burst size [134]. This type of per-flow, absolute service guarantees is particularly appropriate for applications that cannot tolerate or adapt to a lower level of performance than they require.

The IntServ architecture requires per-flow classification in routers. Additionally, IntServ implementations require packet scheduling primitives, e.g., [16, 119], which run a dynamic priority sorting algorithm. The scheduling overhead can become significant when the routers have to process a large number of packets within a short period of time. Also, the IntServ architecture relies on a signaling protocol (e.g., RSVP, [24]) for reserving network resources, and on admission control for determining which flows can be admitted with the assurance that no service violation will occur. Both of these mechanisms require that each router keep per-flow state information. Furthermore, it has been shown that using admission control mechanisms such as peak-rate allocation could result in under-utilizing the network resources [152]. Last, because resource reservations must be updated at routers every time a new flow with service guarantees enters the network, the communication overhead associated with the signaling mechanisms cannot be neglected.

The open issues outlined above have prevented the IntServ architecture from being widely deployed so far, despite the strength of the proposed service guarantees.

1.1.2 Differentiated Services

Taking a step back from the IntServ approach, the interest in Internet QoS shifted in the late 1990s to architectures that make a distinction between operations performed in the network core, and operations performed at the edges of the network. The basic idea is that the amount of traffic in the network core does not permit complex QoS mechanisms, and that most of the QoS mechanisms should be executed at the network edge, where the volume of traffic is smaller.

These recent efforts resulted in the Differentiated Services (*DiffServ*) architecture [18], which bundles flows with similar QoS requirements in classes of traffic. The mapping from individual flows to classes of traffic is determined at the edges of the network, by marking packet headers. In the network core, scheduling primitives only work with a few classes of traffic, and can thus remain

relatively simple. DiffServ currently offers two different types of service in addition to Best-Effort: an Assured Forwarding (AF, [75]) service and an Expedited Forwarding (EF, [43]) service.

Assured Forwarding provides isolation between different classes of traffic. In each traffic class, packets are marked to belong to one of three drop precedence levels. AF offers qualitative loss differentiation between the drop precedence levels of each class, by dropping packets in times of congestion with a probability function of their drop precedence. Assured Forwarding does not require per-flow classification or signaling, but the qualitative guarantees offered provide weaker service assurance than absolute guarantees.

Expedited Forwarding offers absolute service guarantees on delay variations to flows. In essence, providing the EF service to a flow is equivalent to providing a virtual leased-line to this flow, and involves per-flow peak-rate allocation. Because per-flow peak-rate allocation under-utilizes the network resources, Expedited Forwarding can be offered only to a limited amount of traffic. For instance, [142] shows that, to achieve delay bounds in the order of 240 *ms* at a given link, the total amount of EF traffic should not exceed more than 10% of the total capacity of the link.

1.1.3 Design Space of Service Architectures

The IntServ and DiffServ architectures indicate a trade-off between simplicity of the implementation and strength of service guarantees, which we illustrate in Figure 1.1. In Figure 1.1, we plot the complexity of a few service architectures against the strength of service guarantees they offer. On the one hand, IntServ and Expedited Forwarding provide strong, absolute service guarantees, but require per-flow mechanisms. On the other hand, per-class architectures such as Assured Forwarding only support qualitative QoS guarantees.

Recently, researchers have explored the design space described in Figure 1.1, in search of an ideal service with strong service differentiation and low complexity. For instance, the *Proportional Differentiated Services* architecture of [47] offers a stronger class-based service architecture than Assured Forwarding, by providing proportional service guarantees to delays and losses. A very significant advance in devising a service with relatively low overhead and absolute guarantees is

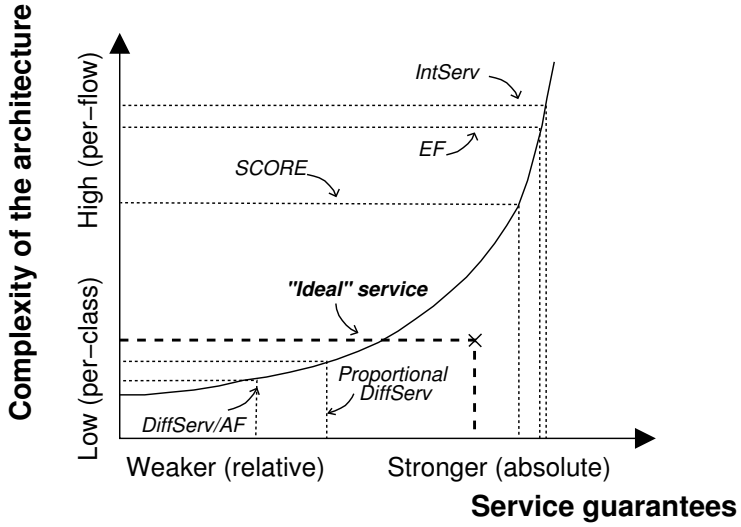


Figure 1.1: **The trade-off between strength of service guarantees and complexity of the implementation.** IntServ and Expedited Forwarding provide very strong service guarantees at the cost of per-flow complexity, while Assured Forwarding only provides limited service assurance, but has low complexity. Ideally, a service should be able to provide strong service differentiation with low complexity. Note that the picture is qualitative.

the Scalable-Core (SCORE, [143]) architecture proposed by Stoica and Zhang, and architectures derived from it [26, 39, 91, 113]. SCORE tries to keep the strength of the IntServ guarantees without resorting to per-flow operations, using a technique called Dynamic Packet State (DPS). DPS puts the state information needed to provide IntServ-like service guarantees in IP packets headers, thereby alleviating the need for maintaining per-flow state information in routers. The algorithm central to the SCORE architecture, called Core Stateless Fair Queueing (CSFQ, [141]) uses DPS to provide end-to-end delay guarantees to flows without requiring per-flow state information at network routers. The basic idea for meeting end-to-end delay requirements is to keep track of the delays experienced by packets along the path from the source to the destination, by storing the values of the experienced delays in the packet headers. The stored information is used for adjusting the priority of packets so that end-to-end requirements are met. The SCORE architecture does not require any per-flow information be maintained in the network core, but relies on packet classification at network boundaries, for instance, interconnects between two ISP's. Mechanisms to alleviate

per-flow classification at inter-network boundaries in SCORE have been recently proposed in [145]. Per-flow classification remains necessary at the network edge.

1.2 Thesis Statement and Contributions

Our thesis research advances the understanding of the limits on the strength of service differentiation that can be provided by class-based architectures for the Internet, without resource reservations. Our thesis statement is as follows:

> *The scope of class-based service guarantees can be significantly enhanced by using appropriate buffer management, scheduling, and the feedback capabilities of the network.*

The goal of this dissertation is to present a new point in the design space described in Figure 1.1, by devising the strongest possible class-based service without reservations. To achieve this goal, we have revisited the tenets of Internet QoS.

Router mechanisms to support service differentiation include scheduling and buffer management. Scheduling determines the order of transmission of packets leaving the router, while buffer management controls which packets enter the router. Until very recently, scheduling and buffer management were handled separately, even though both mechanisms address the issue of managing a transmission queue at a given router. The only difference between the two mechanisms lies in the fact that scheduling manages the head of the transmission queue, deciding which packet will leave the queue next, while buffer management manages the end of the transmission queue, deciding if new packets can be admitted to the queue.

The first contribution of this dissertation is to show that considering buffer management and scheduling in a single step allows for significantly enhancing the service guarantees that class-based architectures can provide, without resorting to resource reservation. We present a scheme based on an adaptive service rate allocation, conditioned by the instantaneous backlog of traffic classes, the service guarantees, and the availability of the resources. Packet scheduling immediately follows from the rate allocation.

The second contribution of this dissertation is to show that a practical algorithm based on feedback control theory to allocate service rates and drop traffic can enforce the desired service guarantees.

The third contribution of this dissertation is to demonstrate that the proposed service architecture can be realized at relatively high speeds. To that effect, we describe our reference implementation in PC-routers of the algorithms we propose, and present measurement experiments obtained from a testbed network.

Mechanisms for providing QoS guarantees have to work in concert with end-to-end mechanisms, such as TCP feedback mechanisms for congestion avoidance and control [10, 82]. However, to the best of our knowledge, with the exception of RIO [37], which builds on the RED algorithm [68] in an effort to reduce packet drops, none of the algorithms used in the proposed service architectures takes into account of the feedback capabilities of TCP traffic. Traffic regulation is always realized by admission control mechanisms or traffic policers, which are separate from the scheduling and dropping mechanisms.

The fourth contribution of this dissertation is to demonstrate that one can extend a service architecture to take into account the particularities of TCP traffic. In particular, we show that exploiting TCP feedback mechanisms to regulate the traffic arrivals by dropping or marking traffic "smartly" is a viable alternative to admission control, signaling or policing for service differentiation.

1.3 Overview of the Proposed Service Architecture

We illustrate how our proposed service architecture is deployed in a network in Figure 1.2. In Figure 1.2, traffic is sent from a source host to a destination host. The source host is connected to the backbone via a router, which supports local, per-class service guarantees. Likewise, a router connects the destination host to the backbone. The backbone consists of a number of routers. In the example of Figure 1.2, only the two routers connecting the hosts to the backbone provide service differentiation. Based on the service guarantees and the available resources, both routers dynamically allocate service rates to traffic classes. Packet scheduling at both routers directly follows from

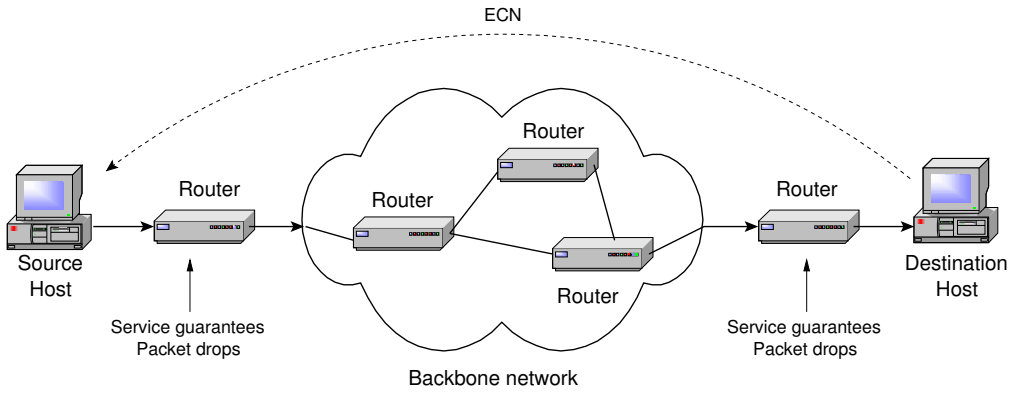


Figure 1.2: **Illustration of the deployment of the proposed service in a network.** Routers are in charge of transmitting and dropping packets according to the available resources and the QoS desired. Routers set the regulation signals (ECN), which are used by the end-hosts to regulate their traffic.

the service rate allocation. The volume of traffic in the network is controlled by discarding traffic at both routers, and by sending feedback from the destinations to the traffic sources to reduce the volume of traffic. There is no communication (i.e., signaling) between the different routers, the rate allocation is independent at each router, and the service guarantees provided are also independent at each router. The service architecture can be incrementally deployed, in the sense that each router that supports the proposed service improves the QoS observed in the entire network. The example of Figure 1.2 assumes that QoS is only needed at access links. However, we emphasize that the service can also be implemented in routers in the network core. We next discuss in more details the service guarantees, packet scheduling and dropping, and traffic regulation.

1.3.1 Service Guarantees

The service we propose consists of per-hop, per-class guarantees, on delay, losses, and throughput of traffic. These guarantees do not immediately translate into end-to-end service guarantees. However, a per-hop, per-class service architecture can be used to build end-to-end service guarantees, for instance if the end applications are in charge of dynamically selecting which class of traffic they require [49].

Our goal is to provide a set of service guarantees that can encompass all of AF, Proportional

Differentiated Services, and other class-based services without reservations. More generally, we want to be able to enforce *any* mix of absolute and proportional guarantees at each participating router. The service guarantees are independent at each participating router. We refer to this service as "Quantitative Assured Forwarding" service (QAF, [34]). Absolute guarantees apply to loss rates, delays, or throughput, and define a lower bound on the service received by each class. Proportional guarantees apply to loss rates and queueing delays, and can be used to differentiate average-case performance. As an example of the service guarantees of Quantitative Assured Forwarding for three classes of traffic, one could specify service guarantees of the form

- Class-1 Delay ≤ 2 *ms*,
- Class-2 Delay $\approx 4 \cdot$Class-1 Delay,
- Class-2 Loss Rate $\leq 1\%$,
- Class-3 Loss Rate $\approx 2 \cdot$Class-2 Loss Rate, and
- Class-3 Service Rate ≥ 1 Mbps

at a given router, and other values at another router. The QAF service does not require resource reservations or signaling, and can be realized without communication between different routers. As a per-hop service, Quantitative Assured Forwarding, used in conjunction with routing mechanisms that can perform route-pinning, can be used to infer end-to-end service differentiation, and can be used to select the most appropriate route for a particular application given the service demands.

Note that, contrary to the AF service, which provides three levels of drop precedence within a class of traffic, Quantitative Assured Forwarding offers a single drop level per class. However, it can be shown that Quantitative Assured Forwarding can be used to emulate the AF service, by assigning each AF drop precedence level to a separate QAF class. Since the QAF service supports absolute guarantees on delays, QAF can also be used to emulate the delay guarantees offered by the EF service. Therefore, our proposed service model can implement and inter-operate with DiffServ networks, with the possible addition of remarking primitives at the boundaries between DiffServ and QAF domains in charge of mapping the different AF drop levels to different QAF classes.

1.3.2 Scheduling and Dropping

The desired service guarantees are realized independently at each router by scheduling and dropping algorithms. Scheduling is based on a service rate allocation to classes of traffic, which share a common buffer. The rate allocation adapts to the traffic demand from different classes. The rates are set so that the per-hop service guarantees are met. If this is not feasible, traffic is dropped. In practice, rate allocation and buffer management are combined in a single algorithm, which recomputes the service rate allocation to classes of traffic at the same time it makes dropping decisions. The service rate allocation is independent at each router, and there is no coordination among different routers.

1.3.3 Regulating Traffic Arrivals

A mechanism has to be in charge of controlling the amount of traffic that enters the network, to ensure that service guarantees can be met. Traditional approaches to QoS use a combination of admission control and per-flow traffic policing. These approaches require to keep per-flow information, which we want to avoid in our architecture. Furthermore, they do not consider the salient feature of TCP traffic, which is to reduce the sending rate when losses occur. Hence, we do not use admission control and policing, but instead, we regulate the amount of traffic that enters the network by dropping traffic at routers and by relying on the congestion control algorithms of TCP.

1.4 Structure of the Dissertation

The remainder of this dissertation presents the details of each of the three components of our service architecture, the service guarantees, the scheduling and buffer management algorithms, and our approach to controlling traffic. The remainder of this dissertation is organized as follows. In Chapter 2, we review previous work. We focus on the different class-based services that have been recently proposed, and discuss the mechanisms required to implement them.

In Chapter 3, we express the provisioning of per-class QoS within a formal framework that inspired by Cruz's network calculus [41, 42]. We define the metrics we use to quantify the level of

service received by classes of traffic, and we offer a formal definition of the set of service guarantees supported by our service architecture.

In Chapter 4, we express the problem of providing Quantitative Assured Forwarding service guarantees as an optimization problem. We show that, assuming infinite computational power, one can design a reference algorithm which dynamically allocates service rates and drop packets according to the solution to a non-linear optimization problem. We discuss the optimization function and the constraints of the optimization problem. We provide numerical simulation examples to illustrate the effectiveness of the approach with respect to service differentiation, and to compare our reference algorithm to existing methods for loss and delay differentiation. We also provide a heuristic approximation of the optimization problem.

While the performance of the reference algorithm with respect to satisfying the service guarantees is excellent, its computational overhead prohibits its implementation in network routers. Thus, we propose in Chapter 5 a closed-loop control algorithm to approximate the reference algorithm. We apply linear feedback control theory for the design of the closed-loop control, and, to this effect, make assumptions to circumvent the non-linearities in the system of study. To illustrate the validity of the assumptions, we use simulation results to show that the closed-loop algorithm and the optimization algorithm have comparable performance.

In Chapter 6, we describe the implementation of our service architecture in PC-routers using the BSD family of operating systems [32]. We present measurement results obtained from a testbed of PC-routers to show that the implementation can realize the desired service guarantees in links with speeds in the order of a few hundred megabits-per-second on a 1 GHz PC-router. We point out that the implementation is being disseminated as part of the popular KAME [3] and ALTQ-3.1 [30] networking extensions to the BSD kernels.

In Chapter 7, we extend our service architecture to TCP traffic. Assuming at first that infinite computational power is available, we present a per-flow reference algorithm which exploits TCP feedback mechanisms for the purpose of avoiding packet losses and regulating traffic. We then discuss a set of approximations to this reference algorithm for implementation purposes. We use multi-stage filters to avoid per-flow management and devise an efficient heuristic approximation.

We present our conclusions and summarize the contributions of this dissertation in Chapter 8. We also outline future research directions.

Chapter 2

Previous Work

The past decade has seen numerous proposals for service architectures, e.g., [14,18,23,38,47,73,78, 100, 113, 143]. Not all of the proposed service architectures directly relate to the work presented in this dissertation. For instance, deployment of per-flow services such as the Tenet protocol suite [14], or the Integrated Services architecture [23] discussed in the introduction is currently not actively pursued.

The research community seems to have reached a consensus that per-class architectures will be a viable solution for providing service guarantees in the Internet, because class-based architectures have the advantage that they work with simpler algorithms for enforcing QoS guarantees than per-flow architectures, and can be deployed with only minor changes to the network architecture.

The discussion in this chapter focuses on recently proposed class-based service architectures, and the mechanisms required to implement them. The remainder of this chapter is organized as follows. In Section 2.1, we discuss in greater detail the Differentiated Services architecture we briefly introduced in Chapter 1. Then, in Section 2.2, we discuss the Proportional Differentiated Services architecture from [47] which has been the starting point of our work. Last, in Section 2.3, we discuss other class-based services that have been recently proposed to improve on either the Best Effort model, or the Differentiated Services architecture.

2.1 Differentiated Services

The Differentiated Services architecture (DiffServ, [18]) is the class-based service architecture proposed by the Internet Engineering Task Force (IETF) for service differentiation on the Internet. DiffServ relies on three fundamental ideas.

First, DiffServ uses flow aggregation to avoid per-flow operations in the core of the network. In DiffServ terminology, individual flows, or *microflows*, are bundled in *macroflows* with similar service requirements. Service guarantees are only provided to macroflows. To that effect, macroflows use different classes of service, called Per-Hop Behavior (PHB). The aggregation of microflows in macroflows requires per-flow classification [137], which is performed at the edge of the network, where computational resources are less scarce than in the core. At the edge, in each packet, the DiffServ CodePoint (DSCP, [114]) of the IP header is marked with a value denoting which class of traffic the packet belongs to. The notion of "edge" is not precisely defined in DiffServ, but one can envision two possibilities. The edge can be the host-network interface at an individual host, in which case, per-flow classification is performed by the host operating system or applications, as in [46,49]. Alternatively, the edge can be the router that connects a local, microflow-aware network, to the rest of the Internet. A router connecting a microflow-aware network to the rest of the Internet is typically called an access (or edge) router.

Second, the DiffServ architecture only provides local, per-hop differentiation at routers, which motivates the name of Per Hop Behavior (PHB) for classes of service. Providing per-hop differentiation has the advantage of eliminating the need for communication between different routers in the network. A second advantage is that service differentiation can only be deployed at points of congestion, without requiring deployment in the rest of the network. As an illustration, [46] gives the example of a network operator, who can over-provision most of its network, thereby alleviating the need for service differentiation, and only deploy DiffServ at transoceanic links where link capacity becomes more expensive, and congestion can occur.

Third, there is no signaling in the DiffServ architecture. Even though some proposals for end-to-end service differentiation in DiffServ, such as the Virtual Wire per-domain behavior [88], originally

called the Virtual Leased Line service [87, 115], require to reserve some resources, the reservation is handled by a centralized agent, called a bandwidth broker [115].

In addition to supporting the traditional best-effort service, the Differentiated Services architecture supports two per-hop behaviors: Expedited Forwarding, and Assured Forwarding.

2.1.1 Expedited Forwarding

The Expedited Forwarding (EF) PHB was initially proposed by Jacobson et al. in 1997 [115], and was refined in [87], as a service that provides a guaranteed peak rate service with negligible queueing delays or losses. While EF only provides local, per-hop guarantees, the objective is to use EF as a building block for network-wide services such as the Virtual Wire [88] per-domain behavior. The goal of the Virtual Wire service is to provide each EF macroflow with a service equivalent to a virtual leased line, or a virtual circuit in ATM networks.

The authors of [87] envision that EF requires shaping at the network edge, so that EF traffic does not enter the network at a rate exceeding a peak rate R. A capacity of R is reserved in the entire network for EF traffic, so that EF macroflows do not experience delay or losses. The bandwidth reservation R is statically configured in a bandwidth broker. The bandwidth broker is a centralized agent configured with a set of policies, which determine the level of service different classes should receive. The bandwidth broker keeps track of the current allocation of traffic to different classes, and handles new requests to mark new traffic subject to the configured policies and current allocation. Routers in turn query the bandwidth broker to determine how much link capacity shall be reserved for EF traffic.

Subsequent research led by Charny, Le Boudec and others [15, 29, 22] showed that even with peak rate allocation for EF macroflows, an EF service cannot be guaranteed negligible losses and delays. Indeed, multiplexing EF traffic from several input ports in routers can result in bursty traffic, which, in turn, may cause delay and losses. This finding led to a change to the original definition of the Expedited Forwarding PHB [43]. Instead of guaranteeing no losses and negligible delays, the authors of [43] propose to guarantee bounded delay variations to EF macroflows. More formally, each EF packet arriving at a router obtains a a delay guarantee $D < F + E$, where F is a target delay

guarantee, and E is an error.

2.1.2 Assured Forwarding

The Assured Forwarding service is based on a proposal that was originally called the "Allocated Capacity framework", introduced by Clark and Fang in [37]. In the Allocated Capacity framework, a class of traffic is provided with a certain bandwidth profile, defined by a rate R. As long as the aggregate amount of traffic from that class has a rate lower than R, traffic is marked as *in-profile*; otherwise it is marked as *out-of-profile*. In times of congestion, out-of-profile traffic is dropped more aggressively than in-profile traffic. In other words, a class is allowed to exceed its profile R when there is no congestion and the network load is low, but is restricted to sending traffic within its profile when the network is congested. The rate R is statically reserved, or provisioned, at network design time.

The AF service of the DiffServ architecture supports qualitative guarantees, but no classes are provided absolute service guarantees, and the difference in the service received by different classes is not quantified. While some have argued that Assured Forwarding provides absolute differentiation, because the profile R can be viewed as a throughput guarantee, we point out that in-profile traffic is not guaranteed a lossless service. Hence, traffic sending at a rate $R' < R$, thereby remaining in-profile, can still experience traffic losses, and obtain a service rate $R'' < R' < R$, which contradicts the notion that R is a throughput guarantee. The absence of throughput guarantee is clearly exhibited in the case of TCP traffic, as discussed in [154]: regardless of how well provisioned the network is, it may be impossible to provide throughput guarantees to TCP flows with the AF service. In fact, the only assurance that in-profile traffic gets is that, should congestion occur, it will not be dropped as aggressively as out-of-profile traffic. In other words, AF only provides isolation between different AF classes, and qualitative loss differentiation between the drop precedence levels within each class. We refer to the discussion in [65] to summarize concerns raised about the actual differentiation offered between different classes of traffic.

2.1.3 Mechanisms

The DiffServ, AF and EF specifications given in [18, 75, 43] do not impose a particular scheduling or buffer management algorithm. EF can for instance be implemented using well-known fixed-priority scheduling algorithms [115], or rate-based scheduling algorithms, e.g., Class-Based Queueing (CBQ, [69]).

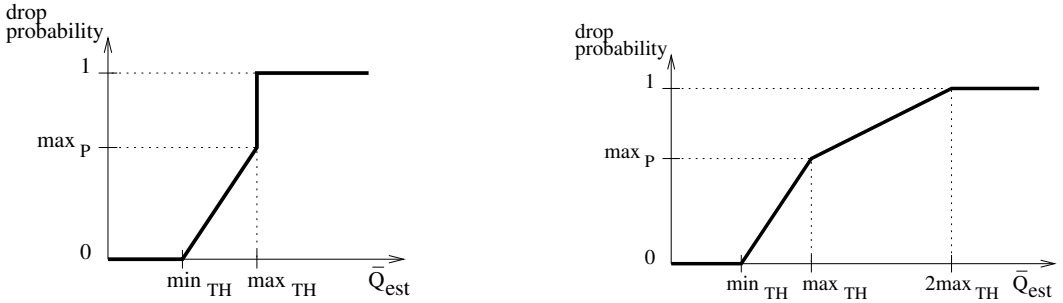
While EF can be realized through appropriate scheduling algorithms, the Assured Forwarding service, on the other hand, can be enforced with buffer management algorithms. Indeed, as long as the network is correctly provisioned, i.e., enough link capacity has been reserved in advance for each class of traffic, scheduling in Assured Forwarding can be realized with a first-in-first-out (FIFO) discipline. Service differentiation can be enforced by marking packets as in-profile or out-of-profile, and using a buffer management algorithm that drops out-of-profile packets more aggressively.

The literature regarding buffer management algorithms, also called active queue management algorithms, is rich, and we present here a brief summary of the proposed buffer management algorithms, that can be used or extended to provide qualitative loss differentiation, as in the Assured Forwarding PHB.

The key mechanisms of a buffer management algorithm are the *backlog controller*, which specifies the time instances when traffic should be dropped, and the *dropper*, which specifies the traffic to be dropped. We refer to a recent survey article [97] for an extensive discussion of buffer management algorithms.

Backlog Controllers. Initial proposals for active queue management in IP networks [60, 68] were motivated by the need to improve TCP performance, without considering service differentiation. More recent research efforts [37, 105, 117, 129] enhance these initial proposals in order to provide service differentiation, and can be used to realize the AF service.

Among backlog controllers for IP networks, Random Early Detection (RED, [68]) is probably the best known algorithm. RED was motivated by the goal to improve TCP throughput in highly loaded networks. RED operates by probabilistically dropping traffic arrivals, when the backlog at a node grows large. RED has two threshold parameters for the backlog at a node, denoted as



(a) RED (b) `gentle_` RED

Figure 2.1: **Drop probability in RED.** The probability of dropping a packet is a function of an estimate on the average queue size in RED and `gentle_` RED.

$\min_{TH}$ and $\max_{TH}$. RED estimates the average queue size, $\overline{Q}_{est}$ and compares the estimate to the two thresholds. If $\overline{Q}_{est} < \min_{TH}$, RED does not drop any arrival. If $\overline{Q}_{est} > \max_{TH}$, RED drops all incoming traffic. If $\min_{TH} \leq \overline{Q}_{est} \leq \max_{TH}$, RED will drop an arrival with probability $P(\overline{Q}_{est})$, where $0 \leq P(\overline{Q}_{est}) \leq 1$ is a function which increases linearly in $\overline{Q}_{est}$, and satisfies $P(\max_{TH}) = \max_P$. We illustrate the drop probability function in RED in Figure 2.1(a). The `gentle_` variant of RED has a smoother piecewise-linear drop probability function, as depicted in Figure 2.1(b), and reportedly improves the robustness of RED with respect to parameter setting [67].

Several algorithms that attempt to improve or extend RED have been proposed, e.g., [13, 37, 60, 77, 105, 117, 129, 148]. For example, Blue [60] uses different metrics to characterize the probability of dropping an arrival. Instead of the backlog, Blue uses the current loss ratio and link utilization as input parameters.

RIO, originally proposed to implement the Allocated Capacity framework [37] from which the Assured Forwarding service is derived, WRED [148], and multi-class RED [129] are extensions to RED which aim at class-based service differentiation. All three schemes have different dropping thresholds for different classes, in order to ensure loss differentiation. Note that in an per-flow context, the idea of using different threshold values is pursued for Flow-RED (FRED, [105]), which uses per-flow thresholds. In FRED, flows are discriminated by their source-destination address pairs.

CHOKe [117] tries to provide per-flow loss differentiation without keeping any per-flow state information. The algorithm works as follows. When the queue size exceeds a first threshold value, a packet is drawn at random from the queue. If the incoming packet and the packet drawn from the queue belong to the same flow, both are dropped. If they belong to different flows, and the queue size does not exceed a second threshold, the incoming packet is dropped with a probability linearly dependent on the size of the queue. If the queue size does exceed this second threshold, the incoming packet is dropped.

Random Early Marking (REM, [13]) is close in spirit to the dropping mechanisms of the algorithm we will present in Chapter 4, since it treats the problem of marking (or dropping) arrivals as an optimization problem. The objective is to maximize a utility function subject to the constraint that the output link has a finite capacity. The REM algorithm marks packets with a probability exponentially dependent on the cost of a link. The cost is directly proportional to the buffer occupancy.

REM can also be expressed in terms of a feedback control problem. Based on a closed-loop formulation of TCP throughput in [110], Hollot et al. propose to use a proportional-integral (PI) backlog controller to achieve fast convergence to the desired queue length and to increase robustness of the system [77]. It can be shown that REM and PI are in fact equivalent [155].

For a link of capacity C and buffer size B, the Adaptive Virtual Queue algorithm (AVQ, [96]) maintains a virtual queue of size B, served at a capacity $\widetilde{C} < C$. Packets are marked or dropped when they overflow the virtual queue. The value $\widetilde{C}$ varies over time as a function of the difference between arrival and departures, and relies on the closed-loop formulation of the TCP throughput in [110].

Droppers. The simplest and most widely used dropping scheme is Drop-Tail, which discards arrivals to a full buffer. For a long time, Drop-Tail was thought to be the only dropper implementable in high-speed routers. Recent implementation studies [147] demonstrated that other, more complex, dropping schemes, which discard packets that are already present in the buffer (push-out), are viable design choices even at high data rates.

The simplest push-out technique is called Drop-from-Front [98]. Here, the oldest packet in the

transmission queue is discarded. In comparison to Drop-Tail, Drop-from-Front lowers the queueing delays of all packets waiting in the system. Note that with Drop-Tail, dropping of a packet has no influence on the delay of currently queued packets.

Other push-out techniques include Lower Priority First (LPF, [94, 104]), Complete Buffer Partitioning (CBP, [104]), and Partial Buffer Sharing (PBS, [94]). LPF always drops packets from the lowest backlogged priority queue. CBP assigns a dedicated amount of buffer space to each class, and drops traffic when this dedicated buffer is full. PBS uses a partitioning scheme similar to CBP, but the decision to drop is made after having looked at the aggregated backlog of all classes. The static partitioning of buffers in LPF, CBP, and PBS is not suitable for relative per-class service differentiation, since no *a priori* knowledge of the incoming traffic is available [48].

2.1.4 DiffServ Deployment

Despite the availability of algorithms suitable for implementing the different DiffServ PHB's, deploying the Expedited Forwarding service as originally specified in [87] turns out to be more difficult than initially expected. The lack of deployment is in part due to open issues regarding the configuration of the components in charge of the resource reservations, that is, the bandwidth brokers. On the one hand, a centralized bandwidth broker, as advocated in [115], is a single point of failure, which may be undesirable for a service with strong guarantees such as the EF service. On the other hand, maintaining consistency with distributed bandwidth brokers schemes remained an open problem, as discussed in [143]. Additionally, the potential difficulties in realizing the service with bursty traffic exhibited in [29] imply that the total amount of EF traffic must be only a small fraction of the network capacity to be able to guarantee low queueing delays [142].

Assured Forwarding seems more amenable to deployment, but relies on weaker service guarantees. The main focus of the research on QoS networks in the past five years has thus been to strengthen the service assurance that can be given within the context of class-based services such as Assured Forwarding.

Two approaches have emerged: some efforts focused on quantifying the differentiation between classes of traffic, without enforcing absolute service guarantees, while other efforts attempt to provi-

sion absolute service guarantees for certain classes, without quantifying the differentiation between other classes.

2.2 Proportional Service Differentiation

Proportional service differentiation, initially proposed by Dovrolis et al. [47] in their Proportional Differentiated Services model is an effort to quantify the differentiation between classes of traffic without absolute service guarantees. The Proportional Differentiated Services model for instance attempts to enforce that the ratios of delays or loss rates of successive priority classes be roughly constant. Proportional Differentiated Services was proposed at approximately the same time as the work by Moret and Fdida on proportional queue control [112], which is a scheduling algorithm to realize proportional delay differentiation.

Proportional service differentiation can be implemented through scheduling algorithms and/or buffer management algorithms. The service guarantees are enforced on a per-node basis and do not require any communication between participating nodes. We next present the scheduling and buffer management algorithms that have been proposed for proportional differentiation.

2.2.1 Scheduling

The majority of work on per-class service differentiation suggests to use well-known fixed-priority, e.g., [115], or rate-based scheduling algorithms, e.g., [69]. A few scheduling algorithms have been specifically designed for proportional delay differentiation.

A number of scheduling algorithms, including those we describe in this dissertation, are based on a rate allocation. Rate allocation to classes of traffic for meeting service guarantees is illustrated by the Generalized Processor Sharing (GPS) algorithm [119]. GPS traffic consists of sessions, which can be flows or classes of traffic. GPS takes a fluid-flow interpretation of traffic, which means that multiple sessions can be served simultaneously at the link governed by GPS. Each session i is allocated a weight ϕ_i. GPS is work-conserving, which means that a GPS link is always busy serving traffic when a backlog is present. Traffic from a given backlogged session, say session j, is served at

a service rate at least equal to $\frac{\phi_j}{\sum_i \phi_i} C$, where C is the total capacity of the GPS link. Approximations of GPS in a packet network, where the fluid-flow assumption does not hold, include Packetized GPS (PGPS, [119]) and Weighted Fair Queueing (WFQ, [45]).

With respect to proportional delay differentiation, the Proportional Queue Control Mechanism (PQCM, [112]) and Backlog-Proportional Rate (BPR, [50]) are variations of GPS. Both PQCM and BPR dynamically adjust service rate allocations of classes to meet proportional guarantees. The service rate allocation is based on the backlog of classes at the scheduler. For two classes with backlogs $B_1(t)$ and $B_2(t)$, at a link of capacity C, PQCM assigns a service rate of

$$r_1(t) = \frac{B_1(t)}{B_1(t) + \alpha B_2(t)} C \ ,$$

to the first class, where $0 < \alpha < 1$ is the proportional differentiation factor characterizing the ratio of the delays of the first class over the delays of the second class. BPR extends PQCM to an arbitrary number of classes. In BPR, the class-i service rate is set to

$$r_i(t) = \frac{B_i(t)}{\sum_j \frac{s_j}{s_i} B_j(t)} \ ,$$

where $\frac{s_j}{s_i}$ characterizes the proportional delay guarantee between classes i and j.

Different from the rate-based schedulers discussed above, a number of algorithms instead use dynamic time-dependent priorities to provide proportional delay guarantees. For instance, Waiting-Time Priority (WTP, [50]) implements a scheduling algorithm with dynamic time-dependent priorities initially proposed in [92], Ch. 3.7. A class-i packet, which arrives at time τ, is assigned a time-dependent priority as follows. If the packet is backlogged at time $t > \tau$, then WTP assigns this packet a priority of $(t - \tau) \cdot \delta_i$, where δ_i is a class-dependent priority coefficient [92]. WTP packets are transmitted in the order of their priorities. In [50], the coefficients δ_i are chosen so that

$$\delta_1 = k \cdot \delta_2 = k^2 \cdot \delta_3 = \ldots = k^Q \cdot \delta_Q \ ,$$

resulting in a delay differentiation under high loads, where Class-$(i+1)$ Delay $\approx k \cdot$Class-i Delay.

The Mean-Delay Proportional scheduler (MDP, [113]) has a dynamic priority mechanism similar to WTP, but uses estimates of the average delay of a class to determine the priority of that class. Thus, the priority of a class-i packet is set to $\delta_i \cdot \overline{D_i}(t)$, where $\overline{D_i}(t)$ is the estimated average delay for class i, averaged over the entire up-time of the link. The coefficients δ_i, are as in WTP, i.e., $\delta_1 = k \cdot \delta_2 = k^2 \cdot \delta_3 = \ldots = k^Q \cdot \delta_Q$.

The Hybrid Proportional Delay scheduler (HPD, [46, 51]) uses a combination of waiting-time and average experienced delay to determine the priority of a given packet. Therefore, the priority of a given class is set to

$$\delta_i(g(t-\tau) + (1-g)\overline{D_i}(t)) \ ,$$

with $0 < g < 1$.

A slightly different approach, pursued by the Weighted-Earliest-Due-Date scheduler of [20], is to provide proportional differentiation in terms of probabilities of deadline violation for a set of classes.

2.2.2 Buffer Management

The Proportional Loss Rate (PLR) dropper [48] is specifically designed to support proportional differentiated services. PLR enforces that the ratio of the loss rates of two successive classes remains roughly constant at a given value. There are two variants of PLR. PLR(M) uses only the last M arrivals for estimating the loss rate of a class, whereas PLR(∞) has no such memory constraints. Average Drop Distance (ADD, [21]) is a variant of PLR(M) which aims at providing loss differentiation regardless of the timescale chosen for computing the loss rates.

Different from PLR and ADD, the authors of [95] propose an algorithm that attempts to enforce end-to-end proportional loss differentiation. To that effect, the proposed algorithm records information about the loss rates observed at each hop in the packet header, using a technique similar to Dynamic Packet State [143].

The recently proposed Class-Distance-Based-Priority-Delay-Loss scheduler (C-DBP-Delay-Loss, [146]) tries to provide proportional delay and proportional loss differentiation in a single

algorithm. Note that C-DBP-Delay-Loss is more recent than our initial proposal for an algorithm combining scheduling and buffer management [102]. At any time, C-DBP-Loss-Delay keeps state information as a set of (m_i, k) pairs, where, for each class i, m_i represents the number of packets that have been successfully transmitted in the last k packets. For each packet, C-DBP-Loss-Delay computes the distance between the current state of the system, and a "failure state", defined as any state where m_i is less than a desired value $\widehat{m}_i$. The lower the distance, the higher the priority assigned to the packet. The authors of [146] conjecture that setting $\widehat{m}_i = k - \delta_i$, where δ_i is selected as in WTP and MDP, can be used to provide proportional loss and delay differentiation.

2.3 Other Class-Based Services

The Proportional Differentiated Services model aims at strengthening the guarantees of the Diff-Serv architecture by quantifying the differentiation provided by AF-like services. In parallel to these efforts to strengthen the guarantees of the Assured Forwarding service, other researchers have explored different directions in the design space of Figure 1.1 [73, 78, 100].

For instance, the Alternative Best-Effort (ABE) service explores simpler mechanisms for service differentiation. The ABE service considers two traffic classes. The first class obtains absolute delay guarantees, and the second class has no delay guarantees, but is given a lower loss rate than the first class. Scheduling and buffer management algorithms for the ABE service are presented in [78], and rely on a combined scheme called Duplicate Scheduler with Deadlines, which enforces delay guarantees for the first class by dropping all traffic that has exceeded a given delay bound. A type of service similar to ABE, called Balanced Forwarding, is proposed in [73].

The Dynamic Core Provisioning service model [100] is a class-based service, which supports absolute delay bounds, and qualitative loss and throughput differentiation, but no proportional differentiation. The mechanisms used in [100] prevent violations on service guarantees by dynamically adjusting scheduler service weights and packet dropping thresholds in core routers. Traffic aggregates are dimensioned at the network ingress by a distributed admission control mechanism that uses knowledge of the entire traffic present in the network. Full knowledge of the traffic travers-

ing a network is generally not available in practice, and the algorithm needs to be approximated.

Chapter 3

A Framework for Per-Class Service Guarantees

In this chapter, we introduce a framework for reasoning about per-class service differentiation in a packet network without information on traffic arrivals. We will use the framework described in the present chapter as a basis for the mechanisms we propose throughout this dissertation.

In particular, the proposed framework will allow us to describe a scheme for providing service differentiation at the output link of a router. The scheme relies on treating buffer management and scheduling as two instances of the same problem, namely, management of the transmission queue. That is, buffer management and scheduling are considered in a single step. We give the name *Joint Buffer Management and Scheduling* (JoBS) to this scheme, which we present in [102, 103]. This dissertation will show that using algorithms based on JoBS at the output link of a router enables us to enhance class-based service differentiation without any a priori information on the traffic arrivals.

In parallel to our efforts, there were other proposals that considered scheduling and buffer management in a single step [106, 146]. We assert that our approach is more rigorous and more general. In [146], Striegel and Manimaran use a scheme combining buffer management and scheduling to simultaneously provide proportional differentiation to losses and delays in the context of the C-DBP-Delay-Loss algorithm discussed in Chapter 2. There are no absolute guarantees. In [106], Liu et al. use a joint buffer management and scheduling scheme in the context of input-queued switches to address the issue of maximizing both the aggregate throughput and buffer utilization at the same time, revisiting a problem first described by Lapiotis in [99]. The approaches in [99, 106] solely focus on efficiently using router resources, and do not attempt to apply the combined scheme to

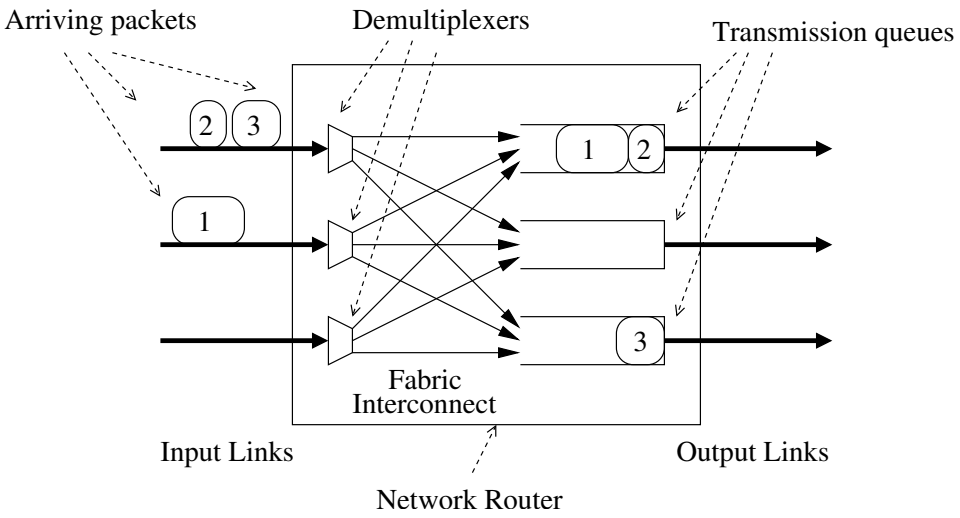


Figure 3.1: **Router architecture.** In this dissertation, we consider an output-queued router, where all queueing occurs at the output links, in the transmission queues. There is no contention at the input links or in the fabric interconnect. The figure shows a 3×3 switch.

provide any service guarantees.

The remainder of this chapter is organized as follows. In Section 3.1, we first give an informal discussion of the operations used by an algorithm based on the Joint Buffer Management and Scheduling scheme. In Section 3.2, we discuss the notions of arrival, input, and output curves, which are adapted from the network calculus formalism originally presented by Cruz in [41, 42]. We use this formalism to introduce the metrics we use to quantify and enforce service differentiation. In Section 3.3, we delve into the details of the service guarantees of Quantitative Assured Forwarding [34] we are interested in providing.

3.1 Overview

In this section, we present an overview of the JoBS scheme. We start by outlining the assumptions we use in our presentation, before succinctly describing the operations one can use in the context of JoBS.

3.1.1 Assumptions

We consider an individual router in the network. For the presentation of this framework, we consider that all traffic backlogged in the router is queued in the transmission queues at the output links. In other words, we assume that the router under consideration uses an output queueing architecture. In Figure 3.1, we give a simplified representation of the router (or switch) architecture we consider. The router consists of N input and output links ($N = 3$ in the example of Figure 3.1), connected by a fabric interconnect. Demultiplexers are used to direct packets from the input links to the proper output links. Each output link is governed by a transmission queue. In the figure, packets flow from left to right. Two packets, marked 1 and 2, are in contention for the same output link, while a third packet, 3, is coming from the same input link as 2. The output queueing architecture implies that the switch interconnect is fast enough to move packets from all input ports to the output ports and completely avoid any contention at the input links or in the switch fabric. In the example of Figure 3.1, packets 2 and 3 are immediately sent to their respective output links and do not create a backlog at the input link they share.

By avoiding traffic backlogs at the input links or in the fabric, the output queueing assumption enables us to solely focus on the operations performed at the output links. However, if C is the capacity of an input link and N is the number of input links, output queueing requires the switch interconnect to have a speed-up of at least N, that is, to have a throughput of at least NC. The throughput requirement on the interconnect can pose practical challenges when N and C are both high, that is, for routers with a large number of high-speed line cards. Even though current hardware may be able to handle the required throughput for interconnects with $N \approx 10$ and $C \approx 10$ Gbps, memory access speeds prevent us from writing packets into the output links transmission queues at such speeds [66].

Therefore, typical high-speed routers generally use input queues in lieu of the demultiplexers of Figure 3.1. Popular router architectures using input queues include Virtual Output Queueing (VOQ, introduced in [12]), where all traffic is queued at N^2 input queues, and Combined Input-Output Queueing (CIOQ), where traffic can be backlogged at both input and output links. Our framework can be extended to CIOQ switches as follows. Previous research contributions showed

that it is possible to perfectly emulate output queueing routers with CIOQ routers with a speed-up of only two, and specific interconnect arbitration mechanisms, such as Critical Cells First (CCF, [35]). Therefore, with CCF and an interconnect throughput of $2C$, we can implement our approach in CIOQ switches.

Another solution to implement output queueing in high speed switches is the Feedback Output Queueing (FOQ) architecture proposed in [66]. FOQ approximates output queueing at high speeds with a modest speedup (between one and two) using a feedback control algorithm that drops packets at the input lines, based on the congestion experienced at the output lines. Because FOQ switches only enqueue traffic at the output links, FOQ switches rely on an output queuing architecture. Hence, our proposed framework directly applies to FOQ switches.

In addition to assuming an output queueing router architecture, we take a fluid-flow interpretation of traffic. That is, the output link is viewed as simultaneously serving traffic from several classes. Since actual traffic is sent in discrete-sized packets, a fluid-flow interpretation of traffic is idealistic. However, scheduling algorithms that closely approximate fluid-flow schedulers with rate guarantees are available [119, 157]. In Chapter 5, we discuss in more detail how we realize the fluid-flow interpretation in a packet network.

Last, we assume for now that no a priori information on the traffic arrivals is available.

3.1.2 JoBS Operations

With the assumptions described above, let us now consider a single router in the network and describe the operations performed by algorithms based on the JoBS scheme. In the router under consideration, each output link performs per-class buffering of arriving traffic and traffic is transmitted from the buffers using a rate-based scheduling algorithm such as [119, 157], with a dynamic, time-dependent service rate allocation for classes. Traffic from the same class is transmitted in a First-Come-First-Served order. There is no admission control and no policing of traffic.

The set of performance requirements are specified to the router as a set of per-class QoS guarantees. As an example, for three classes, the QoS guarantees could be of the form:

- $\frac{\text{Class-2 Delay}}{\text{Class-1 Delay}} \approx 2$,

- $\frac{\text{Class-3 Loss Rate}}{\text{Class-2 Loss Rate}} \approx 10^{-1}$, or
- Class-3 Delay ≤ 5 *ms*.

Using the definitions of Chapter 1, the first two guarantees are proportional guarantees, characterized by a proportional differentiation factor, and the last one is an absolute guarantee, characterized by a bound. All QoS guarantees are deterministic. Statistical guarantees as in [27, 28], which, for instance, provide a delay bound d_i to class-i traffic with probability $1-\varepsilon$ where $\varepsilon > 0$ are outside the scope of this dissertation.

The set of guarantees we consider can be any mix of proportional and absolute guarantees. There is no limit on the number of classes that are supported, and we further require that the algorithms used to implement the service be independent of the choice of specific bounds or proportional differentiation factors for each class. Our objective is that the proposed service generalizes all previous efforts on class-based services.

Because we want to support absolute guarantees and do not use admission control, a set of service guarantees may be infeasible at certain times. For example, it may be impossible to meet both a delay bound and a loss rate bound at the time of a burst of traffic. In case the system of service guarantees is infeasible, some guarantees may need to be relaxed. For instance, proportional guarantees may be relaxed in favor of absolute bounds, or loss guarantees may be relaxed in favor of delay guarantees. We assume that all QoS guarantees are given a precedence order, which is used to determine which constraints are relaxed in case of an infeasible system.

The service rate allocation operates as follows. For each arrival, the service rate allocation to traffic classes is modified so that all QoS service guarantees are met. If there exists no feasible rate allocation that meets all service guarantees, traffic is dropped, either from a new arrival or from the current backlog. We identify two methods of modifying the service rate allocation for meeting the service guarantees:

- Using a *predictive* service rate allocation and buffer management. Here, a prediction on the service differentiation received by all backlogged traffic is made, and based on the prediction,

the service rates are adjusted and traffic is dropped so that per-class QoS guarantees will be met in the future. This approach is discussed in Chapter 4.

- Using a *reactive* service rate allocation and buffer management. Here, the performance of the system with respect to service differentiation is continuously monitored, and compared to the offered per-class QoS guarantees. Upon each arrival, the service rate allocation to each class is adjusted in an effort to attenuate the difference between the service experienced and the QoS guarantees. This approach is presented in Chapter 5.

Using a predictive service rate allocation allows to formulate the service differentiation problem in terms of an optimization problem, as we will demonstrate in Chapter 4. On the other hand, a reactive service rate allocation can help characterize the service rate allocation in terms of a closed-loop problem, as we will show in Chapter 5.

3.2 Formal Description of the Metrics Used in JoBS

We next provide a more formal presentation of the metrics that we use to quantify and enforce service differentiation. To that effect, we use results from the network calculus [41, 42] to express backlog and delay of traffic in the transmission queue of a router. These definitions are useful to describe algorithms based on a reactive service rate allocation and buffer management. Then, we provide a formal definition of the metrics used in the context of a predictive service rate allocation and buffer management. Last, we discuss how we extend the delay and loss metrics to per-class metrics. In this section, we assume that the link at which service differentiation is provided has a capacity C and a total buffer space B.

We assume that all traffic that arrives to the link is marked to belong to one of Q classes. In general, we expect Q to be small, e.g., $Q = 4$. We use a convention whereby a class with a lower index receives a better service. We use $a_i(t)$ and $l_i(t)$, respectively, to denote the class-i arrivals and the amount of class-i traffic dropped ("lost") at time t. We use $r_i(t)$ to denote the service rate allocated to class-i at time t. The service rate of a class i is a fraction of the output link capacity, which can vary over time, and is set to zero if there is no backlog of class-i traffic in the transmission

queue. That is,

$$r_i(t) > 0 \ ,$$

only if there is a backlog of class-i traffic in the buffer and

$$r_i(t) = 0 \ ,$$

otherwise. In addition, we require that scheduling be work-conserving, that is,

$$\sum_i r_i(t) = C \ , \tag{3.1}$$

if there is at least one backlogged class at time t. Note that systems that are not work-conserving, i.e., where the link may be idle even if there is a positive backlog, may be undesirable for networks that need to achieve a high resource utilization or that need to support the best-effort service.

3.2.1 Arrival, Input and Output Curves

We now introduce the notions of *arrival curve*, *input curve*, and *output curve* for a traffic class i in the time interval $[t_1, t_2]$. The arrival curve A_i and the input curve R_i^{in} of class i are defined as

$$A_i(t_1, t_2) = \int_{t_1}^{t_2} \underbrace{\lambda_i(x)dx}_{a_i(x)} \ , \tag{3.2}$$

where $\lambda_i(t)$ is the instantaneous class-i arrival rate at time t, defined by

$$\lambda_i(t) = \frac{dA_i}{dt}(t) \ ,$$

and

$$R_i^{in}(t_1, t_2) = A_i(t_1, t_2) - \int_{t_1}^{t_2} \underbrace{\xi_i(x)dx}_{l_i(x)} \ , \tag{3.3}$$

where $\xi(t)$ is the instantaneous class-i drop rate at time t, defined by

$$\xi_i(t) = \frac{d(A_i - R_i^{in})}{dt}(t) \ .$$

From Eqn. (3.3), the difference between the arrival and input curve is the amount of dropped traffic. The output curve R_i^{out} of class-i is the transmitted traffic in the interval $[t_1, t_2]$, given by

$$R_i^{out}(t_1, t_2) = \int_{t_1}^{t_2} r_i(x)dx \ . \tag{3.4}$$

From now on, we will use the following shorthand notations to denote the arrival, input and output curves at a given time t, respectively:

$$\begin{aligned} A_i(t) &= A_i(0,t) \ , \\ R_i^{in}(t) &= R_i^{in}(0,t) \ , \\ R_i^{out}(t) &= R_i^{out}(0,t) \ . \end{aligned}$$

We refer to Figure 3.2 for an illustration. In the figure, the service rate is adjusted at times t_1, t_2, and t_4, and drops occur at times t_2 and t_3.

The vertical and the horizontal distance between the input and output curves from class i, respectively, are the backlog B_i and the delay D_i. This is illustrated in Figure 3.2 for time t. The delay D_i at time t is the delay of an arrival which is transmitted at time t. Backlog and delay at time t are defined as

$$B_i(t) = R_i^{in}(t) - R_i^{out}(t) \ , \tag{3.5}$$

and

$$D_i(t) = \max_{x<t}\{x \mid R_i^{out}(t) \geq R_i^{in}(t-x)\} \ . \tag{3.6}$$

Upon a traffic arrival, say at time s, JoBS sets new service rates $r_i(s)$ and the amount of traffic to be dropped $l_i(s)$ for all classes. $r_i(s)$ and $l_i(s)$ are selected such that all QoS guarantees can be met at times greater than s. If all service guarantees cannot be satisfied at the same time, then some

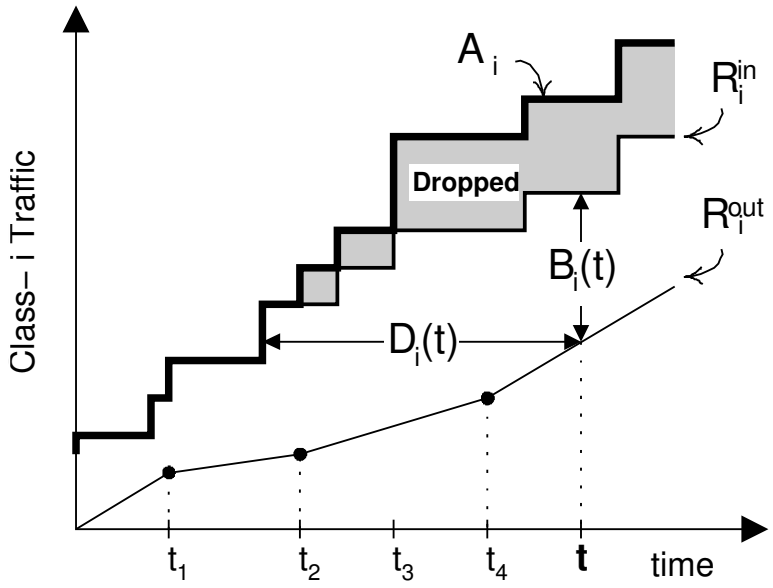


Figure 3.2: **Delay and backlog.** A_i is the arrival curve, R_i^{in} is the input curve and R_i^{out} is the output curve.

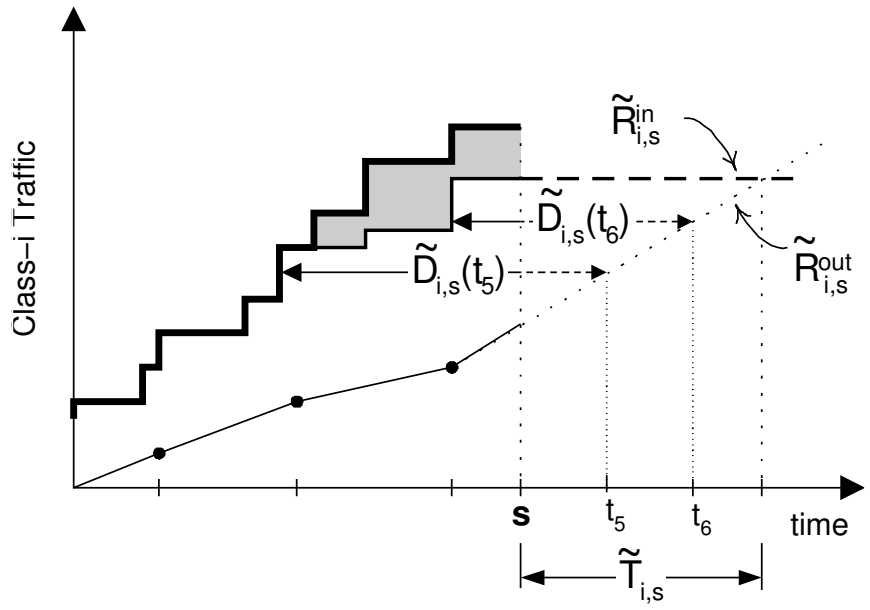


Figure 3.3: **Predicted input curve, predicted output curve, and predicted delays.** The prediction is performed at time s for the time interval $[s, s+\widetilde{T}_{i,s}]$.

guarantees are relaxed in a predetermined order. For instance, absolute guarantees can be given higher precedence than proportional guarantees.

3.2.2 Predictions

The metrics D_i, A_i, R_i^{in}, R_i^{out} and B_i discussed above characterize past and present state of the system. These metrics are therefore useful in the context of a reactive service rate allocation and buffer management. However, D_i, A_i, R_i^{in}, R_i^{out} and B_i do not convey any information regarding potential future behavior and are therefore not adequate in the context of predictive algorithms. Here, we introduce the predictions used by a predictive rate allocation and buffer management algorithm.

For the purpose of the predictions, we assume that the current state of the link will not change after time s. Specifically, indicating predicted values by a tilde (˜), for times $t > s$, we assume that

1. Service rates remain as they are: $\widetilde{r_i}(t) = r_i(s)$,

2. There are no further arrivals: $\widetilde{a}_i(t) = 0$,

3. There are no further drops: $\widetilde{l}_i(t) = 0$.

With these assumptions, we now define the notions of predicted input curve $\widetilde{R}^{in}_{i,s}$, predicted output curve $\widetilde{R}^{out}_{i,s}$, and predicted backlog $\widetilde{B}_{i,s}$, for $t > s$ as follows:

$$\widetilde{R}^{in}_{i,s}(t) = R^{in}_i(s) \ , \tag{3.7}$$

$$\widetilde{R}^{out}_{i,s}(t) = R^{out}_i(s) + (t-s)r_i(s) \ , \tag{3.8}$$

$$\widetilde{B}_{i,s}(t) = \widetilde{R}^{in}_{i,s}(t) - \widetilde{R}^{out}_{i,s}(t) \ . \tag{3.9}$$

We refer to the *predicted horizon* for class i at time s, denoted as $\widetilde{T}_{i,s}$, as the time when the predicted backlog becomes zero, i.e.,

$$\widetilde{T}_{i,s} = \min_{x>0}\{x \mid \widetilde{B}_{i,s}(s+x) = 0\} \ . \tag{3.10}$$

With this notation, we now express predictions for delays in the time interval $[s, s+\widetilde{T}_{i,s}]$. We define the predicted delay $\widetilde{D}_{i,s}(t)$ at time $t \in [s, s+\widetilde{T}_{i,s}]$ as

$$\widetilde{D}_{i,s}(t) = \max_{t-s<x<t}\{x \mid \widetilde{R}^{out}_{i,s}(t) \geq R^{in}_i(t-x)\} \ . \tag{3.11}$$

If there are no arrivals after time s, the delay predictions are correct, in the sense that the predicted delay at time s is the delay that will be encountered when the traffic element departs the system at time t.

In Figure 3.3, we illustrate the predicted input curve, predicted output curve, and predicted delays for predictions made at time s. In the figure, all values for $t > s$ are predictions and are indicated by dashed lines. The figure includes the predicted delays for times t_5 and t_6.

Note that the prediction given in Eqn. (3.11) assumes that, when drops occur, traffic is dropped from the tail of the queue (Drop-Tail). Indeed, if drops are performed at the head of the queue (Drop-from-Front, [98]), drops affect traffic that was admitted in the past. A Drop-from-Front policy has therefore an influence on *past* values of R^{in}_i, which changes the shape of the entire input

curve at any time traffic is dropped. More formally, with a Drop-from-Front policy, Eqn. (3.11) does not hold anymore and must be replaced by

$$\widetilde{D}_{i,s}(t) = \max_{t-s<x<t} \{x \mid \widetilde{R}^{out}_{i,s}(t) \geq R^{in}_i(t-x) - \int_s^t \xi(\tau) d\tau\} \ , \tag{3.12}$$

Eqn. (3.12) imposes knowledge of future drops to make a meaningful prediction. Because it is practically difficult to evaluate how much traffic will be dropped in the future, we will consider a Drop-Tail policy in the remainder of the dissertation.

3.2.3 Per-Class Delay and Loss Metrics

When defining proportional QoS guarantees as in Chapter 1, e.g., Class-2 Delay $\approx 2 \cdot$ Class-1 Delay or Class-2 Loss Rate $\approx 10^{-1} \cdot$ Class-3 Loss Rate, we have assumed that a single metric is available to specify the "delay" or the "loss" of a class. In general, since there are several packets backlogged from a class, each likely to have a different delay, the notion of "delay of class *i*" needs to be further specified. Likewise, the notion of "loss rate of class *i*" requires further clarification.

3.2.3.1 Delay Metrics

Beginning with the delay metric $D_i(t)$ from Eqn. (3.6), we provide the rationale for our choices of per-class delay metrics.

Instantaneous Delay. The measure $D_i(t)$, given by Eqn. (3.6) describes the delay of the class-*i* packet that is in transmission at time *s*. $D_i(t)$ is a good measure for the delay of all class-*i* traffic if $D_i(t)$ is roughly constant. $D_i(t)$ may be an appropriate metric if the service rate allocation is formulated in terms of a closed-loop control problem, i.e., if the service rate allocation to classes is regarded as taking corrective actions to an "error" in the current rate allocation.

Average Delay. Averaging the instantaneous delay $D_i(t)$ over a time window of length τ provides a simple measure for the history of delays experienced by "typical" class-*i* packets. We obtain

$$D^{avg}_{i,t}(\tau) = \frac{1}{\tau} \int_{s-\tau}^{s} D_i(x) dx \ . \tag{3.13}$$

Alternatively, one may want to give more weight to the most recent delays. Using an exponentially weighted moving average, denoted by $D_{i,t}^{ewma}$, one obtains

$$D_{i,t}^{ewma} = D_{i,t-\tau}^{ewma} + w \cdot (D_i(s) - D_{i,s-\tau}^{ewma}) \ , \tag{3.14}$$

where τ defines the window size of the moving average, and $0 \leq w \leq 1$ is the smoothing factor.

Average delay metrics as defined above only take into consideration the history of delays. Since the recent history of delays may not be a good indicator for the delays to be experienced by currently backlogged traffic, using Eqs. (3.13) and (3.14) may lead to poor predictions of delay guarantee violations. On the other hand, Eqs. (3.13) and (3.14) can be appropriate for closed-loop control, when the service rate allocation attempts to correct past observed behavior. Compared to $D_i(t)$, an average delay metric will yield a more conservative, and likely more stable control, at the expense of a reduced reactivity.

The instantaneous and average delay metrics are suitable for a closed-loop control, characteristic of a reactive service rate allocation. Different from the above, the per-class delay metrics used in the case of a predictive rate allocation should attempt to measure the delay for the currently backlogged traffic. Per-class delay metrics we propose to use in the context of a predictive rate allocation take advantage of the notion of predicted delay $\widetilde{D}_{i,s}(t)$ as defined in Eqn. (3.11). Under the assumption that there are no arrivals and no losses after time s, and using the service rate allocation from time s, the predicted delay $\widetilde{D}_{i,s}(t)$ provides the delay of the packet in transmission at time t. We define two delay metrics for the backlog from class i at time s, one for the worst-case delay and one for the "typical" delay.

Maximum Predicted Delay. As a metric for predicting the worst-case delay of the currently backlogged traffic from class i, we define the *maximum predicted delay* at time s, as

$$\widetilde{D}_{i,s}^{max} = \max_{s<t<s+\widetilde{T}_{i,s}} \widetilde{D}_{i,s}(t) \ . \tag{3.15}$$

If there are no arrivals and no changes to the rate allocation after time s, then $\widetilde{D}_{i,s}^{max}$ is an upper bound of the future delays of traffic which is backlogged at time s.

Average Predicted Delay. We define the *average predicted delay* $\overline{D}_{i,s}$ as the time average of the predicted delays from a class, averaged over the horizon $\widetilde{T}_{i,s}$. We obtain

$$\overline{D}_{i,s} = \frac{1}{\widetilde{T}_{i,s}} \int_{s}^{s+\widetilde{T}_{i,s}} \widetilde{D}_{i,s}(x)dx \ . \tag{3.16}$$

Note that this metric takes into account both the time that has already been spent in the scheduler, and the predicted time before the packet is serviced.

3.2.3.2 Loss Metrics

Similar to delays, there are several sensible choices for defining "loss". In this dissertation, we select one specific loss measure, denoted by $p_i(t)$, which expresses the fraction of lost traffic since the beginning of the current busy period at time t_0. A busy period is a time interval with a positive backlog of traffic. For time x with $\sum_i B_i(x) > 0$, the beginning of the busy period t_0 is given by

$$t_0 = \max_{y<x} \left\{ \sum_i B_i(y) = 0 \right\} \ .$$

So, $p_i(t)$ expresses the fraction of traffic that has been dropped in the time interval $[t_0, t]$, that is,

$$\begin{aligned} p_i(t) &= \frac{\int_{t_0}^{t} l_i(x)dx}{\int_{t_0}^{t} a_i(x)dx} \\ &= 1 - \frac{R_i^{in}(t_0, t^-) + a_i(t) - l_i(t)}{A_i(t_0, t)} \ . \end{aligned} \tag{3.17}$$

with

$$t^- = \sup\{x | x < t\} \ .$$

With the metrics just defined, we can now formally introduce the service guarantees our service architecture can support.

3.3 Quantitative Assured Forwarding

We propose here a formal definition of the Quantitative Assured Forwarding service guarantees. In QAF, service differentiation is enforced over the duration of a busy period. An advantage of enforcing service differentiation over short time intervals such as the busy period is that the output link can react quickly to changes of the traffic load. Further, providing differentiation only within a busy period requires little state information to be maintained. As a disadvantage, at times of low load, when busy periods are short, providing service differentiation only with information on the current busy period can be unreliable. However, at underloaded links transmission queues are mostly idle and all service classes receive a high-grade service. So, from now on, we consider $t_0 = 0$.

We want the QAF service to be able to generalize all recently proposed class-based services. In particular, the proposed service should be able to emulate the Assured Forwarding and Expedited Forwarding services of the DiffServ architecture. We also want the proposed service to support the guarantees of the Proportional Differentiated Services architecture, and of other class-based services such as the Alternative Best Effort service.

Quantitative Assured Forwarding provides proportional and absolute differentiation on losses, delays, and throughputs of classes of traffic. The guarantees are expressed as follows.

An absolute delay bound on class i is specified as

$$\forall t : D_i(t) \leq d_i \ , \tag{3.18}$$

where d_i is the desired upper bound on the delay of class i. An absolute loss rate bound for class i is defined by

$$\forall t : p_i(t) \leq L_i \ . \tag{3.19}$$

Throughput guarantees guarantee a minimum throughput to classes that are backlogged. The throughput guaranteed to classes that are not backlogged should be usable by backlogged classes.

A throughput guarantee is defined by a bound on the service rates of class i, specified as

$$\forall t : B_i(t) > 0 \, , r_i(t) \geq \mu_i \, . \tag{3.20}$$

Proportional differentiation on delay and loss, respectively, is defined, for all t such that $B_i(t) > 0$ and $B_{i+1}(n) > 0$, as

$$\frac{D_{i+1}(t)}{D_i(t)} = k_i \ \ , \tag{3.21}$$

and

$$\frac{p_{i+1}(t)}{p_i(t)} = k_i' \, , \tag{3.22}$$

where k_i and k_i' are proportional differentiation factors, that is, constants that quantify the proportional differentiation desired. Note that even though Eqs. (3.21) and (3.22) impose that proportional differentiation only apply to classes with contiguous indices, simply reordering class indices enables to provide proportional differentiation between any two classes.

We have fully specified the service guarantees of Quantitative Assured Forwarding, and we now turn to a description of the specifics of the algorithms we propose to realize the QAF service.

Chapter 4

Service Rate Allocation and Traffic Drops: An Optimization Problem

In this chapter, we discuss two algorithms based on the JoBS scheme to provide the service guarantees of the Quantitative Assured Forwarding service at a given output link. The algorithms we propose in this chapter use a predictive rate allocation and buffer management. That is, every time an arrival occurs, the algorithms predict the delays that will be experienced by backlogged traffic if the current rate allocation stays in force and if no traffic is dropped. If the predicted delays indicate impending service violations, the rate allocation is changed so that the predictions satisfy all proportional and absolute guarantees on delays. Traffic is dropped, subject to loss guarantees, only if no feasible rate allocation can satisfy all delay and throughput guarantees. In other words, at a given time s, the goal is to compute the rate allocation $r_i(s)$ and traffic drops $l_i(s)$ so that service guarantees are met in the future, at times $t \geq s$.

We describe a first algorithm where the service rate allocation and traffic drops are expressed in terms of the solution to an optimization problem [103]. An optimization problem is defined by

- An optimization variable, which is the unknown in the problem,
- A set of constraints, which define the solution space of the problem, and
- An objective function, which selects a specific solution in the solution space.

The optimization is performed each time s at which an arrival occurs. Considering a number Q of classes, the optimization variable is a time-dependent vector

$$\mathbf{x}_s = (r_1(s) \dots r_Q(s)\ l_1(s) \dots l_Q(s))^T \ ,$$

which contains the service rates $r_i(s)$ and the amount of traffic to be dropped $l_i(s)$ for each class.

The optimization problem has the form

$$\begin{aligned} &\textbf{Minimize} && F(\mathbf{x}_s) \\ &\textbf{Subject to} && g_j(\mathbf{x}_s) = 0, \quad j = 1, \dots, M \\ &&& h_j(\mathbf{x}_s) \geq 0, \quad j = M+1, \dots, N, \end{aligned} \tag{4.1}$$

where $F(.)$ is the objective function, and the g_j's and h_j's are constraints. More precisely, g_j's define equality constraints, and h_j's define inequality constraints. The optimization at time s is done with knowledge of the system state before time s, that is the optimizer knows R_i^{in} and R_i^{out} for all times $t < s$, and A_i for all times $t \leq s$.

We use an objective function that minimizes the amount of traffic to be dropped, and, as a secondary objective, aims at maintaining the current service rate allocation. The first objective prevents traffic from being dropped unnecessarily, and the second objective tries to avoid frequent fluctuations of the service rate allocation. The constraints are the Quantitative Assured Forwarding service guarantees defined in Chapter 3 (*QoS constraints*), and constraints on the link and buffer capacity (*system constraints*).

The computational complexity of the algorithm effectively in charge of the rate allocation and traffic drops is determined by the number and the type of constraints, and by the frequency of running the above optimization. For now, we assume that sufficient computing resources are available. The optimization-based algorithm can be used as a reference algorithm against which more practical scheduling and dropping algorithms can be compared. We will later approximate the optimization-based algorithm with a heuristic algorithm that incurs less computational overhead.

The remainder of this chapter is organized as follows. In Section 4.1, we present the constraints

of the optimization problem in details. In Section 4.2, we give a formal definition of the objective function. In Section 4.3, we introduce a heuristic algorithm to approximate the solution to the optimization problem. In Section 4.4, we evaluate the performance of both the optimization and heuristic algorithms with respect to meeting service guarantees through simulation experiments, and provide a numerical comparison with other algorithms available in the literature.

4.1 System and QoS Constraints

We next discuss the constraints of the optimization problem. There are two types of constraints: *system constraints* describe constraints and properties of the output link, while *QoS constraints* enforce the desired service differentiation. All constraints in the optimization problem must be expressed in function of the optimization variable $\mathbf{x}_s$. Hence, the object of the present section is to to derive expressions for the constraint functions $g_j(\mathbf{x}_s)$ and $h_j(\mathbf{x}_s)$ in Eqn. (4.1).

4.1.1 System Constraints

The system constraints specify physical limitations and properties at the output link and the associated transmission queue where service guarantees are enforced.

The first system constraint enforces that scheduling at the output link is work-conserving. At a work-conserving link serving Q classes of traffic, Eqn. (3.1) holds for all times t where $\sum_i B_i(t) > 0$. For the optimization problem as defined by Eqn. (4.1), Eqn. (3.1) is written as an equality constraint defined by the function $g_1(\mathbf{x}_s)$ as follows:

$$g_1(\mathbf{x}_s) = \sum_{i=1}^{Q} r_i(s) - C \; . \tag{4.2}$$

The second set of system constraints characterizes bounds on the service rates and traffic drops. In particular, service rates and traffic drops cannot be negative, and the amount of traffic that can be

dropped is bounded by the current backlog. So, we obtain, for each class i, at any time t,

$$\begin{aligned} r_i(t) &\geq 0 \,, \\ l_i(t) &\geq 0 \,, \\ l_i(t) &\leq B_i(t) \,. \end{aligned}$$

We use a convention whereby h_j^i denotes the j-th inequality constraint, applied to class i, and obtain three inequality constraints per class,

$$\begin{aligned} h_2^i(\mathbf{x}_s) &= r_i(s) \,, && i = 1,\ldots,Q \,, \\ h_3^i(\mathbf{x}_s) &= l_i(s) \,, && i = 1,\ldots,Q \,, \\ h_4^i(\mathbf{x}_s) &= B_i(s) - l_i(s) \,, && i = 1,\ldots,Q \,. \end{aligned} \tag{4.3}$$

The last system constraint states that the total backlog in the transmission queue cannot exceed the buffer size B, that is,

$$\sum_{i=1}^{Q} B_i(t) \leq B \,, \tag{4.4}$$

for all times t. Since we have, for all i,

$$B_i(s) = B_i(s^-) + a_i(s) - l_i(s) \,, \tag{4.5}$$

the buffer size constraint in Eqn. (4.4) can be rewritten as

$$\sum_{i=1}^{Q} \big(B_i(s^-) + a_i(s) - l_i(s)\big) \leq B \,, \tag{4.6}$$

where the only unknown is $l_i(s)$. Eqn. (4.6) directly translates into an inequality constraint characterized by

$$h_5(\mathbf{x}_s) = B - \sum_{i=1}^{Q} \big(B_i(s^-) + a_i(s) - l_i(s)\big) \,. \tag{4.7}$$

4.1.2 QoS Constraints

The QoS constraints describe the service guarantees offered by the Quantitative Assured Forwarding Service. We consider two types of QoS constraints, relative constraints and absolute constraints. Relative constraints are used to enforce proportional service differentiation, while absolute constraints translate absolute bounds on the service received. QoS constraints can be expressed for delays, loss rates and service rates.

The number and type of QoS constraints is not limited. However, since absolute QoS constraints may result in an infeasible system of constraints, one or more constraints may need to be relaxed or eliminated at certain times. We assume that the set of QoS constraints is assigned some total order, and that constraints are relaxed in the given order until the system of constraints becomes feasible. For our optimization-based algorithm, we select a specific relaxation order that gives absolute guarantees priority over proportional guarantees, and that gives loss guarantees priority over delay or rate guarantees. So, using "$<$" to denote the order in which guarantees are relaxed, we have

$$\text{RDC} < \text{RLC} < \text{ADC} < \text{ARC} < \text{ALC} < \text{System constraints} . \tag{4.8}$$

QoS constraints for classes which are not backlogged are simply ignored.

Quantitative Assured Forwarding service guarantees in Eqs. (3.18)–(3.22) are expressed in terms of delays, loss, and service rates, but the only parameters the optimization-based algorithm can directly control at time s are the components of the optimization variable $\mathbf{x}_s$, that is, the service rates $r_i(s)$ and the amount of dropped traffic $l_i(s)$. Furthermore, the optimization-based algorithm uses the predictions of the delays of all backlogged traffic to express the delay guarantees. Therefore, we have to rewrite the service guarantees in Eqs. (3.18)–(3.22) as constraints on $r_i(s)$ and $l_i(s)$ depending on the predictions.

We next discuss how each of the service guarantees of the Quantitative Assured Forwarding Service maps to a set of QoS constraints in the optimization problem.

Delay bounds. A class-i delay bound d_i, as specified in Eqn. (3.18), translates into an *Absolute Delay Constraint* (ADC) in the optimization problem.

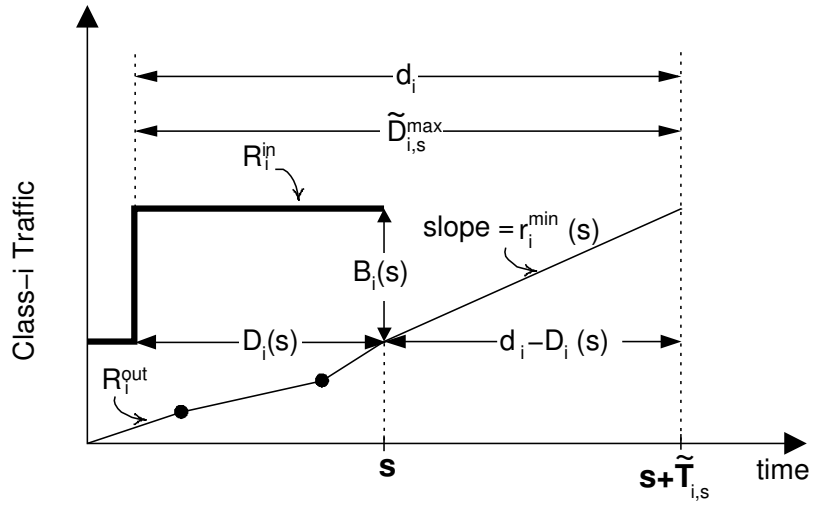


Figure 4.1: **Determining service rates required to meet delay bounds.** The figure indicates the service rate $r_i^{\min}(s)$ required so that the maximum predicted delay satisfies $\widetilde{D}_{i,s}^{max} = d_i$. Allocating at least $r_i^{\min}(s)$ ensures that the delay bound d_i is met for class i.

The delay bound d_i holds if, for any time t, the predicted delays of class i respect the bound d_i. In other words, class i is guaranteed the bound d_i if, for all t,

$$\widetilde{D}_{i,t}^{max} \leq d_i \ , \tag{4.9}$$

where $\widetilde{D}_{i,t}^{max}$ is the maximum predicted delay, defined in Eqn. (3.15). We illustrate the relationship between d_i and the minimum service rate $r_i^{\min}$ required to meet d_i in Figure 4.1. Figure 4.1 describes the worst-case arrival pattern for the maximum predicted delay, where all traffic backlogged at time s arrived in a single burst of traffic. In such a case, the maximum predicted delay $\widetilde{D}_{i,s}^{\max}$ is equal to the predicted delay of the traffic at the tail of the queue, and is given by

$$\widetilde{D}_{i,s}^{\max} = D_i(s) + \widetilde{T}_{i,s} \ ,$$

where $\widetilde{T}_{i,s}$, the predicted horizon defined in Eqn. (3.10), characterizes the time at which the entire backlog at time s will be transmitted if the service rate allocation at time s ($r_i(s) > 0$) is maintained.

Hence, we can express $\widetilde{T}_{i,s}$ with:

$$\widetilde{T}_{i,s} = \frac{B_i(s)}{r_i(s)} \ ,$$

and replacing in the expression for $\widetilde{D}_{i,s}^{\max}$, we get

$$\widetilde{D}_{i,s}^{\max} = D_i(s) + \frac{B_i(s)}{r_i(s)} \ .$$

To satisfy Eqn. (4.9), we obtain

$$D_i(s) + \frac{B_i(s)}{r_i(s)} \leq d_i \ ,$$

We denote by $r_i^{\min}(s)$ the minimum service rate needed at time s to meet the delay bound d_i, and get

$$r_i^{\min}(s) = \frac{B_i(s)}{d_i - D_i(s)} \ .$$

Then, to enforce the delay bound d_i, $r_i(s)$ needs to satisfy:

$$r_i(s) \geq r_i^{\min}(s) \ . \tag{4.10}$$

Using Eqn. (4.5) in Eqn. (4.10) gives, after reorganizing the terms,

$$B_i(s^-) + a_i(s) - l_i(s) - r_i(s)(d_i - D_i(s)) \leq 0 \ . \tag{4.11}$$

Here, the only unknowns are $r_i(s)$ and $l_i(s)$. Thus, the Absolute Delay Constraints are:

$$h_6^i(\mathbf{x}_s) = (d_i - D_i(s))r_i(s) - (B_i(s^-) + a_i(s) - l_i(s)) \ , \quad i = 1, \ldots, Q \ . \tag{4.12}$$

Proportional delay differentiation. We denote the constraints that express proportional delay differentiation between classes, as described in Eqn. (3.21), by *Relative Delay Constraints* (RDC). For the expression of the Relative Delay Constraints, we use the average predicted delay, $\overline{D}_{i,s}$, as defined in Eqn. (3.16), for characterizing the "delay of class i".

To obtain a solution space, rather than a single solution for the service rate allocation, we allow

for some slack in the expression of the relative delay constraints. The relative delay constraints at time s are given as

$$k_i(1-\varepsilon) \leq \frac{\overline{D}_{i+1,s}}{\overline{D}_{i,s}} \leq k_i(1+\varepsilon) \ , \tag{4.13}$$

where $k_i > 1$ is the target proportional differentiation factor and ε ($0 \leq \varepsilon \leq 1$) indicates a tolerance level.

We obtain the following inequality constraints after reorganizing terms in Eqn. (4.13):

$$\begin{aligned} h_7^i(\mathbf{x}_s) &= \overline{D}_{i+1,s} - k_i(1-\varepsilon)\overline{D}_{i,s} \ , \quad i = 1,\ldots,Q-1 \ , \\ h_8^i(\mathbf{x}_s) &= k_i(1+\varepsilon)\overline{D}_{i,s} - \overline{D}_{i+1,s} \ , \quad i = 1,\ldots,Q-1 \ , \end{aligned} \tag{4.14}$$

where $\overline{D}_{i,s}$ and $\overline{D}_{i+1,s}$ can be expressed as a function of the unknowns $r_i(s)$, $r_{i+1}(s)$, $l_i(s)$ and $l_{i+1}(s)$. Since $\overline{D}_{i,s}$ is not a linear function of $\mathbf{x}_s$, $h_7^i(\mathbf{x}_s)$ and $h_8^i(\mathbf{x}_s)$ are not linear.

Next we discuss constraints on the loss rates at the time s of an arrival, $p_i(s)$. Recall that in the definition of $p_i(s)$ given by Eqn. (3.17), all values except $l_i(s)$ are known at time s.

Loss rate bounds. The loss rate bounds of the Quantitative Assured Forwarding service, as defined in Eqn. (3.19), map to *Absolute Loss Constraints* (ALC). Using the definition of the loss rate at time s, $p_i(s)$, given in Eqn. (3.17), the service guarantee Eqn. (3.19) reduces to

$$1 - \frac{R_i^{in}(s^-) + a_i(s) - l_i(s)}{A_i(s)} \leq L_i \ ,$$

which can be rewritten as

$$l_i(s) \leq (L_i - 1)A_i(s) + R_i^{in}(s^-) + a_i(s) \ , \tag{4.15}$$

where $l_i(s)$ is the only unknown. Eqn. (4.15) defines the following inequality constraints:

$$h_9^i(\mathbf{x}_s) = (L_i - 1)A_i(s) + R_i^{in}(s^-) + a_i(s) - l_i(s) \ , \quad i = 1,\ldots,Q \ . \tag{4.16}$$

Proportional loss differentiation. Proportional loss differentiation between classes, as defined by

Eqn. (3.22) results in *Relative Loss Constraints* (RLC). Similar to the RDCs, we provide a certain slack when expressing the Relative Loss Constraints. The RLC for classes $i+1$ and i at time s has the form

$$k_i'(1-\varepsilon') \leq \frac{p_{i+1}(s)}{p_i(s)} \leq k_i'(1+\varepsilon') \ , \tag{4.17}$$

where $k_i' > 1$ is the target differentiation factor, and ε_i' ($0 \leq \varepsilon' \leq 1$) indicates a level of tolerance.

From Eqn. (3.17), at time s, $p_i(s)$ can be expressed as a function of $l_i(s)$. Therefore, the RLCs can be expressed as a function of $\mathbf{x}_s$. From Eqn.(4.17), we get the following set of inequality constraints:

$$\begin{aligned} h_{10}^i(\mathbf{x}_s) &= p_{i+1,s} - k_i'(1-\varepsilon')p_{i,s} \ , \quad i = 1,\ldots,Q-1 \ , \\ h_{11}^i(\mathbf{x}_s) &= k_i'(1+\varepsilon')p_{i,s} - p_{i+1,s} \ , \quad i = 1,\ldots,Q-1 \ . \end{aligned} \tag{4.18}$$

Throughput guarantees. Last, the throughput guarantees of Eqn. (3.20) are mapped to *Absolute Rate Constraints* (ARC), which are expressed as the following inequality constraints:

$$h_{12}^i(\mathbf{x}_s) = r_i(s) - \mu_i \ , \quad i = 1,\ldots,Q \ . \tag{4.19}$$

4.2 Objective Function

Provided that the QoS and system constraints defined above can be satisfied, the objective function will select a solution for $\mathbf{x}_s$. As briefly discussed in the beginning of this chapter, even though the choice of the objective function is a policy decision, we select two specific objectives, which, we believe, have broad validity:

- **Objective 1**: Avoid dropping traffic,
- **Objective 2**: Avoid changes to the current service rate allocation.

The first objective ensures that traffic is dropped only if there is no alternative to satisfy the QoS and system constraints. The second objective tries to hold on to a feasible service rate allocation as long as possible. We give the first objective priority over the second objective.

The following formulation of an objective function expresses the above objectives in terms of a cost function:

$$F(\mathbf{x}_s) = \sum_{i=1}^{Q} (r_i(s) - r_i(s^-))^2 + C^2 \sum_{i=1}^{Q} l_i(s) \ , \tag{4.20}$$

where C is the link capacity. The first term expresses the changes to the service rate allocation and the second term expresses the losses at time s. Note that, at time s, $r_i(s)$ is part of the optimization variable, while $r_i(s^-)$ is a known value. In Eqn. (4.20) we use the quadratic form $(r_i(s) - r_i(s^-))^2$, because

$$\sum_i (r_i(s) - r_i(s^-)) = 0$$

for a work-conserving link with a backlog at time s. The scaling factor C^2 in front of the second sum in Eqn. (4.20) ensures that traffic drops are the dominating term in the objective function.

This concludes the description of the optimization problem. At a given time s, all of the QoS and system constraints are expressed in terms of $r_i(s)$ and $l_i(s)$, that is, as a function of the vector $\mathbf{x}_s$. Summarizing Eqs. (4.2)–(4.20), the optimization problem at the time s of an arrival at the output link under consideration has the form:

$$\begin{array}{lll}
\textbf{Minimize} & \sum_{i=1}^{Q}(r_i(s)-r_i(s^-))^2 + C^2\sum_{i=1}^{Q} l_i(s) & \\
\textbf{Subject to} & \sum_{i=1}^{Q} r_i(s) - C & = 0 \ , \\
& r_i(s) & \geq 0 \ , \quad i = 1,\ldots,Q \ , \\
& l_i(s) & \geq 0 \ , \quad i = 1,\ldots,Q \ , \\
& B_i(s) - l_i(s) & \geq 0 \ , \quad i = 1,\ldots,Q \ , \\
& B - \sum_{i=1}^{Q}(B_i(s^-) + a_i(s) - l_i(s)) & \geq 0 \ , \\
& (d_i - D_i(s))r_i(s) - (B_i(s^-) + a_i(s) - l_i(s)) & \geq 0 \ , \quad i = 1,\ldots,Q \ , \\
& \overline{D}_{i+1,s} - k_i(1-\varepsilon)\overline{D}_{i,s} & \geq 0 \ , \quad i = 1,\ldots,Q-1 \ , \\
& k_i(1+\varepsilon)\overline{D}_{i,s} - \overline{D}_{i+1,s} & \geq 0 \ , \quad i = 1,\ldots,Q-1 \ , \\
& (L_i - 1)A_i(s) + R_i^{in}(s^-) + a_i(s) - l_i(s) & \geq 0 \ , \quad i = 1,\ldots,Q \ , \\
& p_{i+1}(s) - k_i'(1-\varepsilon')p_i(s) & \geq 0 \ , \quad i = 1,\ldots,Q-1 \ , \\
& k_i'(1+\varepsilon')p_i(s) - p_{i+1}(s) & \geq 0 \ , \quad i = 1,\ldots,Q-1 \ , \\
& r_i(s) - \mu_i & \geq 0 \ , \quad i = 1,\ldots,Q-1 \ .
\end{array} \tag{4.21}$$

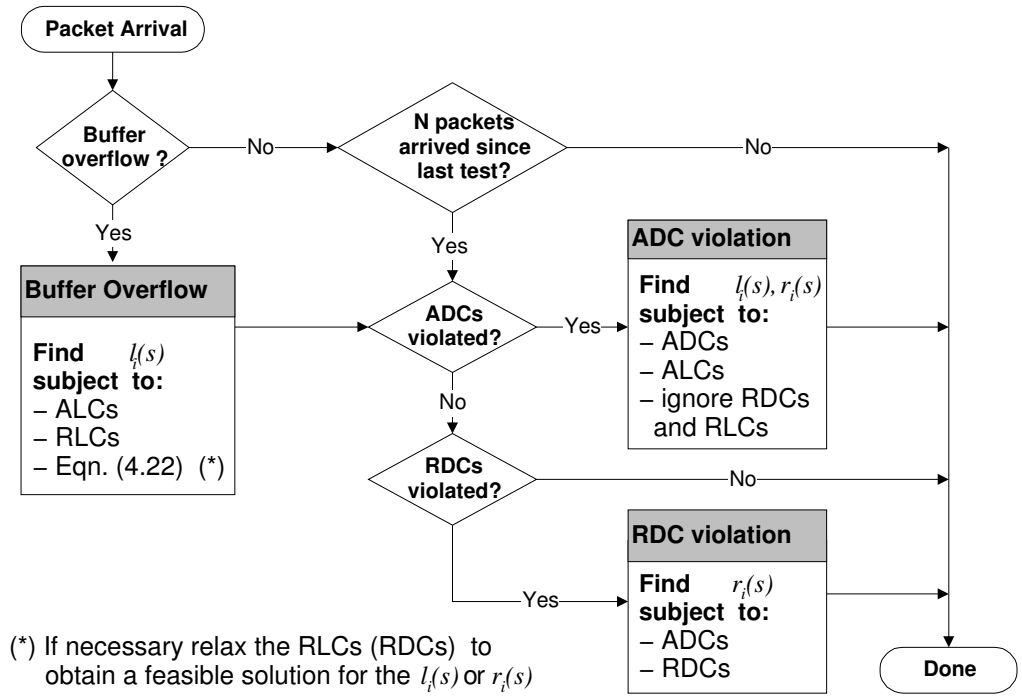


Figure 4.2: **Outline of the heuristic algorithm.** The heuristic algorithm decomposes the optimization problem into several smaller, computationally less intensive, problems.

The system (4.21) is a *non-linear optimization problem*, because (1) the optimization function is quadratic, and (2) we have $\overline{D}_{i,s} = O\left(\frac{1}{r_i(s)}\right)$, which causes the RDC to be non-linear in $\mathbf{x}_s$. A non-linear optimization problem as defined by Eqn. (4.21) can be solved with available numerical algorithms such as [131]. If Eqn. (4.21) yields an empty solution space, some constraints are relaxed, according to the precedence order of Eqn. (4.8).

4.3 Heuristic Algorithm

Approximating the solution to a non-linear optimization problem as presented in this chapter can be performed by well-known techniques such as fuzzy systems, or neural networks. However, these techniques can be computationally expensive. To reduce the computational complexity, we devise a heuristic algorithm, which decomposes the optimization problem into several smaller, computationally less intensive, problems. The heuristic algorithm presented here maintains a feasible rate

allocation until a buffer overflow occurs or a delay violation is predicted. At that time, the heuristic picks a new feasible rate allocation and/or drops traffic. Also, the tests for violations of ADCs and RDCs are not performed for every packet arrival, but only periodically, or when there is a buffer overflow.

As in the optimization-based algorithm, a set of constraints that includes absolute constraints (ALCs or ADCs) may be infeasible at certain times. Then, constraints are relaxed in the order given in Eqn. (4.8).

A high-level overview of the heuristic algorithm is presented in Figure 4.2. We decompose the optimization problem into a number of smaller sub-problems. Each sub-problem characterizes a case that requires to adjust the service rate allocation and/or to drop packets. The proposed heuristic algorithm consists of a sequence of procedures that compute the rate allocation and packet drops required to solve each sub-problem. We next present each of the sub-problems and the associated computations.

Buffer Overflow. If an arrival at time s causes a buffer overflow, one can either drop the arriving packet or free enough buffer space to accommodate the arriving packets. Both cases are satisfied if

$$\sum_i l_i(s) = \sum_i a_i(s) \ . \tag{4.22}$$

The heuristic picks a solution for the $l_i(s)$ which satisfies Eqn. (4.22) and the RLCs in Eqn. (4.17), where we set $\varepsilon' = 0$ to obtain a unique solution. If the solution violates an ALC, the RLCs are relaxed until all ALCs are satisfied. Once the $l_i(s)$'s are determined the algorithm continues with a test for delay constraint violations, as shown in Figure 4.2. Recall that while the algorithm only specifies the amount of traffic which should be dropped from a particular class, and does not select the position in the queue from which to drop traffic, we assume a Drop-Tail dropping policy.

If there are no buffer overflows, the algorithm makes predictions for delay violations only once for every T packet arrivals. The selection of T represents a tradeoff between the runtime complexity of the algorithm and performance of the scheduling with respect to satisfying the constraints. Simulation experiments, as described in Section 4.4, were performed with the value $T = 100$ and

exhibit good performance.

The tests use the current service rate allocation to predict future violations. For delay constraint violations, the heuristic distinguishes three cases.

Case 1: No violation. In this case, the service rates are unchanged.

Case 2: RDC violation. If some RDC (but no ADC) is violated, the heuristic algorithm determines new rate values. Here, the RDCs as defined in Eqn. (4.13) are transformed into equations by setting $\varepsilon = 0$. Together with the work-conserving property, one obtains a system of equations, for which the algorithm picks a solution. If the solution violates an ADC, the RDCs are relaxed until the ADCs are satisfied.

Case 3: ADC violation. Resolving an ADC violation is not entirely trivial as it requires to recalculate the $r_i(s)$'s, and, if traffic needs to be dropped to meet the ADCs, the $l_i(s)$'s. To simplify the task, our heuristic ignores all relative constraints when an ADC violation occurs, and only tries to satisfy absolute constraints.

By simply reordering the terms, we can rewrite the constraint given in Eqn. (4.11), and define a variable ρ_i as

$$\underbrace{\frac{1}{r_i(s)} \frac{B_i(s^-) + a_i(s) - l_i(s)}{d_i - D_i(s)}}_{\rho_i} \leq 1 \; . \tag{4.23}$$

The heuristic algorithm will select the $r_i(s)$ and $l_i(s)$ such that Eqn. (4.23) is satisfied for all i. Initially, rates and traffic drops are set to $r_i(s) = r_i(s^-)$ and $l_i(s) = 0$. Since at least one ADC is violated, there is at least one class with $\rho_i > 1$, where ρ_i is defined in Eqn. (4.23). Now, we apply a greedy method which tries to redistribute the rate allocations until $\rho_i \leq 1$ for all classes. This is done by reducing $r_i(s)$ for classes with $\rho_i < 1$, and increasing $r_i(s)$ for classes with $\rho_i > 1$. If it is not feasible to achieve $\rho_i \leq 1$ for all classes by adjusting the $r_i(s)$'s, the $l_i(s)$'s are increased until $\rho_i \leq 1$ for all i. To minimize the number of dropped packets, $l_i(s)$ is never increased to a point where an ALC is violated.

In terms of computational overhead, the heuristic algorithm requires to solve several linear systems of Q equations, where Q is the number of service classes. The systems of equations used in the heuristic algorithm are made explicit in [101], Appendix B. Solving each of these systems of

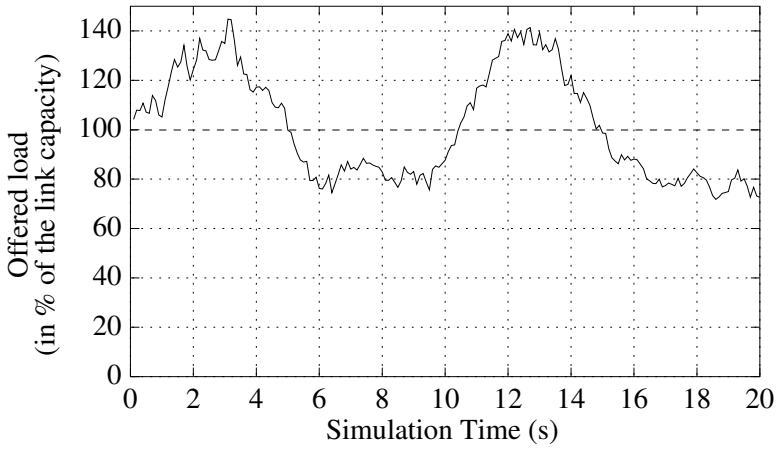


Figure 4.3: **Offered load.** The graph describes the offered load at our simulated bottleneck link.

equations requires to compute the product of a $Q \times Q$ matrix with a vector of dimension Q, which has a computational complexity of $O(Q^2)$.

4.4 Evaluation

We present an evaluation of the algorithms developed in this chapter using packet-level simulation experiments. Our goals are (1) to determine if and how well the desired service differentiation is achieved; (2) to determine how well the heuristic algorithm from Section 4.3 approximates the optimization; and (3) to compare our algorithm with existing proposals for proportional differentiated services.

In the simulations, we evaluate the following four schemes.

- The optimization-based algorithm described in Sections 4.1 and 4.2,
- The heuristic algorithm discussed in Section 4.3. Unless there is a buffer overflow, tests for delay violations are performed once for every $T = 100$ packet arrivals.
- **WTP/PLR(∞) [48]:** We evaluate a combination of the WTP scheduler and the PLR(∞) backlog controller. Both of the WTP and PLR algorithms from [48] are discussed in Chapter 2. Since WTP/PLR(∞) provides a better service differentiation than WTP/PLR(M), we only in-

clude results for WTP/PLR(∞). Note that neither WTP/PLR(∞) nor WTP/PLR(∞) supports absolute guarantees to traffic classes.

- **MDP [113]/Drop-Tail:** The MDP scheduler presented in [113] is discussed in Chapter 2. Since MDP does not provide mechanisms for loss differentiation, we assume a simple Drop-Tail algorithm for discarding packets. As WTP/PLR(∞), MDP does not support absolute QoS guarantees.

We present two simulation experiments using the simulator available in [31]. In the first experiment, we compare the proportional differentiation provided by the optimization-based algorithm, the heuristic algorithm, WTP/PLR(∞), and MDP/Drop-Tail without specifying absolute constraints, at a single node. In the second experiment, we augment the set of constraints by absolute loss and delay constraints on the highest priority class, and show that the algorithms based on the JoBS scheme can effectively provide both proportional and absolute differentiation at a single node.

4.4.1 Simulation Experiment 1: Proportional Differentiation Only

The first experiment focuses on proportional service differentiation, and does not include absolute constraints. We consider a single output link with capacity $C = 1$ Gbps and a buffer size of 6.25 MB. We assume $Q = 4$ classes. The length of each experiment is 20 seconds of simulated time, starting with an empty system. In all experiments, the incoming traffic is composed of a superposition of Pareto sources with $\alpha = 1.2$ and average interarrival time of 300 μs. The shape parameter α of the Pareto distribution essentially characterizes the burstiness of the traffic arrivals. The smaller α, the burstier the traffic. The number of sources active at a given time oscillates between 200 and 550, following a sinusoidal pattern. All sources generate packets with a fixed size of 125 bytes.[1] The resulting offered load is plotted in Figure 4.3. Modulating the number of active Pareto sources by a sinusoidal wave allows us to test our algorithm under very bursty conditions over short timescales, while having the load vary significantly over longer timescales. At any time, each class contributes 25% of the aggregate load.

[1] Packet sizes on the Internet are in fact subject to a multimodal distribution [9], and thus, the simulation presented here is only a simplified model.

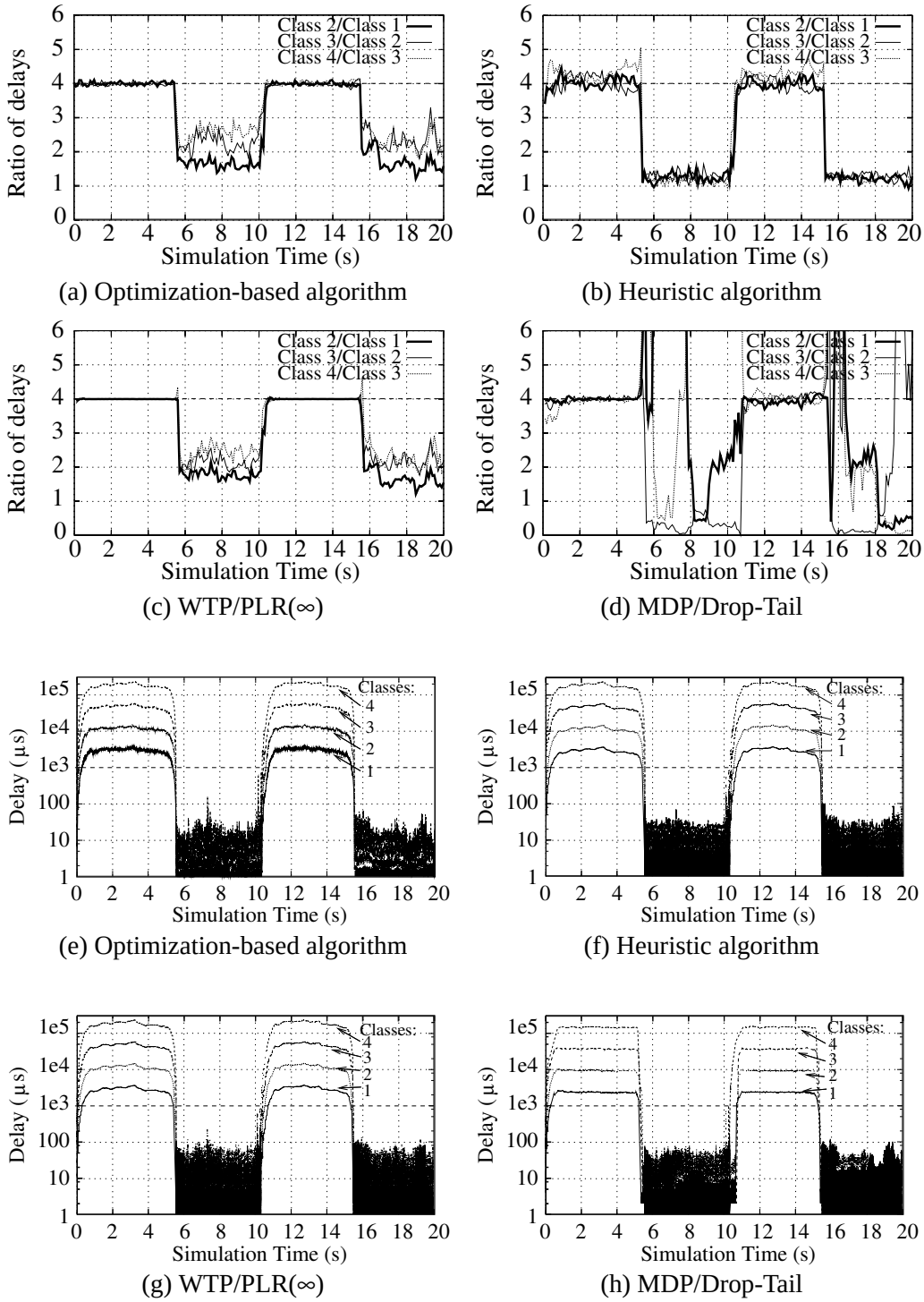


Figure 4.4: **Experiment 1: Proportional delay differentiation.** The graphs show the ratios of the delays for successive classes (a)-(d) and the absolute delay values (e)-(h). The target value for the ratios, indicated by a dashed line, is $k = 4$.

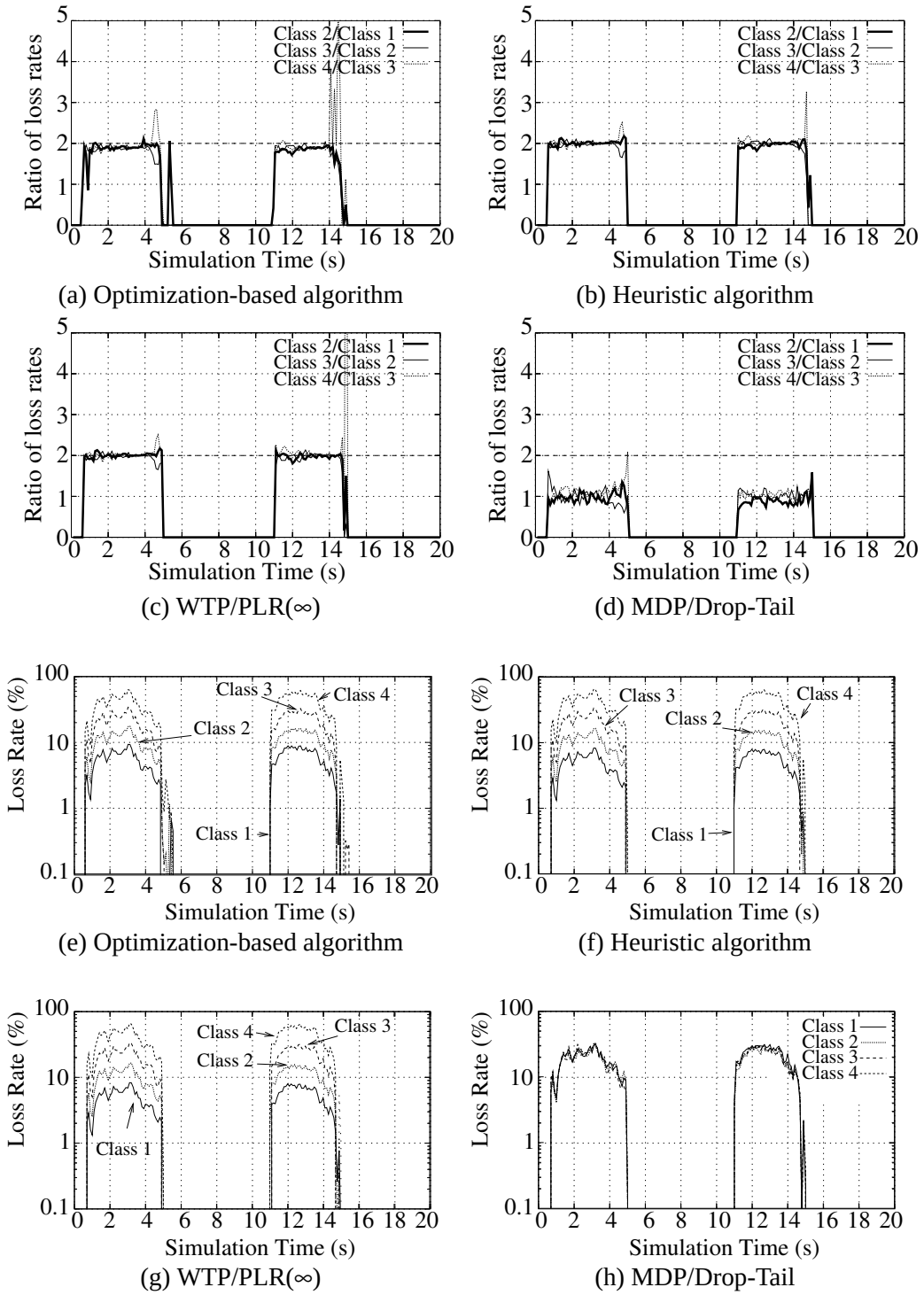


Figure 4.5: **Experiment 1: Proportional loss differentiation.** The graphs show the ratios of loss rates for successive classes (a)-(d) and the loss rates (e)-(h). The target value for the ratios, indicated by a dashed line, is $k' = 2$.

Proportional differentiation factors are set to four for the ratio of delays of two successive classes, and to two for the ratio of loss rates of two successive classes, i. e., we set $k_i = 4$ and $k'_i = 2$ for $i \in \{1,2,3\}$. The tolerance levels are set to $(\varepsilon, \varepsilon') = (0.001, 0.05)$ in the optimization-based algorithm, and to $\varepsilon = 0.01$ in the heuristic algorithm. The results of the experiment are presented in Figures 4.4 and 4.5, where we graph the delays and loss rates, respectively, of successive classes for the optimization-based algorithm, the heuristic algorithm, WTP/PLR(∞), and MDP/Drop-Tail. Using a measure adopted from [48], the plotted delay and loss values are averages over moving time windows of size 0.1 *s*.

When the link load is above 90% of the link capacity, that is, in time intervals $[0\ s, 6\ s]$ and $[10\ s, 15\ s]$, all methods provide the desired service differentiation. The oscillations around the target values in the optimization-based algorithm and the heuristic algorithm are mostly due to the tolerance values ε and ε'. The selection of the tolerance values ε and ε' in our proposed algorithms presents a tradeoff: smaller values for ε and ε' reduce oscillations, but incur more adjustments for the algorithms. When the system load is low, that is, in time intervals $[6\ s, 10\ s]$ and $[16\ s, 20\ s]$, only the optimization-based algorithm and WTP/PLR(∞) manage to achieve some delay differentiation, albeit far from the target values. MDP/Drop-Tail, plotted in Figure 4.4(d), provides some differentiation, but the system seems unstable, particularly after a transient change in the load. At an underloaded link, the absolute values of the delays are very small for all classes, regardless of the scheduling algorithm used, as shown on Figures 4.4(e)-(h), and all classes receive an excellent service. Figures 4.4(e)-(h) also show that the absolute values for the delays are comparable in all schemes. In fact, proportional delay differentiation cannot be realized at low loads without artificially delaying some packets, that is by making the scheduler non-work-conserving.

In Figures 4.5(a) and 4.5(c), we observe all algorithms show some transient oscillations with respect to loss differentiation when the link changes from an overloaded to an underloaded state. These transient spikes during a transition between an overloaded and an underloaded system are caused by the low number of packets dropped when the link becomes underloaded, which makes the ratios of loss rates become less meaningful.

Note that, since proportional differentiation does not guarantee an upper bound, an algorithm

may provide excellent proportional loss differentiation, but have an overall high loss rate, which can translate into poor service. Figures 4.5(e)-(g) show that in the simulations, the loss rates of WTP/PLR(∞) and of our proposed algorithms are very similar. With WTP/PLR(∞) and both of our proposed algorithms, the average of the per-class loss rates is equal to the loss rate obtained with a Drop-Tail policy, plotted in Figure 4.5(h). This shows that, in this experiment, all schemes only drop packets when a buffer overflow occurs.

4.4.2 Simulation Experiment 2: Proportional and Absolute Differentiation

In this second experiment, we evaluate how well our algorithms can satisfy a mix of absolute and proportional guarantees on both delays and losses. Here, we only present results for the heuristic algorithm WTP/PLR(∞) and MDP/Drop-Tail do not support absolute guarantees, and refer to [101] for results obtained with the optimization-based algorithm. The experiment illustrates how traffic is dropped to help meet absolute delay guarantees.

We consider the same simulation setup and the same proportional delay guarantees as in Experiment 1, but add an Absolute Delay Constraint (ADC) for Class 1 such that $d_1 = 1$ *ms*, and we replace the Relative Loss Constraint (RLC) between Classes 1 and 2 by an Absolute Loss Constraint (ALC) for Class 1 such that $L_1 = 1\%$. We call this scenario "with ADC, all RDCs". With the given proportional delay guarantees from Experiment 1, the other classes have "implicit" delay bounds, which are approximately[2] 4 *ms* for Class 2, 16 *ms* for Class 3, and 64 *ms* for Class 4. Removing the RDC between Class 1 and Class 2, we avoid the implicit delay bounds for Classes 2, 3, and 4, and call the resulting constraint set "with ADC, one RDC removed". We also include the results for the heuristic algorithm from Experiment 1, with the ALC on Class 1 replacing the RLC between Classes 1 and 2, and refer to this constraint set as "no ADC, all RDCs". In Figures 4.6(a)–(c) we plot the delays of all packets, and in Figures 4.6(d)–(f) we plot the loss rates of all classes, averaged over time intervals of length 0.1 *s*. We discuss the results for each of the three constraint sets proposed.

[2] Due to the tolerance value ε, the exact values are not integers.

Concerning the experiment "with ADC, all RDCs", Figure 4.6(a) shows that the heuristic maintains the proportional delay differentiation between classes, thus, enforcing the implicit delay bounds for Classes 2, 3, and 4. With a large number of Absolute Delay Constraints, the system of constraints easily becomes infeasible, as pointed out by the following two observations. First, Figure 4.6(d) shows that the loss rates of Classes 2, 3 and 4 are similar. This result illustrates that the heuristic relaxes Relative Loss Constraints to meet the Absolute Delay Constraints.

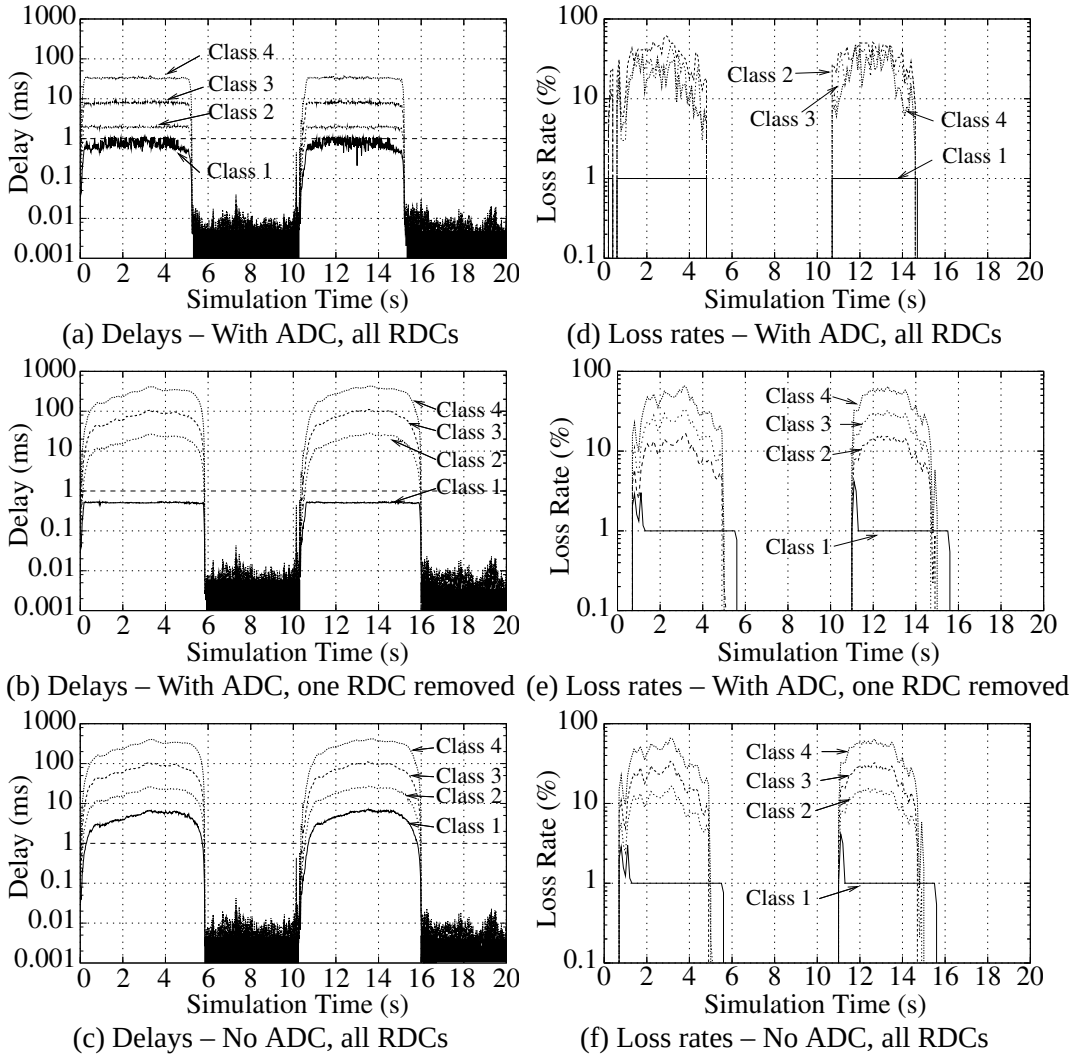
Second, Figure 4.6(a) shows that the absolute delay guarantee d_1 is sometimes violated. However, such violations are rare (over 95% of Class-1 packets have a delay less than 900 μs), and Class-1 packet delays always remain reasonably close to the delay bound d_1. For the experiment "with ADC, one RDC removed", Figure 4.6(b) shows that, without an RDC between Classes 1 and 2, the ratio of Class-2 delays and Class-1 delays can exceed a factor of 10 at high loads. With this constraint set, the absolute delay guarantee d_1 is never violated, and Figure 4.6(e) shows the RLCs are consistently enforced during periods of packet drops.

Finally, for the experiment "no ADC, all RDCs", Figure 4.6(c) shows that, without the ADC, the delays for Class 1 are as high as 5 *ms*. The delay values for Classes 2, 3, and 4 in Figures 4.6(b) and (c) appear similar, especially since we use a log-scale. We emphasize that the values are *not* identical, and that the results are consistent.

4.5 Summary and Remarks

In this chapter, we showed that providing proportional and absolute service guarantees to classes of traffic at an output link without reservations or a priori knowledge of the traffic arrivals can be expressed in terms of an optimization problem.

The formulation of the optimization problem uses predictions on the service that will be received in the future. The constraints of the optimization problem are the service guarantees and properties of the output link under consideration. The objective function of the optimization problem is chosen to minimize traffic losses and changes in the rate allocation. We presented an algorithm that uses the solution to the optimization problem to allocate the service rates and drop traffic



(a) Delays – With ADC, all RDCs (d) Loss rates – With ADC, all RDCs

(b) Delays – With ADC, one RDC removed (e) Loss rates – With ADC, one RDC removed

(c) Delays – No ADC, all RDCs (f) Loss rates – No ADC, all RDCs

Figure 4.6: **Experiment 2: Delay and loss differentiation.** The graphs show the delays of all packets in (a)–(c) and the loss rates of all classes in (d)–(f). All results are for the heuristic algorithm.

so that QAF service guarantees are met.

When the optimization problem does not yield a solution, meaning that it is impossible to satisfy all service guarantees simultaneously, some of the QoS guarantees are selectively ignored, based on a precedence order specified a priori.

Due to the form of the constraints, the optimization problem is a non-linear optimization, which can only be solved numerically. The computational cost of solving a non-linear optimization upon each arrival to the link under consideration may be prohibitive to consider an implementation of an optimization-based algorithm at high-speeds. To reduce the computational overhead of the approach, we described a heuristic algorithm that approximates the optimization-based algorithm.

We showed in a set of simulation experiments that both the optimization-based and heuristic algorithms were effective at providing proportional and absolute per-class QoS guarantees for delay and loss.

Chapter 5

A Closed-Loop Algorithm Based on Feedback Control

In this chapter, we present an algorithm based on a reactive service rate allocation and buffer management [34]. Recall from Chapter 3 that a reactive service rate allocation and buffer management uses measurements of the service experienced and tries to attenuate the difference between the service experienced and the service guarantees offered.

Compared to predictive algorithms such as discussed in the previous chapter, a reactive service rate allocation and buffer management has the following interesting properties. Since mechanisms are based on past measurements, the need for predicting the service that will be experienced in the future is alleviated. In particular, an algorithm based on a reactive service rate allocation and buffer management does not need to compute predicted delays, thereby reducing the total number of operations performed by the algorithm.

In addition, a reactive service rate allocation and buffer management can be described as a closed-loop problem, which enables us to use feedback control theory to analyze the properties of the algorithm. For instance, we will use feedback control analysis to derive configuration bounds that ensure the algorithm will realize the desired service differentiation.

To better reflect the design idea behind the algorithm presented in this chapter, we will refer to the algorithm as a "closed-loop algorithm". Similar to the optimization-based and heuristic algorithms in Chapter 4, the closed-loop algorithm relies on the JoBS scheme to allocate service rates r_i and traffic losses l_i upon each packet arrival, but instead of directly computing the service

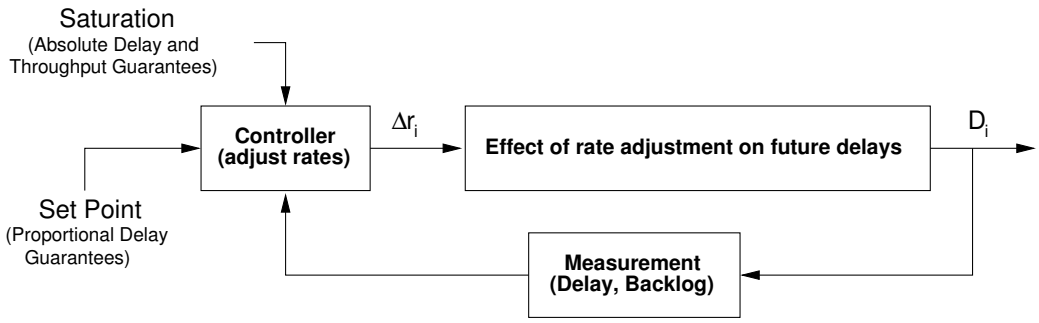


Figure 5.1: **Overview of the closed-loop algorithm.** The figure is an overview of the feedback loop for delay differentiation. A similar mechanism is used for loss differentiation.

rate r_i as in the algorithms of Chapter 4, the closed-loop algorithm computes a change Δr_i in the service rate allocation.

We illustrate in Figure 5.1 how the adjustment Δr_i for class i is computed. Δr_i is the output of a *controller*, which takes a measurement and a *set point* as inputs. The measurement characterizes the value of the class-i backlog and delay of the traffic leaving the router. The set point is obtained from the proportional delay guarantees. The controller is subject to saturation effects, which translate the absolute delay and throughput guarantees, and limit the range of values Δr_i can take. The effect of the rate adjustments on future delays completes the description of the loop, and imposes conditions on the controller to guarantee a stable feedback loop, that is, to ensure that proportional delay guarantees are met.

Recall that, in our JoBS scheme, traffic is dropped, either from a new arrival or from the current backlog, at times when no feasible service rate allocation for meeting all delay guarantees exist, or when the transmission queue is full. A mechanism similar to that presented in Figure 5.1 for service rate allocation governs how traffic is dropped. To satisfy proportional loss guarantees, traffic is dropped from classes according to a drop priority order, defined as follows. For each class, the difference between the loss rate needed for perfect proportional differentiation and the observed loss rate defines an error. The larger the error of a class, the higher its drop priority. For each class i, we stop dropping traffic when either (1) the loss guarantee L_i is reached, or (2) the buffer overflow is resolved or a feasible rate allocation for absolute guarantees exists and there is no need

for dropping traffic anymore.

The remainder of this chapter is organized as follows. In Section 5.1, we start by introducing the formalism used in the description of the closed-loop algorithm. In Sections 5.2 and 5.3 we describe in more details the feedback loops that are used in the design of the algorithm. In Section 5.4, we present simulation results to quantify the performance of the algorithm.

5.1 Notations

In this section, we introduce the notations that we use in the description of our closed-loop algorithm.

5.1.1 A Discrete Time Model

The formalism of the optimization problem and of the associated heuristic of Chapter 4 assumes a continuous time model. That is, all quantities are expressed in function of the continuous time variable t. The description of our closed-loop algorithm, on the other hand, relies on a discrete, event-driven time model, where events are traffic arrivals.

We use $t[n]$ to denote the time of the n-th event in the current busy period, and $\Delta t[n]$ to denote the time elapsed between the n-th and $(n+1)$-th events. We use a shorthand notation with square brackets to specify that a quantity is evaluated at a given event. For instance, instead of writing $a_i(t[n])$ and $l_i(t[n])$, we use $a_i[n]$ and $l_i[n]$ to denote the class-i arrivals and the amount of class-i traffic dropped at the time of the n-th event, respectively. Likewise, we use $r_i[n]$ to denote the service rate allocated to class-i at the time of the n-th event. As in Chapters 3 and 4, we assume bursty arrivals with a fluid-flow service, that is, the output link is viewed as simultaneously serving traffic from several classes.

Let $t[0]$ define the beginning of the busy period. In the discrete time model, the arrival, input,

and output curves for class i at the n-th event, $A_i[n]$, $R_i^{in}[n]$ and $R_i^{out}[n]$ are defined by

$$\begin{aligned} A_i[n] &= \sum_{k=0}^{n} a_i[k] \ , \\ R_i^{in}[n] &= A_i[n] - \sum_{k=0}^{n} l_i[k] \ , \\ R_i^{out}[n] &= \sum_{k=0}^{n-1} r_i[k]\Delta t[k] \ , \end{aligned} \tag{5.1}$$

respectively, by simply rewriting Eqs. (3.2)–(3.4).

Expressing Eqs. (3.5) and (3.6) in terms of events, we have

$$B_i[n] = R_i^{in}[n] - R_i^{out}[n] \ ,$$

and

$$D_i[n] = t[n] - t\left[\max\{k < n \mid R_i^{out}[n] \leq R_i^{in}[k]\}\right] \ . \tag{5.2}$$

Eqn. (5.2) characterizes the delay of the class-i traffic that departs at the n-th event. The loss rate at the n-th event, $p_i[n]$, is simply

$$p_i[n] = \frac{A_i[n] - R_i^{in}[n]}{A_i[n]} \ . \tag{5.3}$$

Last, the service guarantees of the Quantitative Assured Forwarding service are expressed as in Eqs. (3.18)–(3.22).

5.1.2 Rate Allocation and Drop Decisions

We now sketch a closed-loop solution for realizing the service differentiation specified in Eqs. (3.18)–(3.22) at the output link of a router with capacity C and buffer size B. The assumptions on the router architecture outlined in Chapter 3, Section 3.1, still hold. We still assume per-class buffering of incoming traffic, and that traffic from each class is transmitted in a FIFO manner. In the proposed solution, the service rates $r_i[n]$ and the amount of dropped traffic $l_i[n]$ are adjusted at each event n so that the constraints defined by Eqs. (3.18)–(3.22) are met. As before, if not all constraints

in Eqs. (3.18)–(3.22) can be met at the n-th event, some service guarantees need to be temporarily relaxed. We assume that the order in which guarantees are relaxed is given by Eqn. (4.8).

Recall from Eqn. (4.10) that a minimum service rate allocated to class i is required to meet an absolute delay bound on class i, d_i. If class i has, in addition, an absolute throughput bound μ_i, the expression for the minimum rate needed by class i at the n-th event, becomes [1]

$$r_i^{\min}[n] = \max\left\{\frac{B_i[n]}{d_i - D_i[n]}, \mu_i \cdot \chi_{B_i[n]>0}\right\} . \tag{5.4}$$

The service rate that can be allocated to class i is bounded by the output link capacity minus the minimum service rates needed by the other classes, that is, by $r_i^{\max}$ with

$$r_i^{\max}[n] = C - \sum_{j \neq i} r_j^{\min}[n] .$$

Therefore, the service rate can take any value $r_i[n]$ with

$$r_i^{\min}[n] \leq r_i[n] \leq r_i^{\max}[n] ,$$

subject to the constraint $\sum_i r_i[n] \leq C$. Given this range of feasible values, $r_i[n]$ can be selected to satisfy proportional delay differentiation.

We view the computation of $r_i[n]$ in terms of the recursion

$$r_i[n] = r_i[n-1] + \Delta r_i[n] , \tag{5.5}$$

where $\Delta r_i[n]$ is selected such that the constraints of proportional delay differentiation are satisfied at event n. From Eqs. (5.1) and (5.2), the delay $D_i[n]$ at the n-th event is a function of $r_i[k]$ with $k < n$. By monitoring $D_i[n]$ we can thus determine the deviation from the desired proportional differentiation resulting from past service rate allocations, and infer the adjustment $\Delta r_i[n] = f(D_i[n])$ needed to attenuate this deviation.

If no feasible service rate allocation for meeting all delay guarantees exists at the n-th event, or

[1] For any expression "*expr*", we define $\chi_{expr} = 1$ if "*expr*" is true and $\chi_{expr} = 0$ otherwise.

if there is a buffer overflow at the n-th event, traffic must be dropped, either from a new arrival or from the current backlog. Loss differentiation determines which class(es) suffer(s) traffic drops at the n-th event.

We obtain a recursion to express loss differentiation. We rewrite Eqn. (5.3) as the difference equation

$$\begin{aligned} p_i[n] &= \frac{A_i[n] - R_i^{in}[n]}{A_i[n]} \\ &= \frac{A_i[n-1] + a_i[n] - (R_i^{in}[n-1] + a_i[n] - l_i[n])}{A_i[n]} \\ &= \frac{A_i[n-1] - R_i^{in}[n-1] + l_i[n]}{A_i[n]} \\ &= p_i[n-1]\frac{A_i[n-1]}{A_i[n]} + \frac{l_i[n]}{A_i[n]} \ . \end{aligned} \tag{5.6}$$

From Eqn. (5.6), we can determine how the loss rate of class i evolves if traffic is dropped from class i at the n-th event. Thus, we can determine the set of classes that can suffer drops without violating absolute loss bounds. In this set, we choose the class whose loss rate differs by the largest amount from the objective of Eqn. (3.22).

Having expressed the service rate and the loss rate in terms of a recursion, we can characterize the service rate allocation and dropping algorithm as feedback control problems. In the next sections, we will describe two feedback problems: one for delay and absolute rate differentiation ("delay feedback loop"), and one for loss differentiation ("loss feedback loop").

5.2 The Delay Feedback Loop

In this section, we present feedback loops which enforce the desired delay and rate differentiation given by Eqs. (3.18), (3.20), and (3.21). We have one feedback loop for each class with proportional delay differentiation. In the feedback loop for class i, we linearize the control loop around an operating point, and derive a stability condition on the linearized control loop, similar to a technique used in [77, 107, 108, 120]. While the stability condition derived does not ensure that the non-linear

control loop converges, the stability condition gives useful guidelines for selecting the configuration parameter of the controller.

An alternative to using a linear approximation of the non-linear system under consideration is to directly apply non-linear control techniques to derive the stability conditions. Non-linear control techniques, e.g., adaptive control [136], resort to algorithms such as gradient estimators. It is not immediate how a gradient estimator could be implemented to be executed upon each packet arrival in a network router. Furthermore, adaptive control theory is used to dynamically estimate unknown parameters that remain constant over time [136]. All quantities in the feedback loops we are studying vary over time, which implies that some approximations have to be made to use adaptive control theory. These approximations, e.g., assuming that the backlog remains constant over a very short time interval, are similar to the approximations we will use to linearize the feedback loops, so that there is no immediate advantage of using adaptive control in the design of our algorithm.

5.2.1 Objective

Let us assume for now that all Q classes are offered proportional delay guarantees and no absolute delay guarantees. Later, this assumption will be relaxed. The set of constraints given by Eqn. (3.21) leads to the following system of equations:

$$\begin{aligned} D_2[n] &= k_1 \cdot D_1[n] \ , \\ &\vdots \\ D_Q[n] &= \left(\prod_{j=1}^{Q-1} k_j\right) D_1[n] \ . \end{aligned} \tag{5.7}$$

Let $m_i = \prod_{j=1}^{i-1} k_j$ for $i > 1$, and $m_1 = 1$. We define a "weighted delay" of class i at the n-th event, denoted by $D_i^*[n]$, as

$$D_i^*[n] = \left(\prod_{k=1,\ k\neq i}^{Q} m_k\right) D_i[n] \ . \tag{5.8}$$

The weighted delay $D_i^*[n]$ is the delay of class i at the n-th event, multiplied by a scaling factor expressing the proportional delay differentiation desired. By multiplying each line of Eqn. (5.7) with $\prod_{j\neq i} m_j$, we see that the desired proportional delay differentiation is achieved for all classes

if

$$\forall i, j\ , \forall n : D_i^*[n] = D_j^*[n]\ . \tag{5.9}$$

Eqn. (5.9) is equivalent to

$$\forall i\ , \forall n : D_i^*[n] = \overline{D}^*[n]\ ,$$

where

$$\overline{D}^*[n] := \frac{1}{Q}\sum_i D_i^*[n]\ . \tag{5.10}$$

We set $\overline{D}^*[n]$ to be the set point common to all delay feedback loops. The feedback loop for class i reduces the difference $|\overline{D}^* - D_i^*[n]|$ of class i from the common set point $\overline{D}^*[n]$.

Remark: If proportional delay differentiation is requested for some, but not for all classes, constraints as in Eqn. (5.7) can be defined for each group of classes with contiguous indices. Then, the feedback loops are constructed independently for each group.

5.2.2 Service Rate Adjustment

Next, we determine how to adjust the service rate to achieve the desired proportional delay differentiation. Let $e_i[n]$, referred to as "error", denote the deviation of the weighted delay of class i from the set point, i.e.,

$$e_i[n] = \overline{D}^*[n] - D_i^*[n]\ . \tag{5.11}$$

Note that the sum of the errors is always zero, that is, for all n,

$$\sum_i e_i[n] = Q\overline{D}^*[n] - \sum_i D_i^*[n] = 0\ .$$

If proportional delay differentiation is achieved, we have $e_i[n] = 0$ for all classes. We use the error $e_i[n]$ to compute the service rate adjustment $\Delta r_i[n]$ needed for class i to satisfy the proportional delay differentiation constraints. From Eqn. (5.11), we note that if $e_i[n] < 0$ then $D_i^*[n] > \overline{D}^*[n]$, which means that class i delays are too high with respect to the desired proportional delay differentiation. Therefore, $r_i[n]$ must be increased. Conversely, $e_i[n] > 0$ indicates that class i delays are too low,

and $r_i[n]$ must be decreased. Hence, the rate adjustment $\Delta r_i[n]$ is a decreasing function of the error $e_i[n]$, written as $\Delta r_i[n] = f(e_i[n])$, where $f(.)$ is a monotonically decreasing function. We choose

$$\Delta r_i[n] = K[n] \cdot e_i[n] \ , \tag{5.12}$$

where $K[n] < 0$, which, in feedback control terminology, is the controller. An advantage of the controller in Eqn. (5.12) is that it requires a single multiplication. Another advantage is that, at any n, we have

$$\sum_i \Delta r_i[n] = K[n] \sum_i e_i[n] = 0 \ . \tag{5.13}$$

From Eqn. (5.13), the controller produces a work-conserving system, as long as the initial condition $\sum_i r_i[0] = C$ is satisfied.

We next derive two conditions on $K[n]$. The first condition makes the feedback loops stable, in the sense that they attenuate the errors $e_i[n]$ over time. The second condition on $K[n]$ ensures that the rate adjustments $\Delta r_i[n]$ do not create a violation of the absolute delay and rate constraints.

5.2.3 Deriving a Stability Condition on the Delay Feedback Loop

The first condition we derive is a stability condition on $K[n]$, which ensures that the delay feedback loops attenuate the errors $e_i[n]$ over time. We linearize the delay feedback loops to obtain the stability condition, using a set of assumptions. Then, we express the relationship between the service rate adjustments Δr_i and the delays D_i. We use this relationship to derive the stability condition.

Assumptions. We consider a virtual time axis, where the event numbers, n, are equidistant sampling times. The analysis of the control loop is performed on the virtual time axis, so that the stability condition on the control loop applies on the virtual time axis. The stability condition applies to real time only if the skew between virtual time and real time can be neglected, that is, if we assume that, over very short time intervals such as a busy period, class-i packets arrive at an almost constant rate. We note that, over a busy period, the aggregate arrival rate remains roughly equal to the link capacity, because, if the aggregate arrival rate remains below the link capacity for too long, the

queue becomes empty and the busy period ends. If the aggregate arrival rate remains above the link capacity for a while, the queue overflows, and the rate of traffic admitted (i.e., not dropped) to the queue becomes equal to the amount of traffic leaving the queue, that is, to the link capacity. So, assuming that class-i packets arrive at an almost constant rate is equivalent to assuming that the traffic mix between classes does not change much in the router over a busy period.

In addition to the assumption that the traffic mix does not change much over a busy period, we assume that over a busy period, the class-i backlog, delay, and service rate remain in the vicinity of an operating point characterized by a triplet $\left(B_i, D_i, r_i\right)$. So, we consider that the class-i backlog, delay and service rate are only subject to small variations over the duration of a busy period. The intuition behind this assumption is that (1) the variations on the backlog are bounded by the buffer size, which we expect to be relatively small, and (2) the changes in the service rate allocation should be relatively limited when the traffic mix does not change much. This also indicates that the delays do not vary much, because the delays are a function of the service rates and the backlogs.

In practice, we cannot be certain of the validity of the assumptions we just made, and cannot make any claim as to the stability of the delay feedback loops resulting from the analysis presented here. However, the numerical data in Section 5.4 suggests that the loops converge adequately well.

With the assumptions above, we next describe the effect of the rate adjustment $\Delta r_i[n]$ on the delay $D_i[n]$.

Effect of the rate adjustment on future delays. To express the relationship between rate and delays, we start by defining $\tau_i[n]$ as:

$$D_i[n] = t[n] - t[n - \tau_i[n]] \ .$$

In other words, $\tau_i[n]$ denotes the delay of class-i traffic departing at the n-th event, expressed as a number of events. We formalize here the small variation assumption on the delays.

Assumption (A1). The delay of class-i traffic does not vary significantly between events n and $(n+1)$, i.e.,

$$D_i[n+1] \approx D_i[n].$$

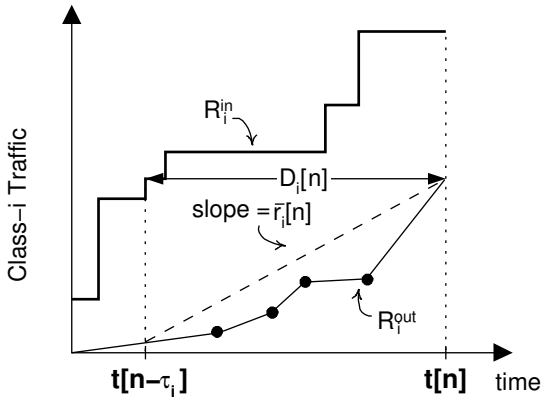


Figure 5.2: **Definition of the average rate, $\overline{r_i}$.** This figure shows the relationship between $D_i[n]$, τ_i and $\overline{r_i}[n]$.

This implies

$$\tau_i[n] \approx \tau_i[n+1] \ .$$

We will, from now on, refer to $\tau_i[n]$ and $\tau_i[n+1]$ as τ_i.

Let us define $\overline{r_i}[n]$ as the average rate experienced by the class-i traffic departing at the n-th event over the time this class-i traffic was backlogged. Using Assumption (A1), we have

$$\overline{r_i}[n] = \frac{B_i[n-\tau_i]}{D_i[n]} \ , \tag{5.14}$$

which we illustrate in Figure 5.2. Figure 5.2 shows that traffic leaving the router at $t[n]$ with a delay $D_i[n]$ has been subject to a changing service rate allocation between its arrival time $t[n-\tau_i]$ and $\tau[n]$, indicated by changes in the slope of the output curve R_i^{out}. The line between the values of the output curve at times $t[n-\tau_i]$ and $\tau[n]$ represents the average service rate $\overline{r_i}[n]$. We express the relationship between $\Delta r_i[n]$ on $\overline{r_i}[n]$ to model the effects of a rate adjustment $\Delta r_i[n]$ on the delay. This is where we use the assumption that the backlog is only subject to small variations.

Assumption (A2). The backlog of class-i traffic does not vary significantly between events $(n-\tau_i)$ and $(n+1-\tau_i)$, that is,

$$B_i[n+1-\tau_i] \approx B_i[n-\tau_i].$$

With Assumptions (A1) and (A2), we obtain the following relationship between r_i and $\overline{r_i}$ on the virtual time axis where events are equally spaced:

$$\overline{r_i}[n+1] = \frac{(\tau_i - 1)\overline{r_i}[n] + r_i[n]}{\tau_i} \ . \tag{5.15}$$

We just characterized the relationship between the service rate at event n and the average rate that will be experienced by class-i traffic departing at event $(n+1)$. Let us now define

$$\Delta\overline{r_i}[n+1] = \overline{r_i}[n+1] - \overline{r_i}[n] \ . \tag{5.16}$$

Combining Eqs. (5.15) and (5.16), we get

$$\Delta\overline{r_i}[n+1] = \frac{(\tau_i - 1)\Delta\overline{r_i}[n] + \Delta r_i[n]}{\tau_i} \ . \tag{5.17}$$

Eqn. (5.17) describes the relationship between a change in the service rate and a change in the average rate.

We now derive the relationship between $\Delta\overline{r_i}[n]$ and a change in the delay of class i, denoted as $\Delta D_i[n]$, and defined by

$$\Delta D_i[n+1] = D_i[n+1] - D_i[n] \ .$$

Since we have from Eqn. (5.14) that

$$D_i[n] = \frac{B_i[n-\tau_i]}{\overline{r_i}[n]} \ ,$$

and

$$D_i[n+1] = \frac{B_i[n+1-\tau_i]}{\overline{r_i}[n+1]} \ ,$$

we get

$$\Delta D_i[n+1] = \frac{B_i[n+1-\tau_i]}{\overline{r_i}[n+1]} - \frac{B_i[n-\tau_i]}{\overline{r_i}[n]} \ . \tag{5.18}$$

We use the small variation assumption on the service rate to linearize Eqn. (5.18):

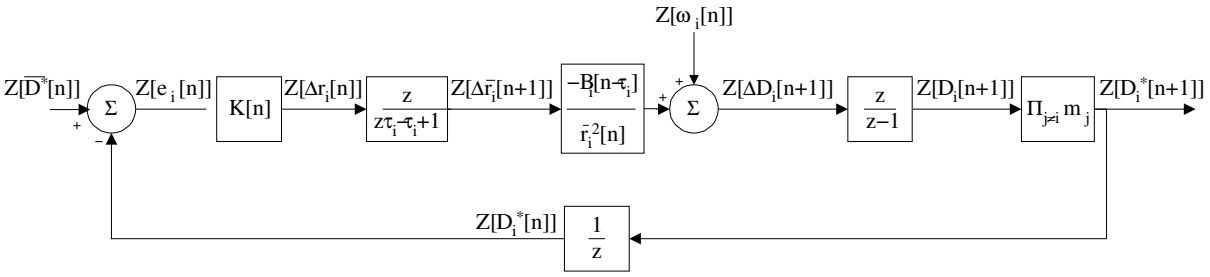


Figure 5.3: **The class-*i* delay feedback loop.** This model uses z-transforms of the relationships derived in Section 5.2.2.

Assumption (A3). The variations in the average rate are small compared to the average rate. This is expressed as

$$\Delta\overline{r_i}[n+1] \ll \overline{r_i}[n] \ .$$

Using Assumptions (A1), (A2), and (A3), a first order expansion of Eqn. (5.18) gives

$$\Delta D_i[n+1] = -\frac{B_i[n-\tau_i]}{\overline{r_i}^2[n]}\Delta\overline{r_i}[n+1] + \omega_i[n] \ , \tag{5.19}$$

where $\omega_i[n]$ is the error in the evaluation of $\Delta D_i[n+1]$ resulting from Assumptions (A1), (A2) and (A3). Then, the relationship between delay variations and the delay is given by

$$D_i[n+1] = \sum_{k=0}^{n+1} \Delta D_i(k) \ , \tag{5.20}$$

$D_i[n+1]$ is used to compute $D_i^*[n+1]$, using Eqn. (5.8). From Eqs. (5.10) and (5.11), $D_i^*[n+1]$ defines the new error $e_i[n+1]$ at the $(n+1)$-th event, that is, the next time a rate adjustment is performed. This remark completes the description of a linearized model of the delay feedback loop. We can now turn to the derivation of a stability condition for our linearized model.

Modeling the loop using z-transforms. The derivation of the stability condition on the linearized model relies on a modeling of the loop using z-transforms of Eqs. (5.8)–(5.20). We denote the z-transform of a function $f[n]$ by $Z[f[n]]$, defined as

$$Z[f[n]] = \sum_{n=0}^{+\infty} f[n]z^{-n} \ .$$

Eqs. (5.8), (5.11), (5.12), (5.19) are unchanged when using z-transforms. Eqn. (5.17) yields

$$Z[\Delta\overline{r_i}[n+1]] = (\tau_i - 1) \cdot \frac{Z[\Delta\overline{r_i}[n]]}{\tau_i} + \frac{Z[\Delta r_i[n]]}{\tau_i} \ ,$$

which gives, using the property that for any continuous function f, $Z[f[n]] = \frac{1}{z}Z[f[n+1]]$, that

$$Z[\Delta\overline{r_i}[n+1]] = (\tau_i - 1) \cdot \frac{Z[\Delta\overline{r_i}[n+1]]}{z\tau_i} + \frac{Z[\Delta r_i[n]]}{\tau_i} \ .$$

By reordering the terms we obtain

$$Z[\Delta\overline{r_i}[n+1]] \left(1 - \frac{\tau_i - 1}{z\tau_i}\right) = \frac{Z[\Delta r_i[n]]}{\tau_i} \ ,$$

which is equivalent to

$$Z[\Delta\overline{r_i}[n+1]] = \frac{z}{z\tau_i - \tau_i + 1} Z[\Delta r_i[n]] \ .$$

Similarly, using z-transforms, Eqn (5.20) becomes

$$Z[D_i[n+1]] = \frac{z}{z-1} Z[\Delta D_i[n+1]] \ .$$

Also, the relationship between the weighted delay at the $(n+1)$-th and n-th iterations is given by

$$Z[D_i^*[n]] = \frac{1}{z} Z[D_i^*[n+1]] \ .$$

The z-transforms discussed above are summarized in Figure 5.3, where we give a representation of the class-i delay feedback loop using z-transforms.

We notice that in the class-i delay feedback loop of Figure 5.3, some quantities (e.g., τ_i, B_i, $\overline{r_i}$) are time-dependent. This does not cause stability problems if the product of all individual blocks in Figure 5.3 (called the "loop gain"), is non-increasing over time. Since the coefficient $K[n]$ is time-dependent, we have to select $K[n]$ so that the loop gain is non-increasing over time.

Denoting the loop gain by $G(z)$, a necessary and sufficient condition for the loop to be stable is

that the roots of the characteristic equation

$$1 + G(z) = 0 \ ,$$

have a module less than one [71]. Taking the products of all blocks in Figure 5.3, we get

$$G(z) = -\frac{1}{z}\frac{z}{z-1}\frac{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]K[n]}{\overline{r_i}^2[n]}\frac{z}{z\tau_i - \tau_i + 1} \ .$$

The negative sign comes from the fact that $D_i^*[n]$ is subtracted from $\overline{D}^*[n]$ to obtain $e_i[n]$. We use the small variation assumption on the service rate to further simplify the expression for $G(z)$. We use the following approximation:

$$\Delta\overline{r_i}[n+1] \approx \Delta r_i[n] \ ,$$

which enables us to approximate the gain of the second block, $\frac{z}{z\tau_i+1-\tau_i}$ by 1. with this approximation, we get a new loop gain $G'(z)$ as follows

$$G'(z) = -\frac{1}{z}\frac{z}{z-1}\frac{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]K[n]}{\overline{r_i}^2[n]} \ ,$$

The characteristic equation of the approximate system is

$$1 - \frac{1}{z-1}\frac{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]K[n]}{\overline{r_i}^2[n]} = 0 \ ,$$

which has exactly one root,

$$\widehat{z} = 1 + \frac{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]K[n]}{\overline{r_i}^2[n]} \ .$$

With the root $\widehat{z}$, we obtain the following stability condition

$$\left| 1 + \frac{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]K[n]}{\overline{r_i}^2[n]} \right| \leq 1 \ ,$$

or, equivalently,

$$1+\frac{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]K[n]}{\overline{r_i}^2[n]} \geq -1 \tag{5.21}$$

$$1+\frac{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]K[n]}{\overline{r_i}^2[n]} \leq 1\,. \tag{5.22}$$

All quantities in Eqn. (5.22), with the exception of $K[n]$, are positive. Hence, the condition described by Eqn. (5.22) simply reduces to $K[n] \leq 0$. The condition in Eqn. (5.21) becomes, after reordering the terms,

$$K[n] \geq -2 \cdot \frac{\overline{r_i}^2[n]}{\left(\prod_{j\neq i} m_j\right) B_i[n-\tau_i]}\,. \tag{5.23}$$

Since, from Eqn. (5.14), we have

$$\begin{aligned}\frac{\overline{r_i}^2[n]}{B_i[n-\tau_i]} &= \frac{\frac{B_i[n-\tau_i]^2}{D_i[n]^2}}{B_i[n-\tau_i]} \\ &= \frac{B_i[n-\tau_i]}{D_i[n]^2}\,,\end{aligned}$$

Eqn (5.23) can be rewritten as

$$K[n] \geq -2 \cdot \frac{B_i[n-\tau_i]}{\left(\prod_{j\neq i} m_j\right) D_i^2[n]}\,. \tag{5.24}$$

The condition given by Eqn. (5.24) requires to keep a history of the backlogs. The need to maintain a backlog history can be alleviated, by replacing Assumption (A2) by the slightly stronger assumption:

Assumption (A2'). The backlog of class-*i* traffic does not vary significantly between events $(n-\tau_i)$ and n, that is,

$$B_i[n-\tau_i] \approx B_i[n].$$

Assumption (A2') allows us to get a simplified expression for the stability condition for the class-*i* delay feedback loop:

$$-2 \cdot \frac{B_i[n]}{\prod_{j\neq i} m_j \cdot D_i^2[n]} \leq K[n] \leq 0\,.$$

Since $K[n]$ must be common to all classes for Eqn. (5.13) to hold, we finally get

$$-2 \cdot \min_i \left\{ \frac{B_i[n]}{\prod_{j \neq i} m_j \cdot D_i^2[n]} \right\} \leq K[n] \leq 0 \ . \tag{5.25}$$

The condition in Eqn. (5.25) ensures that the linearized delay feedback loops will not engage in divergent oscillations. We cannot be certain that the assumptions made to linearize the delay feedback loops hold in practice, and cannot claim that Eqn. (5.25) ensures stability of the (non-linear) delay feedback loops, but can use Eqn. (5.25) as a design guideline for $K[n]$.

5.2.4 Including the Absolute Delay and Rate Constraints

We have obtained a stability condition on $K[n]$, which is necessary to enforce proportional differentiation. So far, we have not considered the absolute delay and rate constraints in the construction of the delay feedback loops. These absolute delay and rate constraints are viewed as a "saturation constraint" on the rate adjustment, and yield a second bound on $K[n]$. To satisfy the constraints $r_i[n] \geq r_i^{\min}[n]$, we may need to clip $\Delta r_i[n]$ when the new rate is below the minimum. This, however, may violate the work-conserving property resulting from Eqn. (5.13). To respect the saturation constraint, $K[n]$ has to satisfy

$$r_i[n-1] + K[n] e_i[n] \geq r_i^{\min}[n] \ ,$$

and apply that $K[n]$ to all control loops. The above implies that we must have

$$K[n] \geq \max_i \left(\frac{r_i^{\min}[n] - r_i[n-1]}{e_i[n]} \right) \ . \tag{5.26}$$

We note that if

$$\max_i \left(\frac{r_i^{\min}[n] - r_i[n-1]}{e_i[n]} \right) > 0 \ ,$$

we cannot have $K[n] < 0$. In other words, we cannot satisfy absolute delay and throughput bounds and proportional delay differentiation at the same time. In such a case, we relax either Eqn. (5.25) or (5.26) according to the precedence order on the service differentiation given in Eqn. (4.8).

5.3 The Loss Feedback Loop

We now describe the feedback loop which controls the traffic dropped from class i to satisfy proportional loss differentiation within the limits imposed by the absolute loss bounds. As before, we assume that all classes are offered proportional loss differentiation. The assumption can be relaxed as described in the remark at the end of Section 5.2.1.

Traffic must be dropped at the n-th event either if there is a buffer overflow or if absolute delay bounds cannot be satisfied given the current backlog. With a buffer size B, to prevent buffer overflows at the n-th event, the following condition must hold:

$$\sum_{k=1}^{Q} \big(B_k[n-1] + a_k[n] - l_k[n]\big) - \Delta t[n-1]C \leq B \; . \tag{5.27}$$

To provide absolute delay and throughput bounds, the following condition must be satisfied

$$\sum_{k=1}^{Q} \max \left\{ \frac{B_k[n-1] - r_k[n-1]\Delta t[n-1] + a_k[n] - l_k[n]}{d_k - D_k[n]}, \mu_k \cdot \chi_{B_k[n]>0} \right\} \leq C \; . \tag{5.28}$$

To choose the amount of traffic to drop from each class so that Eqs. (5.27) and (5.28) hold, we define the weighted loss rate to be

$$p_i^*[n] = \left(\prod_{j=1,\ j\neq i}^{Q} m'_j \right) p_i[n] \; ,$$

where $m'_i = \prod_{j=1}^{i-1} k'_j$ for $i > 1$ and $m'_1 = 1$. With this definition, Eqn. (3.22) is equivalent to

$$\forall (i,j) \; , \forall n : p_i^*[n] = p_j^*[n] \; .$$

This condition is equivalent to

$$\forall i \; , \forall n : p_i^*[n] = \overline{p}^*[n] \; .$$

where $\overline{p}^*[n]$ is the set point for the loss feedback loop, given by

$$\overline{p}^*[n] = \frac{1}{Q}\sum_i p_i^*[n] \ .$$

We use the set point to describe an error

$$e_i'[n] = \overline{p}^*[n] - p_i^*[n] \ .$$

To reach the set point, the error is decreased by increasing $p_i^*[n]$ for classes that have $e_i'[n] > 0$ as follows. Let $\langle i_1, i_2, \ldots, i_R \rangle$ be an ordering of the class indices from all backlogged classes, that is, $B_{i_k}[n] > 0$ for $1 \le k \le R$, such that $e_{i_s}'[n] \ge e_{i_r}'[n]$ if $i_s < i_r$. Traffic is dropped in the order of $\langle i_1, i_2, \ldots, i_R \rangle$.

Absolute loss rate bounds impose an upper bound, $l_i^*[n]$, on the traffic that can be dropped at event n from class i. The value of $l_i^*[n]$ is determined from Eqs. (3.19) and (5.6) as

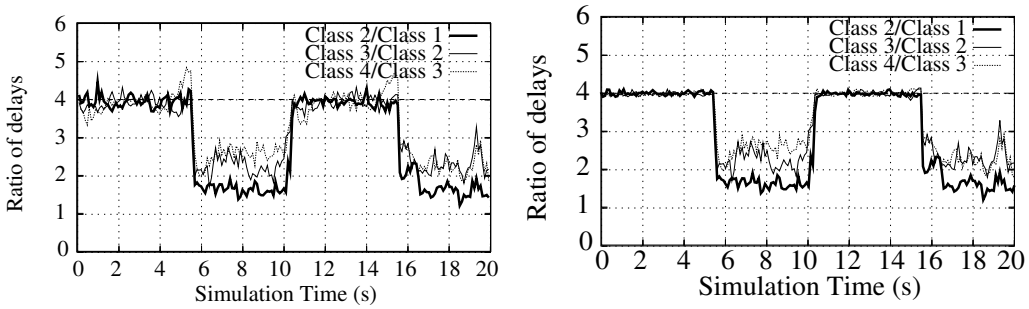
$$l_i^*[n] = A_i[n]L_i - p_i[n-1]A_i[n-1] \ .$$

If the conditions in Eqs. (5.27) and (5.28) are violated, traffic is dropped from class i_1 until the conditions are satisfied, or until the maximum amount of traffic $l_{i_1}^*[n]$ has been dropped. Then traffic is dropped from class i_2, and so forth. Suppose that the conditions in Eqs. (5.27) and (5.28) are satisfied for the first time if $l_j^*[n]$ traffic is dropped from classes $j = i_1, i_2, \ldots, i_{\widehat{k}-1}$, and $\widehat{x}[n] \le l_{\widehat{k}}^*[n]$ traffic is dropped from class $i_{\widehat{k}}$, then we obtain:

$$l_i[n] = \begin{cases} l_i^*[n] & \text{if } i = i_1, i_2, \ldots, i_{\widehat{k}-1} \ , \\ \widehat{x}[n] & \text{if } i = i_{\widehat{k}} \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{5.29}$$

If $l_k[n] = l_k^*[n]$ for all $k = i_1, i_2, \ldots, i_R$, we allow absolute delay and rate conditions to be violated. In other words, condition (5.28) is relaxed.

The loss feedback loop never increases the maximum error $e_i'[n]$, if $e_i'[n] > 0$, and more than



(a) Closed-loop algorithm (b) Optimization-based algorithm

Figure 5.4: **Experiment 1: Delay differentiation.** The graphs show the proportional delay differentiation obtained by each class at the simulated 1 Gbps bottleneck link. (b) is a duplicate of Figure 4.4(a).

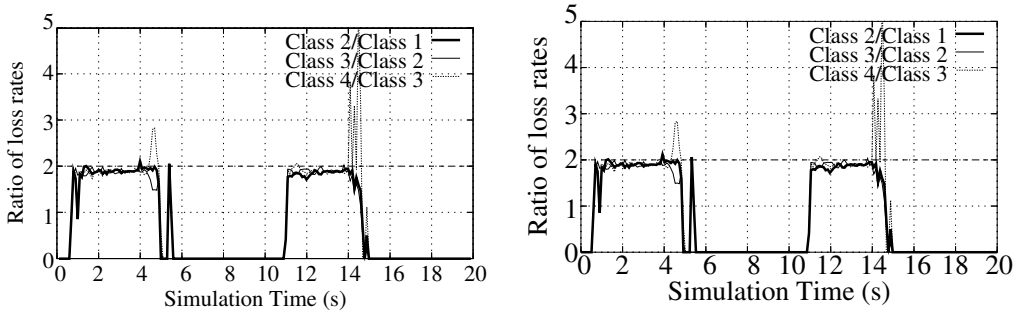
one class is backlogged. Thus, the errors remain bounded and the algorithm presented will not engage in divergent oscillations around the target value $\overline{p}^*[n]$. Additionally, the loss feedback loop and the delay feedback loops are independent of each other, since we always drop traffic from the tail of each per-class buffer, losses do not have any effect on the delays of traffic admitted into the transmission queue.

5.4 Evaluation

As in Chapter 4, we perform an evaluation by simulation of our closed-loop algorithm. We compare the performance of the closed-loop algorithm with the optimization-based algorithm from the previous chapter, using the same single-bottleneck simulation experiment as in Section 4.4. This single-bottleneck simulation experiment also allows us to assess the stability of the feedback loops used in the closed-loop algorithm by subjecting the bottleneck link to a highly variable offered load. Then, we present a multi-node network simulation experiment with a mix of TCP and UDP traffic, and examine how per-hop, per-class guarantees translate into end-to-end guarantees.

5.4.1 Simulation Experiment 1: Single Node Topology

We first use the single node simulation experiment we described in Section 4.4 to compare the performance of our closed-loop algorithm to that of the optimization-based algorithm discussed in



(a) Closed-loop algorithm (b) Optimization-based algorithm

Figure 5.5: **Experiment 1: Loss differentiation.** The graphs show the proportional loss differentiation obtained by each class at the simulated 1 Gbps bottleneck link. (b) is a duplicate of Figure 4.5(a).

the previous chapter. So, we use the same service guarantees, network topology, and traffic pattern as in Experiment 1 of Chapter 4.

We plot the ratios of delays and loss rates in Figures 5.4 and 5.5. In the graphs, each datapoint corresponds to a moving average with a sliding window of size 0.1 *s*. The results obtained for the ratios of delays in Figure 5.4(a) show that proportional delay differentiation is achieved with good accuracy when the link is overloaded. Furthermore, the plots show that the cloded-loop algorithm reacts immediately when the offered load goes from underload to overload, and reacts swiftly (between 0 and 0.2 *s* depending on the class concerned) when the link goes from overload to underload. This indicates that the delay feedback loops used in the closed-loop algorithm are stable. Proportional delay differentiation does not match the target proportional factors $k_i = 4$ when the link is underloaded, due to the fact that our algorithms are work-conserving, and therefore cannot artificially generate delays when the load is small. As illustrated in Figure 5.4(b), which is a duplicate of Figure 4.4(a), we observe the exact same behavior with the reference optimization-based algorithm from Chapter 4.

Results for ratios of loss rates in Figure 5.5(a) indicate that proportional loss differentiation is achieved when the output link buffer overflows and traffic is dropped. The transient spikes observed when the link goes from overload to underload are, as discussed in Chapter 4, due to short busy periods leading to a very small number of packet drops, which in turn, makes ratios of loss

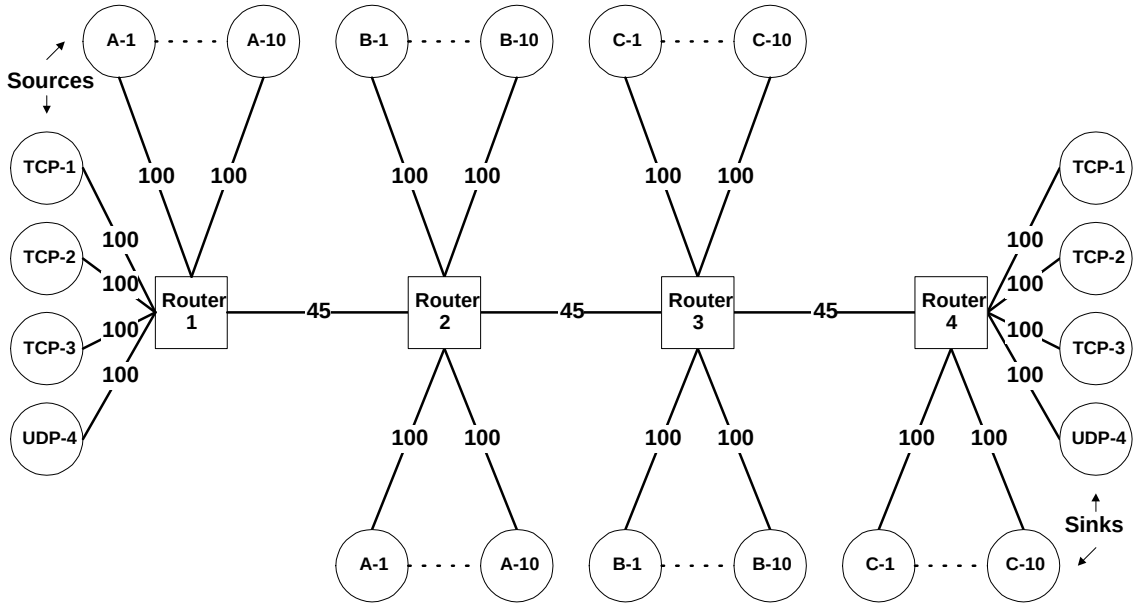


Figure 5.6: **Experiment 2: Network topology.** The labels of the links denote the links capacities in Mbps.

rates become less meaningful. Comparing Figures 5.5(a) and Figure 5.5(b), which is a duplicate of Figure 4.5(a), indicates that the loss differentiation obtained with the closed-loop algorithm is almost identical to the loss differentiation realized by the optimization-based algorithm.

From this simulation experiment, we conclude that:

- The feedback loops used in the closed-loop algorithm appear to be robust to variations in the offered load.
- The results of the closed-loop algorithm closely match those of the optimization-based algorithm from Chapter 4.

5.4.2 Simulation Experiment 2: Multiple Node Simulation with TCP and UDP Traffic

Next, we present a multi-node network simulation, and evaluate if our approach provides the desired service, in the context of a mix of TCP and UDP traffic, with multiple hops and propagation delays. We assess the *end-to-end* service seen by traffic in the presence of per-hop guarantees. To that effect,

Flow	**Class**	**Type**				
		Protocol	Traffic	On	Off	α
TCP-1	1	TCP	Greedy	N/A	N/A	N/A
TCP-2	2	TCP	Greedy	N/A	N/A	N/A
TCP-3	3	TCP	Greedy	N/A	N/A	N/A
UDP-4	4	UDP	Pareto On-off	10 *ms*	10 *ms*	1.9
A-1	1	TCP	Exponential On-off	1000 pkts	200 *ms*	N/A
A-2, A-3	2	TCP	Exponential On-off	1000 pkts	200 *ms*	N/A
A-4, A-5, A-6	3	TCP	Exponential On-off	1000 pkts	200 *ms*	N/A
A-7, A-8, A-9, A-10	4	UDP	Pareto On-off	120 *ms*	200 *ms*	1.9

Table 5.1: **Experiment 2: Traffic mix.** The traffic mix for flows B-1, ..., B-10 and C-1, ..., C10 is identical to the traffic mix described here for flows A-1,...,A-10. α is the shape parameter used in the Pareto distribution.

Class	**Service Guarantees**			
	d_i	L_i	k_i	k'_i
1	2 *ms*	0.1 %	–	–
2	–	–	4	4
3	–	–	4	4
4	–	–	N/A	N/A

Table 5.2: **Experiment 2: Service guarantees.** The guarantees are identical at each router.

we implemented our closed-loop algorithm in the *ns-2* network simulator [5]. This implementation is now included in the standard *ns-2* distribution, as of *ns*-2.26.

We simulate a network with a topology as shown in Figure 5.6. We have four routers connected by three 45 Mbps links, and sources and sinks connected to the routers by independent 100 Mbps links. Each 45 Mbps link has a propagation delay of 3 *ms*, and each 100 Mbps link has a propagation delay of 1 *ms*. There are four classes of traffic. The composition of the traffic mix is given in Table 5.1 and the service guarantees are given in Table 5.2. Traffic consists of a mix of TCP and UDP flows. TCP sources run the *TCP Reno* congestion control algorithms. TCP flows are either greedy, to model long file transfers, or on-off flows with exponentially distributed on and off periods, to model short, successive file transfers (e.g., HTTP requests). UDP flows are on-off flows using a Pareto distribution for the on and off periods. Due to the presence of on-off TCP flows, we do not need to have UDP traffic as bursty as in the experiment of the previous section, where

$\alpha = 1.2$, and we set here the shape parameter α to $\alpha = 1.9$.

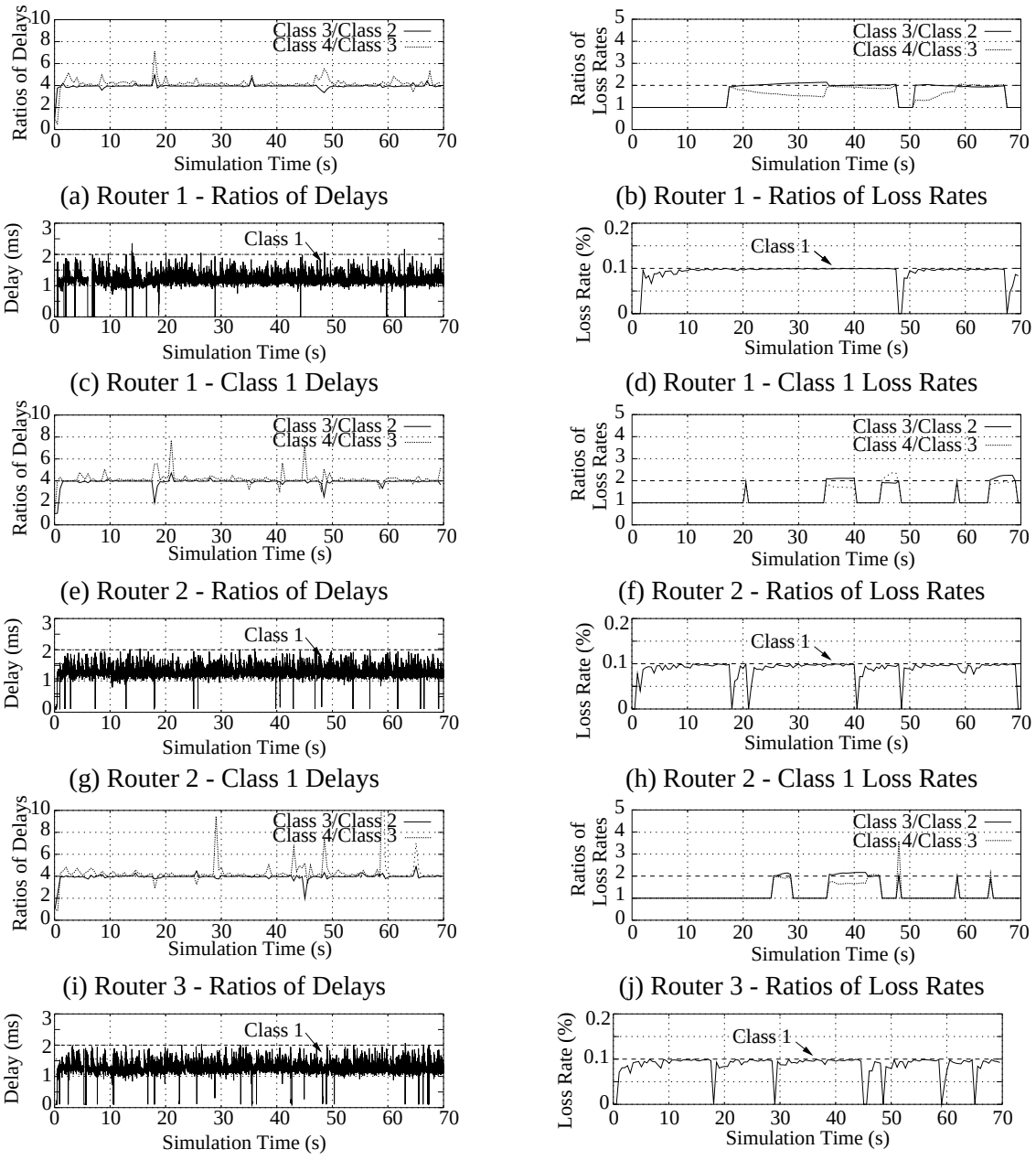
Cross-traffic flows (denoted by A-1, ..., C-10) start transmitting at time $t = 0$ *s*. The flows TCP-1, TCP-2, TCP-3 and UDP-4 start transmitting at time $t = 10$ *s*. All flows consists of packets with a fixed size of 500 bytes, and the experiment lasts 70 seconds of simulated time. The resulting load at the bottlenecks is roughly constant and equal to the capacity of the bottleneck links.

From Tables 5.1 and 5.2, Classes 1, 2 and 3 only consist of TCP traffic, and Class 4 only consists of UDP traffic. Initially Class 1 contributes 10% of the aggregate cross-traffic, Class 2 contributes 20%, Class 3 contributes 30% and Class-4 contributes 40 %. We made the choice of having almost as much UDP traffic as TCP traffic so that we could examine the effects of mixing the two types (UDP and TCP) of traffic more easily.

Per-hop per-class QoS. We graph the per-class queueing delays and per-class loss rates at each of the first three routers in Figure 5.7, starting at time $t = 0$ *s*. Given that the aggregate arrival rate at Router 4 is always less than the total output capacity of Router 4, there is never any backlog at Router 4, and thus, the queueing delays and loss rates are constantly equal to zero. With the exception of Figure 5.7(c), (g) and (k), where the individual packet delays are plotted, each point on Figure 5.7 represents an average over a sliding window of size 0.5 *s*. Figure 5.7 shows that the proposed algorithm manages to enforce all proposed service guarantees at each router, with only a couple of transient violations of the absolute delay bound on Class 1 at Router 1, and that the algorithm seems to respond appropriately to transient changes such as the introduction of additional traffic at time $t = 10$ *s*.

End-to-end per-flow QoS. Finally, we present end-to-end measurements for the flows TCP-1, TCP-2, TCP-3 and UDP-4. Each of these four flows traverses four routers, each router providing an absolute delay guarantee of 2 *ms* on Class 1. Adding to these per-node delay guarantees the propagation delays between each node, one can infer that the end-to-end delays of TCP-1 packets have to be less than $4 \times 2 + 3 \times 3 + 2 \times 1 = 19$ *ms*. Similarly, the end-to-end loss rate encountered by TCP-1 should be less than $1 - (1 - L_1)^4 \approx 0.004$, that is, 0.4%.

In Figure 5.8(a), we present the individual end-to-end packet delays encountered by each flow. The figure shows that Flow 1's end-to-end delays are indeed always below 19 *ms*, represented by



(a) Router 1 - Ratios of Delays

(b) Router 1 - Ratios of Loss Rates

(c) Router 1 - Class 1 Delays

(d) Router 1 - Class 1 Loss Rates

(e) Router 2 - Ratios of Delays

(f) Router 2 - Ratios of Loss Rates

(g) Router 2 - Class 1 Delays

(h) Router 2 - Class 1 Loss Rates

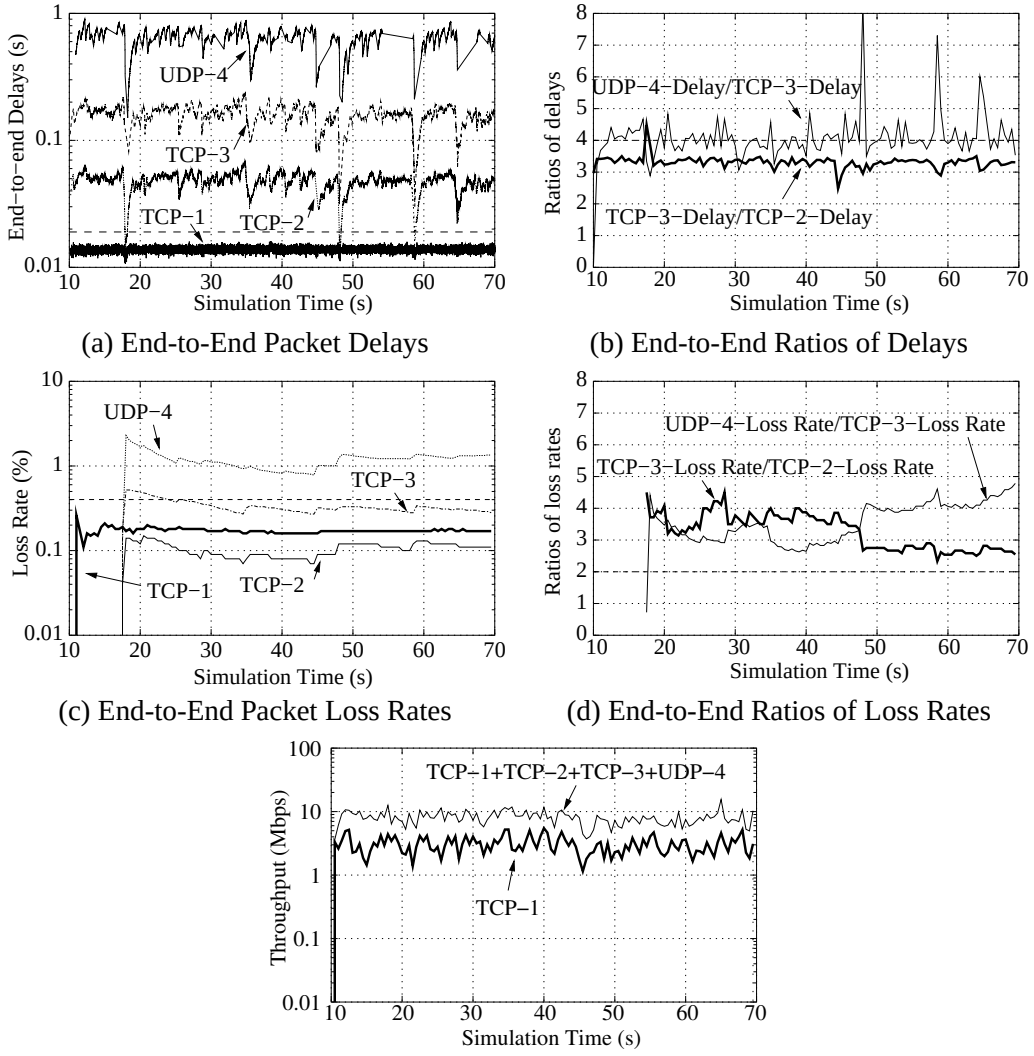
(i) Router 3 - Ratios of Delays

(j) Router 3 - Ratios of Loss Rates

(k) Router 3 - Class 1 Delays

(l) Router 3 - Class 1 Loss Rates

Figure 5.7: **Experiment 2: Multiple node simulation with TCP and UDP traffic.** The graphs show the delays and loss rates encountered at each router by Class 1 traffic, and the ratios of delays and the ratios of loss rates for Classes 2, 3 and 4 at each router. The absolute constraints and the target ratios are indicated by straight dashed lines.



(a) End-to-End Packet Delays

(b) End-to-End Ratios of Delays

(c) End-to-End Packet Loss Rates

(d) End-to-End Ratios of Loss Rates

(e) Class-1 Throughput

Figure 5.8: **Experiment 2: End-to-end packet delays.** The graphs represent the individual, end-to-end, packet delays encountered by flows TCP 1, TCP-2, TCP-3, UDP-4 (a), the ratios of delays over a sliding window of size 0.5 *s* for TCP-2, TCP-3, and UDP-4 (b), the loss rates (c), ratio of loss rates (d), and the throughput obtained by TCP-1, as well as the aggregate throughput obtained by all four flows (e). The end-to-end delay guarantee on TCP-1, indicated by a dashed line in (a), is a delay bound of 19 *ms*, the end-to-end loss guarantee on TCP-1, indicated by a dashed line in (c), is 0.4%.

the dashed line, and we see again that the algorithm we propose uses a conservative estimate of the delays for enforcing delay bounds, since most Flow 1 packets encounter a total delay close to 15 *ms*.[2] Figure 5.8(b) also suggests that the proportional delay differentiation holds with respect to the end-to-end delays between Classes 3 and 4, even if the relative delay constraints are enforced only on a per-node basis. This result can be explained by the fact that the propagation delays are negligible compared to the large queueing delays encountered by TCP-3 and UDP-4. The propagation delay cannot be neglected compared to the queueing delays in the case of the flow TCP-2, which explains why the proportional differentiation between TCP-2 and TCP-3 is close to a factor of 3.3 instead of the desired factor of 4.

We plot the end-to-end packet loss rates in Figure 5.8(c). The end-to-end loss rate bound of 0.4% on flow TCP-1, represented by a dashed line, is respected.[3] However, as shown in Figure 5.8(d), proportional guarantees on loss rates between classes do not translate into proportional guarantees between end-to-end flows: loss rates ratios between flows TCP-2, TCP-3, and UDP-4 are consistently above the desired ratios $k'_2 = k'_3 = 2$, even though the per-hop, per-class loss guarantees are consistently respected. This result confirms that per-class guarantees do not necessarily translate into per-flow guarantees. Here, a discrepancy between per-flow differentiation and per-class guarantees is exhibited because the different flows present in the network do not have the same probability of suffering packet drops, since some of them are greedy flows, while others are on-off flows. The result indicates that additional mechanisms identifying greedy flows are needed to provide a better match between per-class guarantees and per-flow differentiation. Describing mechanisms to identify greedy flows and using these mechanisms to improve per-flow differentiation will be the object of Chapter 7.

Last, in Figure 5.8(e), we graph the throughput received by flow TCP-1, as well as the aggregate throughput received by flows TCP-1, TCP-2, TCP-3 and UDP-4. In this experiment, there is no throughput guarantee on any class, but we observe that the flow TCP-1 consistently gets an end-

[2]The queueing delays at Router 4 are always zero due to the topology we use. If we ignore the guarantees offered at Router 4 in the computation of the end-to-end delay guarantee, the end-to-end delay guarantee becomes 16 *ms*, and is respected as well.

[3]Due to the topology we chose, the loss rate at Router 4 is always zero. If we ignore the guarantees offered at Router 4 in the computation of the end-to-end loss guarantee, we obtain an end-to-end loss bound of $1-(1-L_1)^3 \approx 0.003$, that is, 0.3%, which is also respected.

to-end throughput greater than 1 Mbps, and close to 3 Mbps in general. This result shows that the absolute guarantees on the loss and delay of Class 1 are not realized at the expense of a low throughput. For readability purposes, we do not show the throughput plots for the three other flows, but mention that they present values close to that of TCP-1.

As a conclusion to this second experiment, we showed that our algorithm was able to provide the desired per-class, per-node service guarantees in a multiple node simulation, with a mix of TCP and UDP traffic. We also showed how these per-class, per-node service guarantees could translate into end-to-end, per-flow performance.

5.5 Summary and Remarks

We proposed a closed-loop algorithm for realizing the Quantitative Assured Forwarding service at a router. The delays and losses experienced by classes are monitored, which allows the algorithm to infer a deviation compared to the expected service differentiation. The algorithm then adjusts service rate allocation and the drop rates to attenuate the difference between the service experienced and the service guarantees.

We used linear control techniques in the design of the algorithm. In particular, we proposed to use a proportional control to achieve proportional delay differentiation. Absolute differentiation is expressed in terms of saturation constraints that limit the range of the controller. We linearized the control loop around an operating point, and derived a stability condition on the linearized control loop. While the stability condition derived does not ensure that the non-linear control loop converges, the stability condition gives useful guidelines for selecting the configuration parameter of the controller.

Simulation results indicate that the proposed closed-loop algorithm is an effective approximation of the optimization-based algorithm, and that the feedback control is stable. Additionally, we described the effect of per-hop service differentiation on end-to-end service guarantees.

Chapter 6

Implementation

In this chapter, we present the design of a configurable router that provides the service guarantees of Quantitative Assured Forwarding in IP networks such as the Internet [32]. We call such an IP router a "QoSbox". Using a QoSbox at a bottleneck link, the network operator only needs to specify the service guarantees that are desired at the bottleneck link, by means of a configuration file, and turn on the QoSbox to obtain the desired QoS. The central mechanism used in the QoSbox is the closed-loop algorithm for buffer management and service rate allocation discussed in Chapter 5.

We discuss our implementation of the QoSbox in PC-routers running an operating system from the BSD family, i.e., FreeBSD [1], NetBSD [4] or OpenBSD [6]. The implementation we describe is now part of the base distributions of the KAME [3] and ALTQ-3.1 [30] packages. Inclusion in the base BSD distributions as part of ALTQ is currently under consideration.

We use the implementation of the QoSbox in PC-routers to demonstrate that Quantitative Assured Forwarding can be realized in packet networks with links of speeds in the order of a few hundred megabits per second. We also outline how the closed-loop algorithm can be further modified to be implemented in switch architectures at higher line speeds, at the expense of some reduced accuracy in the service differentiation.

This chapter is organized as follows. In Section 6.1, we present an overview of the implementation. In Section 6.2, we discuss the implementation details. In Section 6.3, we present a numerical evaluation using a testbed of PC-routers. The evaluation illustrates how service differentiation is

realized, and assesses the overhead of the associated mechanisms. In Section 6.4, we briefly review the related work on publicly available implementations of service architectures for PC-routers.

6.1 Implementation Overview

In this section, we present an overview of the implementation. Recall from the discussion in Chapter 3, that Quantitative Assured Forwarding imposes that there be no communication between different routers, admission control or policing of traffic. Hence, realizing the QAF service in an IP network only requires to implement our algorithms for service differentiation in the data path of IP processing at QoSboxes.

The QoSbox is an output queueing architecture. On the other hand, the PC-routers that we use for our implementation rely on a shared memory architecture, with input and output queues, and are therefore CIOQ routers. From the discussion in [111], input queues in a PC-router are empty when the CPU of the PC-router is not overloaded. So, barring CPU overloads, PC-routers are equivalent to output queueing architectures. From now on, we assume that traffic control is only performed at the output queues and solely focus on the operations performed at the output queues. We will show that the overhead associated to our proposed mechanisms is limited enough so that we should not face CPU overload conditions in typical access networks where our implementation in PC-routers can be deployed, and that, as a result, the output queueing assumption is justified.

In the remainder of this section, we discuss how service guarantees are configured in a QoSbox. Then, we present an overview of the mechanisms in place in the transmission queues at the output link(s) governed by a QoSbox.

6.1.1 Configuration of the Service Guarantees

Network interfaces in a QoSbox are configured with a configuration file. The structure and syntax of the configuration file is based on the syntax of an ALTQ-3.1 configuration file [52]. Figure 6.1 is an example of a QoSbox configuration file. The configuration file defines the properties of the output interface(s), the guarantees each class of traffic receives and the filters used by the classifier to map

```
(1) interface fxp0 bandwidth 100M qlimit 200 jobs
(2) class jobs fxp0 high_class NULL priority 0\
    adc 2000 rdc -1 alc 0.01 rlc -1 arc 10M
(3) class jobs fxp0 med2_class NULL priority 1\
    adc -1 rdc 2 alc -1 rlc 2 arc -1
(4) class jobs fxp0 med1_class NULL priority 2\
    adc -1 rdc 2 alc -1 rlc 2 arc -1
(5) class jobs fxp0 low_class NULL priority 3 default\
    adc -1 rdc -1 alc -1 rlc -1 arc -1
(6) filter fxp0 high_class 0 0 0 0 0 tos 1
(7) filter fxp0 med2_class 0 0 0 0 0 tos 2
(8) filter fxp0 med1_class 0 0 0 0 0 tos 3
(9) filter fxp0 low_class  0 0 0 0 0 tos 4
```

Figure 6.1: **Example of a QoSbox configuration file.** The configuration file defines (1) the properties of the output interface, (2) the guarantees each class of traffic receives and (3) the filters used by the classifier to map packets to given classes of traffic. Line numbers are not part of the configuration file, but are used here for readability purposes.

packets to given classes of traffic. In the example of Figure 6.1, the interface concerned, `fxp0`, has a bandwidth of 100 Mbps and a buffer size set to 200 packets. The field `jobs` indicates that traffic control at this interface relies on the JoBS scheme discussed in Chapter 3, and, in particular, on the closed-loop algorithm presented in Chapter 5.

The next set of configuration commands, in lines (2)–(5), contains the service guarantees offered to each class. In the example of Figure 6.1, the class `high_class` is given a delay guarantee, indicated by the keyword `adc`,[1] of 2000 microseconds, no proportional delay differentiation, as specified by the field `rdc -1`, a loss bound of 1%, configured by the command `alc 0.01`, no proportional loss differentiation (`rlc -1`) and a guaranteed throughput of 10 Mbps (`arc 10M`). The `priority` field simply indicates a class index, but does not denote a priority order. Classes `med1_class`, `med2_class`, `med3_class` are not offered delay, loss, or throughput bounds, but are subject to proportional delay and loss differentiation.

The `rdc` and `rlc` keywords specify the proportional differentiation desired between the class to which they are applied, and the class denoted by the following class index. In the example of

[1] Even though the QoSbox uses an implementation of the closed-loop algorithm, the keywords are based on the names of the QoS constraints in the optimization-based algorithm.

Figure 6.1, since the class `med2_class` with class index 1 is given an `rdc` factor of 2, packets which belong to the class with the class index 2, i.e., `med1_class`, should get queueing delays twice as long as those experienced by `med2_class` packets. Note that the parameters taken by `rdc` or `rlc` can be any positive value, including values less than one.

The commands in lines (6)–(9) describe the mapping from packet headers to service classes. In this example, the only classification criterion is the Type-of-Service (TOS) field of the IP header, recently renamed DiffServ Codepoint (DSCP, [114]). In the present example, a value of 0x03 in the DSCP field of an incoming packet indicates that the packet belongs to the class `med1_class`. The configuration file presented above assumes the use of IP version 4 (IPv4, [122]), but the implementation of the QoSbox presented in this chapter also supports IP version 6 (IPv6, [44]). In the case of IPv6, the DSCP corresponds to the IPv6 Traffic Class octet.

The example of Figure 6.1 assumes that marking of the DSCP field is performed upstream, for instance by the applications at the end hosts. To avoid maintaining per-flow information in QoSboxes, we advocate that marking should not be performed by QoSboxes. We note however that ALTQ provides per-flow marking primitives, using IP Filter [125],[2] and that a network operator could configure QoSboxes to mark packets based on source/destination pairs.

6.1.2 Mechanisms

All output queues in the QoSbox have the same architecture, which is outlined in Figure 6.2. Each class of traffic is associated with a FIFO per-class buffer. When a packet is passed to a network interface a classifier looks up which class the packet belongs to, and places the packet in the appropriate per-class FIFO buffer. The classifier does not identify the flow to which the incoming packet belongs. The per-class buffers have a finite size selected by the network operator as follows. The maximum amount of traffic that can be held in each per-class buffer can be fixed to a constant (*separate buffers*), or, alternatively, the maximum total amount of traffic backlogged can be bounded (*shared buffer*).

[2] Future versions of ALTQ are likely to instead use the `pf` packet filter [74].

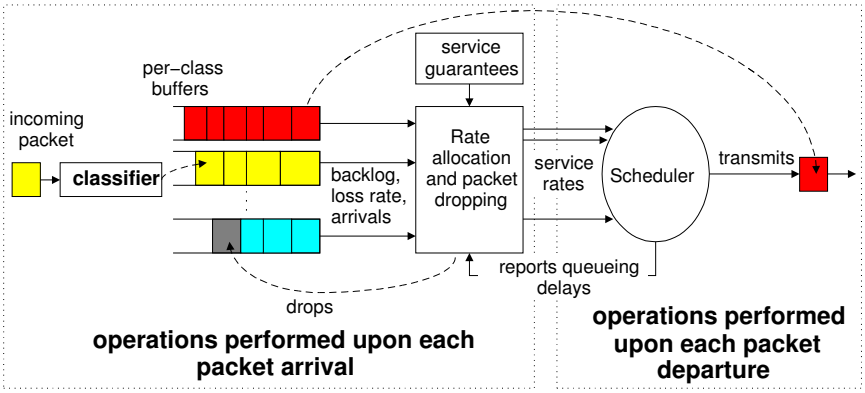


Figure 6.2: **Architecture of an output queue in the QoSbox.** The three main components are the packet classifier, in charge of storing incoming packets in the proper FIFO per-class buffer, the rate allocation and dropping algorithm, and the scheduler, which forwards packets according to the service rates allocated.

After the incoming packet has been placed in a per-class buffer, the closed-loop algorithm adjusts the service rates allocated to each class of traffic and possibly drops packets in order to enforce the desired service guarantees. The computation of the service rates and packet drops is based on the current backlog, arrivals, loss rate on the one hand, and on queueing delays reported by the scheduler on the other hand. If needed, packets are dropped from the tail of each per-class buffer.

The service rates calculated by the rate allocation algorithm must be translated into packet forwarding decisions, which is the task of the packet scheduler. Schedulers translating service rates into packet forwarding decisions, such as Packetized-GPS [119] or Virtual Clock [157], have been proposed in the early 1990s. These schedulers have provable worst-case delay bounds, but require dynamic sorting of the packets backlogged in the system, and have a worst-case complexity of $O(N)$ where N is the number of packets backlogged in the system. We propose a heuristic, inspired by the Deficit-Round Robin algorithm [135], that avoids packet sorting. The heuristic has an $O(Q)$ complexity, where Q is the number of classes in the system.

A variable recording the amount of traffic that has been sent in each class since the beginning of the current busy period, $Xmit_i$, is maintained by the scheduler. The output curve, R_i^{out}, is updated

every time a packet enters the output queue, with

$$R_i^{out} \leftarrow R_i^{out} + r_i \cdot \Delta t \ , \tag{6.1}$$

where Δt corresponds to the amount of time that has elapsed since the last update of R_i^{out}. In other words, the output curve R_i^{out} corresponds to the amount of traffic that would have been transmitted since the beginning of the current busy period if packet scheduling perfectly matched the service rate allocation, as is the case in the fluid-flow model we used in the previous chapters. Every time the output link is available for transmission of a packet, the scheduler computes, for each class, the difference $R_i^{out} - Xmit_i$. Denoting by k the index of the class for which this difference is maximum, meaning that class k is the "most behind" its allocated service rate, the scheduler chooses to transmit the packet at the head of the class-k buffer, and records the queueing delay experienced by the transmitted packet, by taking the difference between the current time and the time this packet was enqueued.

6.2 Implementation Details

Next, we describe the details of our implementation. We first focus on the specifics of the implementation we carried out for BSD kernels using the Alternate Queuing framework (ALTQ, [30]). We provide a short review of ALTQ, and then turn to a discussion of the operations performed by our implementation. In our PC-router implementation, all operations are sequential. Even without exploiting potential parallelism in the algorithm, we will show in Section 6.3 that our implementation can operate at line speeds in the order of 100-500 Mbps in a 1 GHz PC if the number of classes is small.

6.2.1 ALTQ

Our implementation of the QoSbox for PC-routers builds on ALTQ. ALTQ is an extension to the FreeBSD, OpenBSD and NetBSD operating system kernels. In addition to various bug fixes to

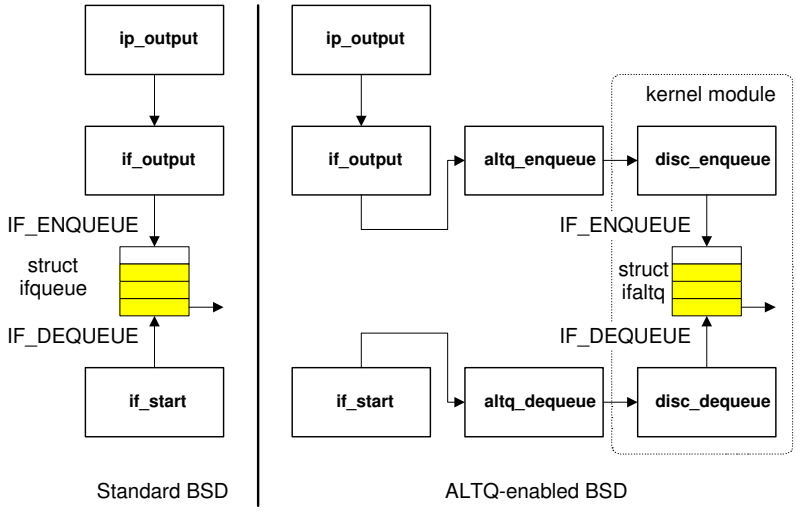


Figure 6.3: **Functions and structures associated with the output queue in BSD and ALTQ-enabled BSD.** Each queueing discipline implemented in ALTQ consists of a kernel module, represented by the dotted box.

networking device drivers, ALTQ provides a modular framework for replacing the default FIFO queueing discipline of network interfaces by custom-designed queueing disciplines.

In BSD kernels, an output networking interface is governed by the `if_output` and `if_start` functions, which enqueue and dequeue packets from the transmission queue, respectively. The transmission queue is represented by the `ifqueue` structure and is shown on the left in Figure 6.3. An incoming packet is passed to `ip_output` which, after looking up the route, filling the IP header, and possibly fragmenting the packet passes it to `if_output`; `if_output` enqueues the packet in the `ifqueue` structure. When the output link is available for transmission, a packet is dequeued from the `ifqueue` structure by the `if_start` function.

As shown on the right in Figure 6.3, ALTQ replaces the operations performed by `if_output` and `if_start` by user-defined transmission queue structures and functions included in dynamically loadable kernel modules. Each kernel module implements a specific queueing discipline. A custom packet queue structure (`struct ifaltq`) is used as a replacement to the `ifqueue` structure to implement the transmission queue. Transmission queue management is realized by enqueue and dequeue functions specific to each queueing discipline, as denoted by `disc_enqueue`

and `disc_dequeue` in the figure. For instance, the enqueue and dequeue functions of our the QoSbox are called `jobs_enqueue` and `jobs_dequeue`, respectively. Queueing disciplines that have been developed for ALTQ include Class-Based Queueing (CBQ, [69]), Hierarchical Fair Service Curve (HFSC, [144]), Random Early Detection (RED, [68]) and Blue [60].

Additionally, ALTQ provides a classifier that is used to map incoming packets to classes of traffic. In the QoSbox, classification only consists of selecting in which buffer to store the packet by looking up a class index contained in the DSCP, as marked upstream, and adding a priority field tag to the packet buffer. In a BSD implementation, packets are stored in memory buffers called `mbuf`'s. The `mbuf` structure can be modified to record information, such as the priority field tag we need to insert, in addition to the packet header and payload

We refer to [30] for more details on the implementation of ALTQ.

6.2.2 Packet Processing

All mechanisms specific to the QoSbox are realized by `jobs_dequeue` and `jobs_dequeue`. We next describe both functions in detail.

The `jobs_dequeue` function implements the packet scheduler described in the last paragraph of Section 6.1.2. The `jobs_dequeue` function has knows the value of the variables R_i^{out} and $Xmit_i$ for each class, finds the class for which the difference $R_i^{out} - Xmit_i$ is maximized, and dequeues the packet located at the head of the corresponding per-class buffer. These operations consist of an integer subtraction and an integer comparison per class, and the actual packet dequeueing. There is a total of $2Q$ arithmetic operations and $2Q+1$ memory accesses, one for reading each of the variables R_i^{out} and $Xmit_i$, and one for accessing the packet to be dequeued.

The processing overhead in the QoSbox is caused by the operations occurring during the enqueueing of an incoming packet, that is, the operations performed by the rate allocation and packet dropping algorithm. This algorithm is implemented by the `jobs_enqueue` function in ALTQ and relies on several arithmetic operations (e.g., computation of K). In a network simulator, these operations can be performed using double precision floating-point numbers. In the case of a kernel-level implementation, floating-point operations should be avoided, because the hardware floating-point

```
(1)  function jobs_enqueue(pkt)
(2)      if (output_link_is_idle())
(3)          reset_all_variables();
(4)          transmit(pkt);
(5)      else
(6)          insert_tail(pkt);
(7)          if (not_backlogged_anymore() or now_backlogged())
(8)              reset_rates();
(9)          while (buffer_overflow())
(10)             i = select_dropped_class();
(11)             drop(i);
(12)         accounting(pkt);
(13)         compute_min_rates();
(14)         while (∑min_rates > C and can_drop())
(15)             i = select_dropped_class();
(16)             drop(i);
(17)             compute_min_rates();
(18)         adjust_rates();
(19) return (dropped);
```

Figure 6.4: **Rate allocation and packet dropping in the QoSbox.** This sequence of operations is performed immediately after a packet arriving at the output queue has been classified. Line numbers are printed for readability purposes.

unit (FPU) is generally not supported in the kernel, and floating-point operations using the FPU emulation library are extremely slow.

The computations in `jobs_enqueue` only use fixed-point arithmetic. In our implementation, all quantities are expressed using 64-bit unsigned integers, which requires to adopt some specific units. Delays are expressed in clock ticks, service rates are expressed in bytes per clock tick scaled by a factor of 2^{32}, and loss rates are expressed as fractions of 2^{32}. These units can achieve a satisfactory degree of precision.

Pseudo-code for the `jobs_enqueue` function, which returns a boolean indicating whether packets have been dropped, is presented in Figure 6.4. The first set of operations, in lines (2)–(4), are executed when a new busy period starts. A memory access to check the status of the output link is followed by a reset, for each class, of all four variables corresponding to the arrival curve, the input curve, the output curve and the transmissions $Xmit_i$. The current time is recorded as the start of a busy period, and all variables of the class corresponding to the incoming packet are subsequently set

to the size of the incoming packet. This requires $4(Q+1)+1$ assignments and arithmetic operations before transmitting the packet.

If the output link is not idle, the incoming packet is added to a per-class buffer, based on the value of its priority field tag. The `insert_tail` operation stores the arrival time of the packet in a timestamp, which is inserted at the tail of a timestamp list.[3] To record the arrival time, `jobs_enqueue` needs to access the CPU clock. A simple solution would be to use the `microtime()` function provided in BSD, which has a microsecond granularity. Using `microtime()` increases portability of the implementation, because all BSD systems implement the `microtime()` function since 4.4-BSD. However, `microtime()` may not have a fine enough granularity, and requires a periodic adjustment to account for possible clock skews. Also, using `microtime()` generates significant overhead.[4] A more efficient solution is to directly read the timestamp counter (TSC) register available in the Pentium series processors [81], and compatible architectures, such as AMD processors. This register is an unsigned 64-bit precision integer, and gives the number of cycles elapsed since the machine has been turned on. The resolution of the TSC register is much finer than that provided by `microtime()`. A similar counter (processor cycle counter, PCC) can be found on DEC Alpha architectures, but only provides a 32-bit precision [40]. We read the TSC or PCC registers if they are available, and if not, roll back to `microtime()` to ensure portability of our implementation. Despite potential variations in the duration of each clock cycle in recent processors, due for instance to power management, reading a cycle counter provides time measurements accurate enough for our implementation.

Once the packet has been added to its per-class buffer, the test in lines (7) and (8) in Figure 6.4 checks if the set of classes with a backlog in the per-class buffers has changed. If there is a change, service rates of all classes are reset: classes with no backlog get a service rate of zero, while backlogged classes equally share the capacity of the output link. In our implementation, there are Q tests to check which classes are backlogged, and, if a change in the backlogged classes is detected, the `reset_rates` call consists of Q assignments.

[3] One could also use an extra tag added to the `mbuf` holding the packet to mark its arrival time.

[4] As of 4.4-BSD, a `nanotime()` function is also available. `nanotime()` provides nanosecond granularity, but suffers the same overhead and clock skew adjustment problems as `microtime()`.

Next, in lines (9)–(11), `jobs_enqueue` drops packets in case a buffer overflow occurs. The `while` loop in lines (10) and (11) is executed in the worst-case for each backlogged packet. However, since this while loop is executed upon each packet arrival, the number of iterations is in fact bounded by the number of packets that have arrived since the last packet arrival. In a PC-router, it is extremely rare, if not impossible, in practice, to have simultaneous packet arrivals. In other switch architectures, the number of iterations is bounded by the internal speed-up of the switch, which is generally less than the number of line cards in the switch, and remains a relatively small number.

The overhead of the function `select_dropped_class` in line (10) depends on the type of loss guarantees offered to the incoming packet(s). If only absolute loss guarantees are offered, `select_dropped_class` checks that dropping the incoming packet(s) does not violate loss guarantees. This requires looking up the incoming packet size, and computing the value for the loss rate if the incoming packet is dropped, which can be done with one addition and one integer division. If no class can be dropped without violating a loss bound, the addition and division are performed for each class, for a total of $2Q$ operations. Then, `select_dropped_class` has to find the class for which the violation is minimal. This requires another Q subtractions and comparisons. So, the worst-case corresponds to a total of $3Q$ arithmetic operations and Q comparisons. The `select_dropped_class` function implements the ordering $\langle i_1, i_2, \ldots, i_R \rangle$ on classes indices provided by the loss feedback loop for proportional loss differentiation, as discussed in Chapter 5. Determining this ordering requires $4Q+1$ arithmetic operations and Q comparisons. If both proportional loss differentiation and a loss rate bound are offered to all classes, the worst-case overhead amounts to a total of $(4Q+1+Q)+3Q+Q = 9Q+1$ operations.

After buffer overflows are resolved, `jobs_enqueue` executes, in line (12), the accounting operations that take place upon each packet arrival. In addition to increasing the arrival curve and possibly the input curve if no packet is dropped, the function also updates the output curve R_i^{out} using Eqn. (6.1). This requires one memory access, and two arithmetic operations.

The computation in line (13) determines the service rates $r_i^{\min}$ needed for meeting the delay bounds and minimum throughput guarantees. For each class i, the computation of $r_i^{\min}$, given by Eqn. (5.4), involves looking up the timestamp of the head-of-the-line packet in the per-class

buffer and reading the TSC register, to obtain D_i, performing a subtraction ($d_i - D_i$) and an integer division ($\frac{B_i}{d_i - D_i}$). These operations are augmented by a comparison in case a throughput guarantee is also present. In case a delay bound violation has occurred (i.e., for a class i, $d_i - D_i \leq 0$ and $B_i > 0$), the minimum rate is set to C, the capacity of the output link. Here, the entire class-i backlog is transmitted as soon as possible in an effort to resolve the situation in a timely manner. Non-backlogged classes are assigned a minimum service rate of zero.

The `while` loop in lines (14)–(17) drops packets until a feasible rate allocation exists or all loss rate bounds are reached. The relaxation order of Eqn. (4.8) imposes that loss rate bounds have higher precedence than delay bounds, which is enforced by the `can_drop()` test in line (14). Note that all operations in the `while` loop in lines (14)–(17), including the functions `select_dropped_class` and `compute_min_rates`, can be executed once for each backlogged packet in the worst case. To reduce the total number of operations, we propose to replace lines (14)–(17) by a call to a function called `greedy_alloc`. The function `greedy_alloc` implements the greedy service rate allocation used by the heuristic algorithm proposed in Chapter 4 in the case of an ADC violation. The reduced complexity offered by the function `greedy_alloc` comes at at the expense of a potential relaxation of proportional loss guarantees when packets have to be dropped to meet delay bounds. The `greedy_alloc` uses Q arithmetic operations to redistribute the service rates. Most of the overhead in `greedy_alloc` comes from the number of operations required when dropping packets. Each time a packet is dropped, a memory access is performed, and four arithmetic operations are used to update the variables R_i^{in}, R_i^{out} and $Xmit_i$. In the worst case, all backlogged packets are dropped.

The last step, described in line (18), adjusts the service rates subject to proportional delay differentiation. The function `adjust_rates` implements the controller of the delay feedback loop, and computes the parameter K, common to all classes. To compute K, `adjust_rates` needs to compute the e_i's, the D_i^*'s, and $\overline{D}^*$. In the worst case, computing K requires $Q(Q-1)$ multiplications to compute $\prod_{j \neq i} m_j$ for all i.[5] With the values for $\prod_{j \neq i} m_j$, one needs another Q multiplications to compute D_i^* for all i. Then, one needs $Q-1$ additions and one integer division to compute $\overline{D}^*$, and

[5] This computation is only required when there is a change in the classes backlogged.

Q subtractions to compute e_i for all i. The condition for K given in Eqn. (5.25) then requires Q multiplications and Q integer divisions, followed by Q comparisons. Implementing the condition for K in Eqn. (5.26) requires Q subtractions, Q integer divisions, and Q comparisons. There is an additional comparison to check if both conditions (5.25) and (5.26) can be met at the same time. All in all, the computation of K requires $Q^2 + 5Q$ arithmetic operations in the worst case.

6.2.3 Overhead Reduction

We have described our implementation in PC-routers running a BSD kernel. We next describe how the overhead of our implementation can be reduced for higher performance switches.

The overhead of our implementation is primarily caused by three operations in the `jobs_enqueue` function: the adjustment of the service rates for proportional differentiation, performed by the call to the function `adjust_rates`, the computation of the minimum service rates required for meeting delay bounds, implemented by the call to `compute_min_rates`, and the reallocation of service rates and packet drops for meeting delay bounds, realized by the call to `greedy_alloc`.

An option to reduce the computational overhead is to reduce the frequency at which `adjust_rates`, `compute_min_rates`, and `greedy_alloc` are called. This comes at the expense of degraded performance with respect to QoS guarantees. The degradation in performance may remain acceptable for high sampling frequencies. For instance, recall from the evaluation in Chapter 4, that updating the service rate allocation every T arrivals, with T in the order of 10–100, managed to achieve almost the same results as adjusting the service rates upon each packet arrival.

When the calls to `adjust_rates`, `compute_min_rates`, and `greedy_alloc` are performed only every T arrivals, we can split `jobs_enqueue` into operations that have to be performed on a per-packet basis, that is, lines (1)–(12) in Figure 6.4, from sampled operations, that is, lines (13)–(18). Because no per-packet operation follows a sampled operation, sampled operations can be delegated to a co-processor. As a practical example, in an architecture such as the Intel IXP 1200 network processor [2], sampled operations such as `adjust_rates` could be performed on the StrongARM processor, whereas per-packet operations should be performed by the micro-engines.

In addition to sampling some operations, simplifications to the arithmetic operations involved can be carried out at the expense of flexibility. For instance, one may want to restrict the proportional differentiation factors to powers of two, so that all the multiplications used for proportional differentiation can be replaced by bit-shifting operations. One may also consider fixed sampling intervals for computing loss rates, instead of the current busy period, to avoid integer divisions in the loss rate computations. Last, we use byte counters in our implementation in PC-routers, but one may instead elect to use packet counters, which require increment/decrement operations instead of additions/subtractions for accounting.

6.3 Evaluation

We present measurement experiments of our QoSbox implementation in ALTQ in a testbed of PC routers. The PCs are Dell PowerEdge 1550 with 1 GHz Intel Pentium-III processors and 256 MB of RAM. The system software is FreeBSD 4.3 and ALTQ 3.0. Each system is equipped with five 100 Mbps-Ethernet interfaces.

In our experiments, we determine if the QoSbox provides the desired service differentiation on a per-node basis, through two experiments with different traffic mixes. We also evaluate the overhead associated to the `jobs_enqueue` and `jobs_dequeue` operations of the QoSbox.

The objective is to show that the implementation of the QoSbox in BSD-based PC-routers is a solution that can be readily deployed for service differentiation in medium-speed access networks with capacities in the order of a few hundreds megabits per seconds, and to confirm that the operations responsible for most of the overhead are those that can be performed as background tasks in higher performance architectures.

6.3.1 Testbed Experiment 1: Near-Constant Load

We use a local network topology using point-to-point Ethernet links as shown in Figure 6.5. All links are full-duplex and have a capacity of $C = 100$ Mbps. Three PCs are set up as routers, indicated in Figure 6.5 as Router 1, 2 and 3. Other PCs are acting as sources and sinks of traffic. The topology

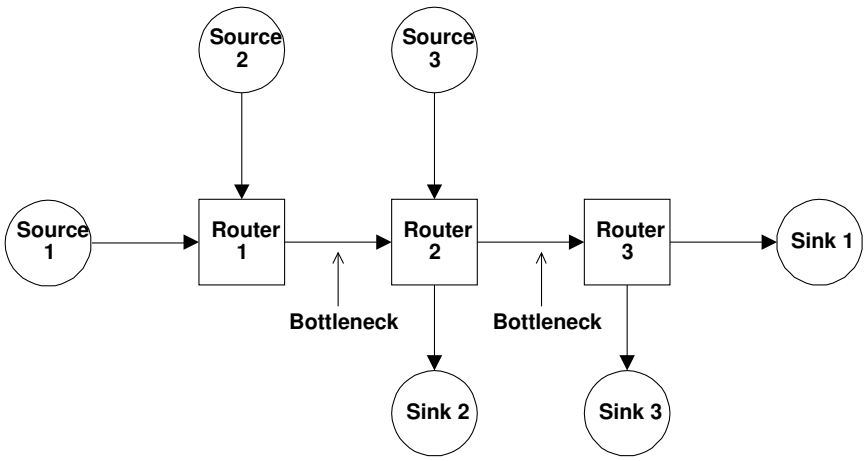


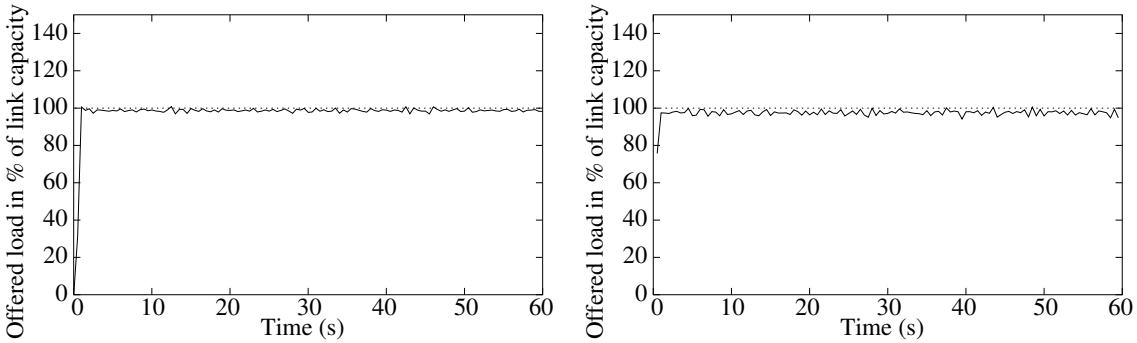
Figure 6.5: **Experiments 1 and 2: Network topology.** All links have a capacity of 100 Mbps. We measure the service provided by Router 1 and 2 at the indicated bottleneck links.

Class	**Service Guarantees**				
	d_i	L_i	μ_i	k_i	k_i'
1	8 *ms*	1 %	–	–	–
2	–	–	35 Mbps	2	2
3	–	–	–	2	2
4	–	–	–	N/A	N/A

Table 6.1: **Service guarantees.** The guarantees are identical at each router.

Class	**No. of flows**	**Type**	
		Protocol	Traffic
1	6	UDP	On-off
2	6	TCP	Greedy
3	6	TCP	Greedy
4	6	TCP	Greedy

Table 6.2: **Experiment 1: Traffic mix.** The traffic mix is identical for each source-sink pair. The on-off UDP sources send bursts of 20 packets during an on-period, and have a 150 *ms* off-period. All TCP sources are greedy, i.e., they always have data to transmit, and run the *NewReno* congestion control algorithm.



(a) Router 1. (b) Router 2.

Figure 6.6: **Experiment 1: Offered load.** The graphs show the offered load at Routers 1 and 2.

has two bottlenecks: the link between Routers 1 and 2, and the link between Routers 2 and 3. The buffer size at the output link of each router is set to $B = 200$ packets.

We consider four traffic classes with service guarantees as summarized in Table 6.1. The traffic mix, the number of flows per class, and the characterization of the flows for each source is as shown in Table 6.2. Class 1 traffic consists of on-off UDP flows, and the other classes consist of greedy TCP flows. All sources start transmitting packets with a fixed size of 1024 bytes at time $t = 0$ until the end of the experiments at $t = 60$ seconds.

Sources 1, 2 and 3 send traffic to Sinks 1, 2 and 3, respectively. The traffic mix, the number of flows per class, and the characterization of the flows, is identical for each source, and as shown in Table 6.2. Each source transmits six flows from each of the classes. Class 1 traffic consists of on-off UDP flows, and the other classes consist of greedy TCP flows. All sources start transmitting packets with a fixed size of 1024 bytes at time $t = 0$ until the end of the experiments at $t = 60$ seconds.

Traffic is generated using the *netperf* v2.1pl3 tool [90]. The network load overloads the bottleneck links of Figure 6.5. Congestion control at the TCP sources maintains the total load at a level of about 99% of the link capacity at Router 1 and Router 2, as shown in Figure 6.6.

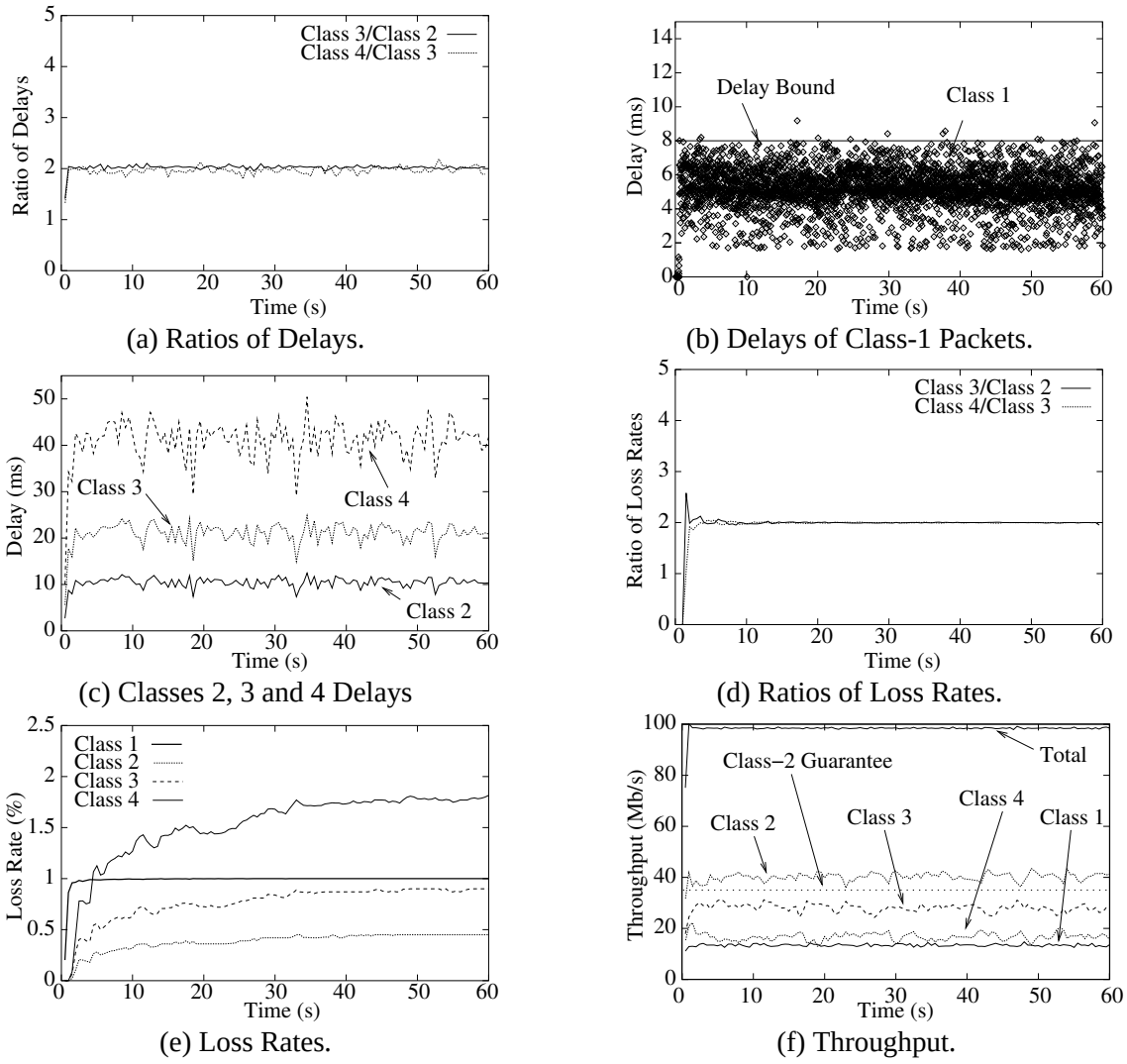
In Figures 6.7 and 6.8, we present our measurements of the service received at the bottleneck links of Routers 1 and 2, respectively. All datapoints correspond to moving averages over sliding windows of size 0.5 *s*, except in Figures 6.7(b) and 6.8(b), which presents the delays of each class-1 packet.

Figures 6.7(a) and 6.8(a) depict the ratios of the delays of classes 4 and 3, and the delays of classes 3 and 2. The plots show that the target value of $k = 2$ (from Table 6.1) is achieved.

In Figures 6.7(b) and 6.8(b) we show the delay of class-1 packets at Router 1 and Router 2. The delay bound of $d_1 = 8$ *ms* is satisfied, with few ($< 1.5\%$) violations. The violations occur due to the precedence order we chose for our absolute guarantees in Eqn. (4.8), that is, in case of an infeasible set of service guarantees, absolute delay guarantees are relaxed in favor of absolute loss guarantees. No class-1 packet ever experiences a delay higher than 10 *ms* at either Router 1 or 2. Figures 6.7(c) and 6.8(c) indicate that delay values of other classes are in the range 10-50 *ms*.

In Figures 6.7(c) and (d), and Figures 6.8(c) and (d), we show the measurements of the loss rates. Figures 6.7(c) and 6.8(c) depict the ratios of loss rates for classes 4 and 3, and for classes 3 and 2. The desired ratios of $k_2' = k_3' = 2$ are maintained most of the time. As Figures 6.7(d) and 6.8(d) indicate, the bound on the loss rates for class 1 of $L_1 = 1$ % is always kept. We also see that the loss rate of class 1 may be higher than the loss rate of other classes, because class 1 is not tied to other classes by proportional guarantees. Our implementation always drops first from class 1, until the loss bound L_1 as been reached, before dropping to satisfy proportional loss guarantees. Note that much less traffic is dropped at Router 2, because Router 2 receives traffic from Source 3 and Router 1, instead of receiving traffic from two sources. Half of the traffic arriving at Router 2 has already been policed by Router 1, resulting in a lower loss rate.

Finally, in Figures 6.7(f) and 6.8(f) we include the throughput measurements of all classes. We observe that the rate guarantee for class 2 of $\mu_2 = 35$ Mbps is maintained. The total throughput of all classes, labeled in Figures 6.7(e) and 6.8(e) as 'Total', is close to the link capacity of 100 Mbps



(a) Ratios of Delays.

(b) Delays of Class-1 Packets.

(c) Classes 2, 3 and 4 Delays

(d) Ratios of Loss Rates.

(e) Loss Rates.

(f) Throughput.

Figure 6.7: **Experiment 1: Router 1.** The graphs show the service obtained by each class at the output link of Router 1.

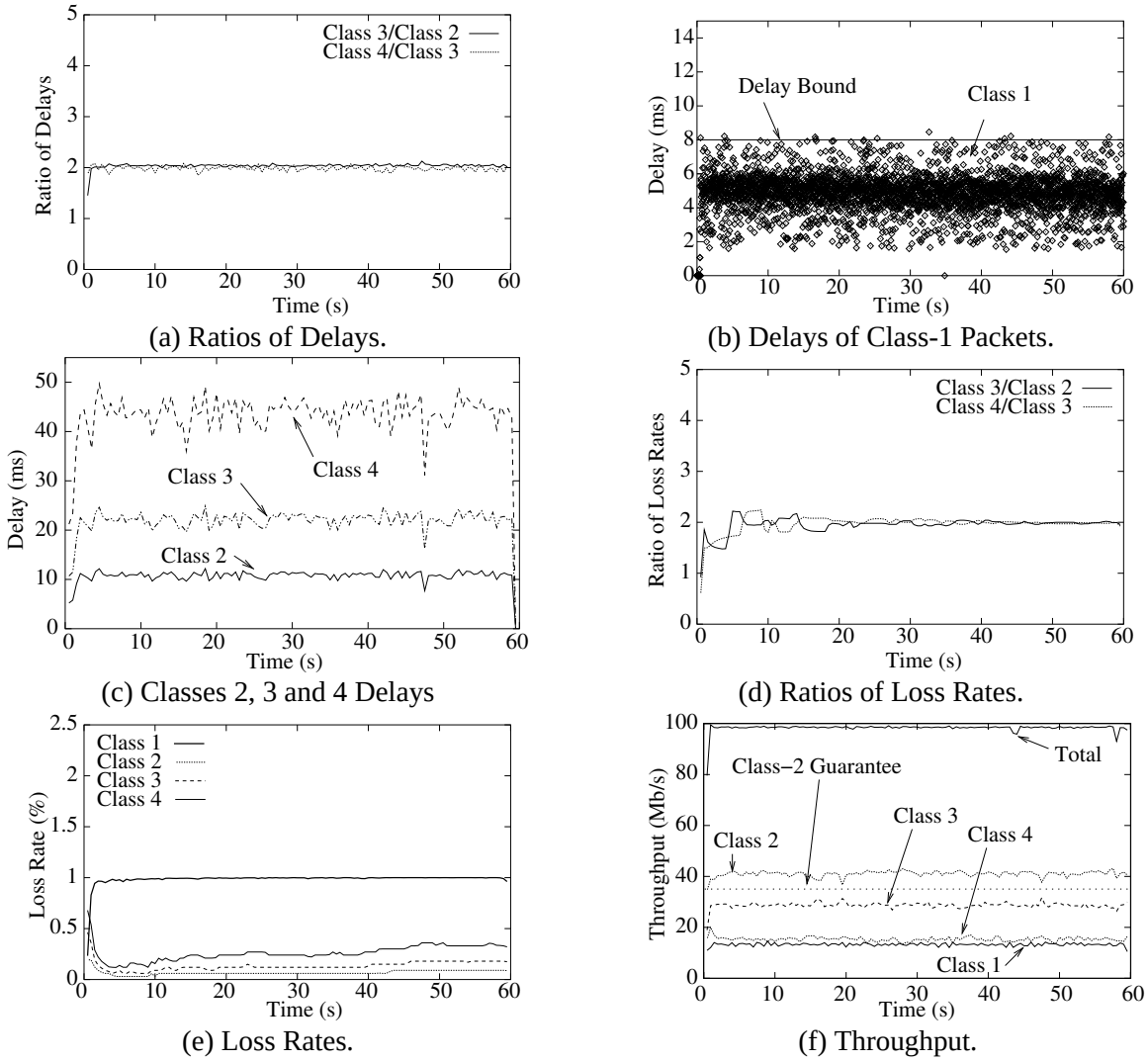


Figure 6.8: **Experiment 1: Router 2.** The graphs show the service obtained by each class at the output link of Router 2.

Class	**No. of flows**	**Type**	
		Protocol	Traffic
1	6	UDP	On-off
2	6	TCP	Greedy/On-off
3	6	TCP	Greedy/On-off
4	6	TCP	Greedy/On-off

Table 6.3: **Experiment 2: Traffic mix.** The traffic mix is identical for each source-sink pair. The on-off UDP sources send bursts of 20 packets during an on-period, and have a 150 *ms* off-period. TCP sources are greedy during time intervals $[0s, 10s]$, $[20s, 30s]$, and $[40s, 50s]$, and transmit chunks of 8 KB with a pause of 175 *ms* between each transmission during time intervals $[10, 20s]$, $[30, 40s]$, and $[50s, 60s]$. TCP sources run the *NewReno* congestion control algorithm.

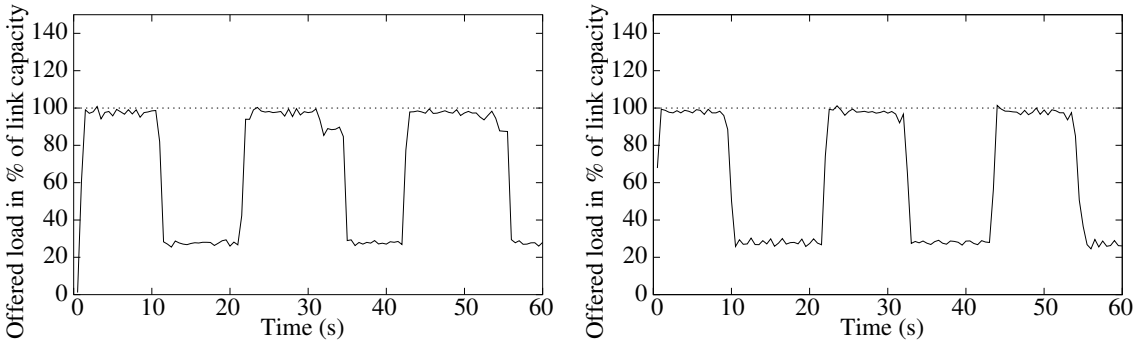
at each router.

6.3.2 Testbed Experiment 2: Highly Variable Load

The second experiment uses the network topology of Figure 6.5 and the parameters of Table 6.1. The difference between Experiments 1 and 2 consists in the traffic generation of TCP flows. Instead of using greedy TCP sources over the whole experiment, we configured the TCP sources to be greedy during time intervals $[0s, 10s]$, $[20s, 30s]$ and $[40s, 50s]$. In the remaining time intervals $(10s, 20s)$, $(30s, 40s)$, and $(50s, 60s)$, the TCP sources send chunks of 8KB of data and pause for 175 *ms* between the transmission of each chunk. We summarize the traffic mix for Experiment 2 in Table 6.3. This modification to the behavior of the TCP sources results in a more variable offered load at Routers 1 and 2, which we present in Figure 6.9.

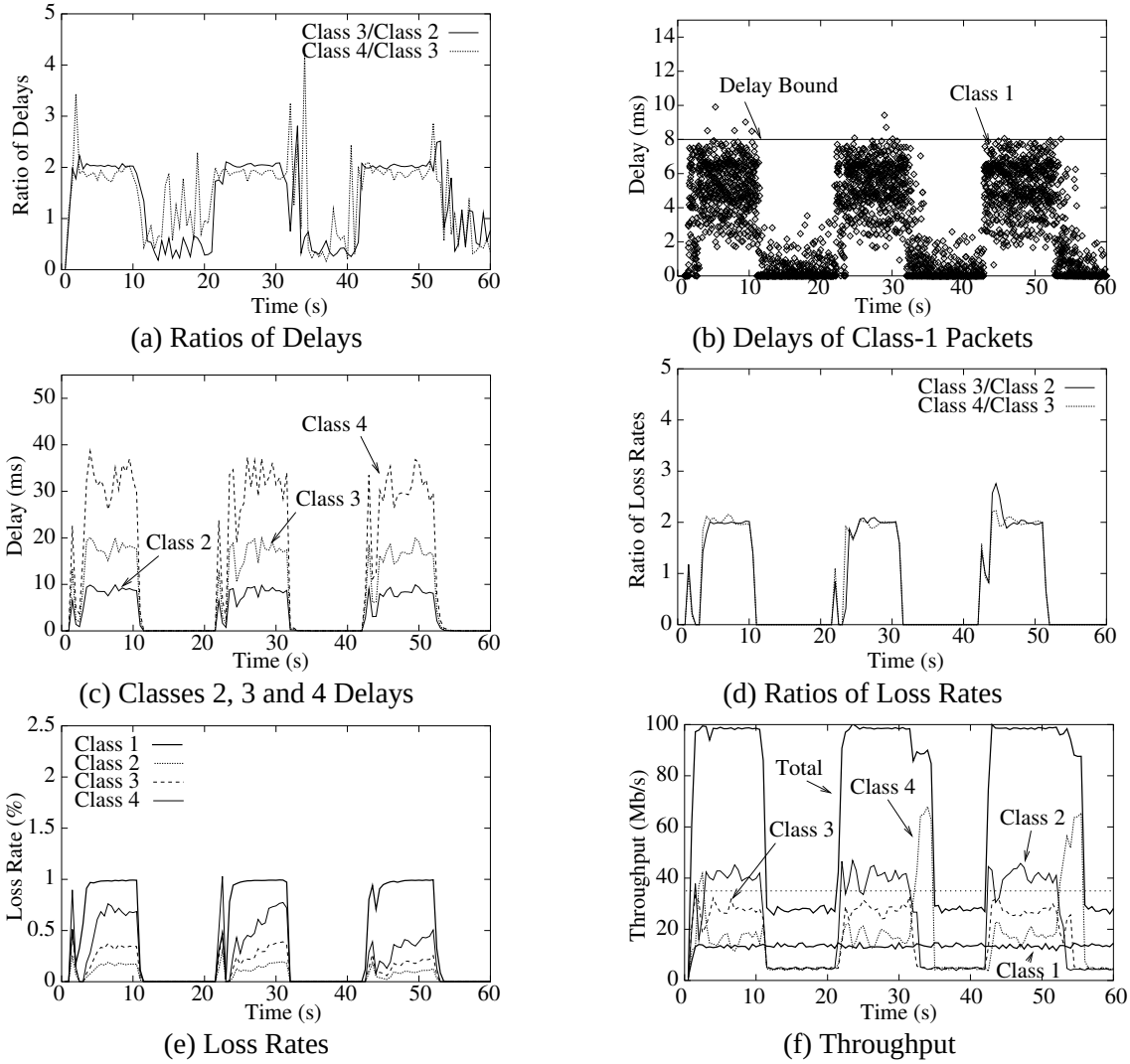
As in Experiment 1, we measure the delay, the loss rate, and the throughput of each traffic class at the bottleneck link and present our results on Figures 6.10 and 6.11. Except for Figures 6.10(b) and 6.11(b), which plot all class-1 packet delays, all datapoints correspond to moving averages over a sliding window of 0.5 *s*.

In Figures 6.10(a) and 6.11(a), we present the ratios of the delays of classes 4 and 3, and the delays of classes 3 and 2. We observe that when the load is high, in time intervals $[0s, 10s]$, $[20s, 30s]$, and $[40s, 50s]$, the target value of $k_2 = k_3 = 2$ is achieved. When the load is low, we observe oscillations in the ratios of delays, but, at both routers, all classes get low delays, as shown in Fig-



(a) Router 1. (b) Router 2.

Figure 6.9: **Experiment 2: Offered load.** The graphs show the offered load at Routers 1 and 2.



(a) Ratios of Delays (b) Delays of Class-1 Packets

(c) Classes 2, 3 and 4 Delays (d) Ratios of Loss Rates

(e) Loss Rates (f) Throughput

Figure 6.10: **Experiment 2: Router 1.** The graphs show the service obtained by each class at the output link of Router 1.

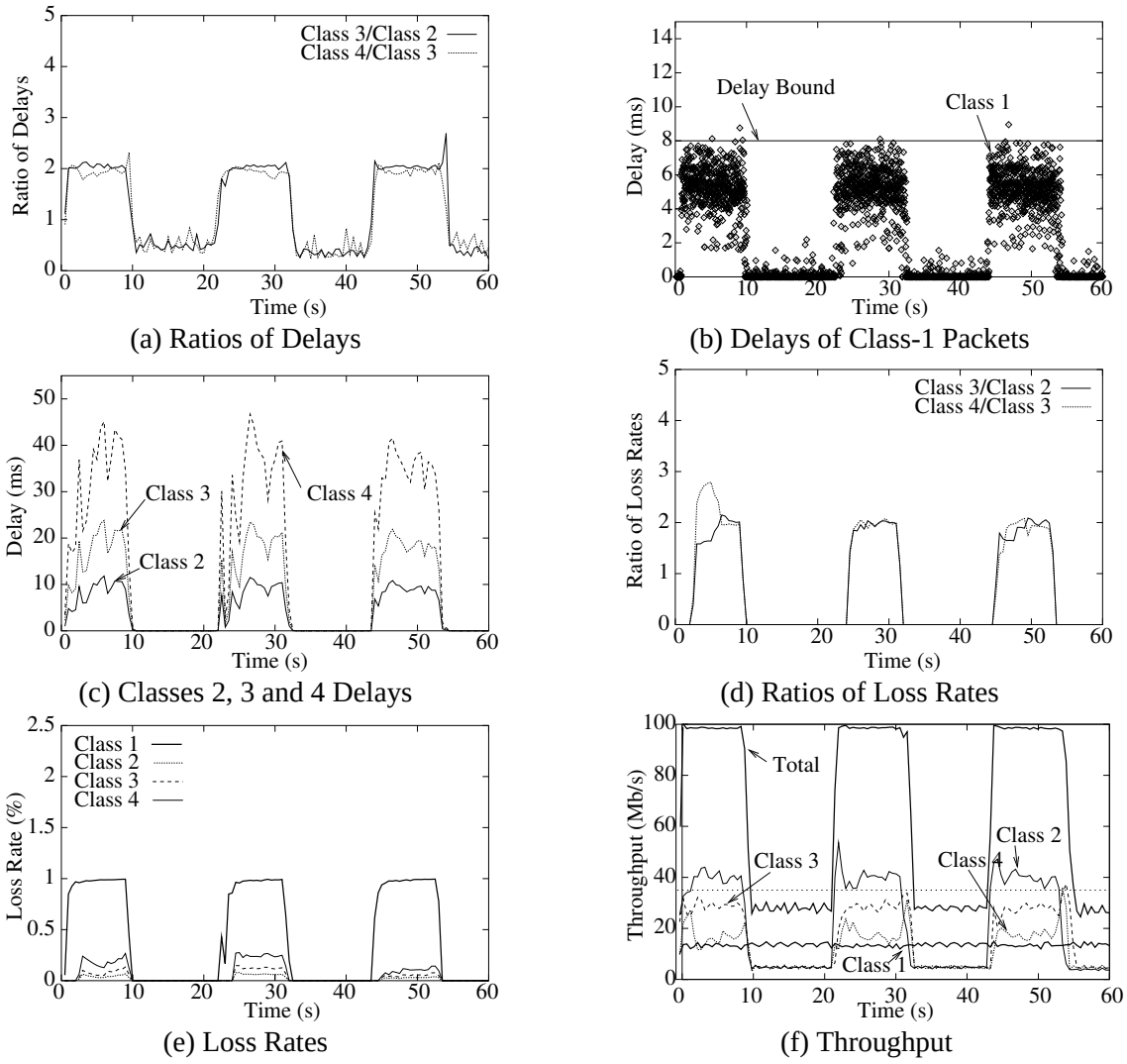


Figure 6.11: **Experiment 2: Router 2.** The graphs show the service obtained by each class at the output link of Router 2.

ures 6.10(b), 6.10(c), 6.11(b), and 6.11(c), and one can argue that there is no need for differentiation since all classes receive a high-grade service. We also see that, at times $t = 0$, $t = 20$ and $t = 40$, when the load increases abruptly over a short period of time, the delay differentiation is realized almost immediately. This confirms that our implementation quickly reacts to rapid increases in the offered load. As seen before in Experiment 1, Figures 6.10(b) and 6.11(b) show that the absolute delay guarantee of class 1, $d_1 = 8$ *ms* is generally enforced. The delay bounds are violated when it is not possible to satisfy simultaneously absolute loss and delay guarantees. In Experiment 2, the delay bound violations occur for less than 0.15% of all class-1 transmitted traffic.

Figs. 6.10(d) and (e) and 6.11(d) and (e) present plots of the ratios of loss rates averaged over a sliding window of size 0.5 *s*, and show that proportional loss differentiation is realized, with the desired factor $k_2' = k_3' = 2$, whenever there are packet losses. Figs. 6.10(d) and 6.11(d) show the loss rate experienced by class-1 traffic, and we see that, even at times of packet drops, the loss rate of class 1 remains below the loss guarantee of 1%. Loss rates of other classes are below 1%, which indicates that traffic is dropped mostly to satisfy the delay bound on Class 1.

We include the throughput measurements for all classes in Figures 6.10(f) and 6.11(f). The throughput guarantee for class 2 ($\mu_2 = 35$ Mbps) is maintained whenever class 2 is sending at more than 35 Mbps. As in Experiment 1, the QoSbox can use the full output link capacity of 100 Mbps when needed. We infer that the time needed to run the `jobs_enqueue` and `jobs_dequeue` functions is less than the average transmission time of a packet.

6.3.3 Overhead

Experiments 1 and 2 showed that our implementation of the QoSbox in PC-routers with a 1 GHz processor can fully utilize the capacity of a 100 Mbps link. We next present an analysis of the overhead of our implementation, where we attempt to predict the data rates that can be supported by the PC-router implementation of the QoSbox, and where we measure the sensitivity of our implementation to the number of service constraints and to the number of classes. We will show measurements of number of cycles consumed by the `jobs_enqueue` and `jobs_dequeue` functions for four different sets of service guarantees, tested for four traffic classes.

Set	`jobs_enqueue`		`jobs_dequeue`		**Pred.**
	$\overline{X}$	s	$\overline{X}$	s	f_{pred} (Mbps)
1	**11323**	3140	**1057**	316	**291**
2	**10723**	2305	**1092**	340	**305**
3	**3039**	1512	**1138**	348	**864**
4	**2573**	668	**1078**	343	**988**
FIFO	**25**	66	**221**	147	–

Table 6.4: **Overhead and predicted maximum throughput.** This table presents, for four different sets of service guarantees, the average number of cycles ($\overline{X}$) consumed by the `jobs_enqueue` and `jobs_dequeue` operations, the standard deviation (s), and the predicted throughput f_{pred} (in Mbps) that can be achieved. In the 1 GHz PCs we use, one cycle roughly corresponds to one nanosecond.

Set 1: Same guarantees as in Table 6.1.

Set 2: Set 1 with absolute guarantees from Set 1 removed.

Set 3: Set 2 with proportional guarantees from Set 1 removed.

Set 4: No service guarantees.

In the measurements we determine the number of cycles consumed for the `jobs_enqueue` and `jobs_dequeue` functions, and we indicate the contribution of each function call in `jobs_enqueue` to the total overhead. The TSC register of the Pentium processor is read at the beginning and at the end of each of the monitored functions, for each execution of the function.

We compiled our implementation with a code optimizer, in our case, we use the *gcc* v2.95.3 compiler [138] with the "-O2" flag set. The results of our measurements, collected under FreeBSD 4.5 and ALTQ 3.1, are presented in Table 6.4, where we include the machine cycles consumed by `jobs_enqueue` and `jobs_dequeue`, and the cycles spent in each of the functions called by `jobs_enqueue`. The measurements are averages of over 500,000 datagram transmissions on a heavily loaded link, using the same topology as in Figure 6.5. The measurements in Table 6.4 were collected at Router 1. Measurements collected at Router 2 showed deviations of no more than $\pm$5% compared to Router 1. The row containing "FIFO" denotes the overhead of the FIFO queueing discipline in ALTQ, and is used to measure the overhead created by ALTQ itself.

Set	`accounting`		`select dropped_class`		`compute min_rates`		`greedy_alloc`		`adjust_rates`	
	$\overline{X}$	s	$\overline{X}$	s	$\overline{X}$	s	$\overline{X}$	s	$\overline{X}$	s
1	777	393	46	585	582	419	3128	667	6471	1512
2	1052	350	44	616	219	135	–	–	6696	1087
3	774	401	1138	1005	568	434	3172	1044	–	–
4	948	382	793	190	202	121	–	–	–	–

Table 6.5: **Overhead distribution.** This table presents, for the four considered sets of service guarantees, the average number of cycles ($\overline{X}$) consumed by each of the functions called by `jobs_enqueue`, and the standard deviation (s). In the 1 GHz PCs we use, one cycle roughly corresponds to one nanosecond.

Since the `jobs_enqueue` and `jobs_dequeue` functions are invoked once for each IP datagram, we can predict the maximum throughput of a PC-router to be

$$f_{pred} = \frac{F}{n_{\text{enqueue}} + n_{\text{dequeue}}} \cdot \overline{P}\,, \tag{6.2}$$

where F denotes the CPU clock frequency in Hz, n_{enqueue} denotes the number of cycles consumed by the `jobs_enqueue` function, n_{dequeue} denotes the number of cycles consumed by the `jobs_dequeue` function, and $\overline{P}$ is the average size of a datagram. The equation given above assumes that clock cycles have a fixed duration, neglects bus contention and operations that occur in an interrupt context (e.g., arrival of a packet at the input link), and does not take into account the cost of packet classification. Thus, Eqn. (6.2) is an optimistic estimate. In the case of our implementation in 1 GHz PCs, we have $F = 10^9$. Data from a recent report [9] indicates that the packet size distribution on the Internet is multimodal, and that the average size of a packet on the Internet is $\overline{P} \approx 451$ bytes. Using these values for $\overline{P}$ and F in the above equation shows that, in the four sets of constraints considered, we estimate that our implementation can be run at data rates of at least 291 Mbps.[6]

Table 6.5 indicates that a dominant portion of the overhead is linked to the presence of proportional delay guarantees (Sets 1 and 2). In particular, Table 6.5 confirms most of the overhead is incurred by functions that can be sampled, notably, the implementation in `greedy_alloc` of the

[6] Previous measurements in [34] exhibited slightly more significant overhead under the same type of arrivals, but were collected using an older version of our software, and an older version of the FreeBSD operating system.

Number of classes Q	`jobs_enqueue`	`jobs_dequeue`
2	3094	1000
4	11323	1057
6	15090	1091
8	20656	1224

Table 6.6: **Overhead in function of the number of classes.** This table shows that the overhead, expressed as an average number of cycles, appears to be linear in the number of classes.

greedy algorithm for the redistribution of the service rates in presence of absolute delay bounds, and the rate adjustment for proportional delay differentiation, in `adjust_rates`. The row corresponding to Set 4 gives some insight as to the cost of the operations that have to be performed on a per-packet basis: considering that `compute_min_rates` can be sampled, the overhead, albeit not negligible, appears to be reasonable.

Last, varying the number of classes in Set 1, we gathered the overhead of the `jobs_enqueue` and `jobs_dequeue` functions in Table 6.6. Table 6.6 indicates that, as discussed in Section 6.2, the overhead of the `jobs_enqueue` function appears linear in the number of classes. The small discrepancy observed for $Q = 2$ is linked to the absence of proportional guarantees in that experiment. Measurements for the `jobs_dequeue` function tend also to exhibit linearity, but with a much higher constant cost independent of the number of classes.

6.4 Related Work

Tools that facilitate the implementation and configuration of queueing disciplines in PC-routers have been devised since the days of UNIX System V, with STREAMS [126]. More recent packages, such as ALTQ [30], Netgraph [53], the *x*-kernel [79], Click [93] and Dummynet [127] allow for implementing sophisticated queueing disciplines in Linux or BSD.

Hence, the implementation of QoS architectures using PC-routers is not new. For instance, the ALTQ package itself supports natively the Class-Based Queueing (CBQ, [69]) and the Hierarchical Fair Service Curves (HFSC, [144]) schedulers. However, without external admission control,

the ALTQ implementations of these QoS schedulers are in practice essentially used to control the bandwidth individual users can receive.

With respect to building fully functional QoS networks, one can cite the attempts at creating DiffServ networks using PC-routers. Implementations of DiffServ components in the Linux 2.1 kernel are for instance discussed in [19]. The authors of [19] integrate traffic policing and scheduling/dropping in the same router, which generates significant overhead, and, as a result, traffic can only be forwarded at approximately 20 Mbps. A similar effort to implement DiffServ components in the Linux kernels has been recently pursued by [11], and as an application of Click [93]. Last, an implementation of the Proportional Differentiated Services architecture is described in [72].

6.5 Summary and Remarks

We discussed how to implement our service architecture in an IP router, called a QoSbox. We showed through analysis and measurements that our implementation of the QoSbox in BSD-based PC-routers can be used in networks with link speeds in the order of a couple of hundred megabits per second. We also identified a few techniques that can be applied to implement our proposed mechanisms at higher line speeds.

We evaluated the performance of our implementation using a testbed of PC-routers on a 100 Mbps Ethernet network, and showed that our implementation was able to provide the desired service differentiation, thereby corroborating the simulation results presented in the previous chapters.

A version of the QoSbox for BSD kernels is available to the public, along with the source code and documentation at `http://qosbox.cs.virginia.edu`. The software has been available under the BSD license since late October 2001, and is now distributed as part of ALTQ-3.1, and of the KAME snap-kits. Inclusion in the base BSD distributions as part of ALTQ is under consideration.

Chapter 7

Extending JoBS to TCP Traffic

The JoBS scheme and the algorithms we presented so far in this dissertation do not make any distinction between TCP and non-TCP (e.g., UDP) traffic. Our algorithms drop traffic when no feasible service rate allocation exists for meeting all service guarantees, and do not take into account the sensitivity of TCP traffic to losses. TCP, which accounts for more than 90% of the total traffic on the Internet [9, 36], is a feedback-driven protocol that uses losses as an indicator for congestion avoidance and control [10, 82, 83]. Hence, TCP packet losses lead to significant performance degradation of the throughput of TCP sources. In addition, due to the relatively complex relationship between packet losses and TCP throughput [116] and the lack of discriminating mechanisms between flows belonging to the same service class, quantitative loss differentiation on traffic aggregates can result in unpredictable throughput differentiation between individual TCP flows [154].

Furthermore, JoBS is a hop-by-hop scheme, while TCP is an end-to-end protocol. A service architecture which offers absolute bounds on delays, losses and throughput, such as the Quantitative Assured Forwarding service we propose, requires to control the amount of traffic that enters the network to ensure that all guarantees can be met at all times. For instance, in the numerical examples of Chapter 6, we observed that the offered delay bounds were sometimes violated. These violations are caused by an impossibility to satisfy all service guarantees at the same time at a given link, due to the instantaneous traffic arrivals at the link. To avoid packet losses and service violations, one must consider mechanisms to reduce the traffic input to the network.

In an effort to reduce losses in TCP/IP networks, Explicit Congestion Notification (ECN, [124]) has been proposed as an additional congestion signal for TCP flows. ECN allows to mark packets with a Congestion Experienced (CE) codepoint. When a packet marked with the CE codepoint is received by its destination, the data is acknowledged with a packet containing the CE-ECHO codepoint. When the CE-ECHO marked acknowledgment reaches the sender, the sender reduces its throughput, as if a loss had happened in the network.

The emergence of ECN has stimulated research on appropriate marking algorithms at routers that indicate congestion to TCP sources to avoid packet losses resulting from buffer overflows [13,60,61,62,68,76,77,105,117]. The key idea behind these algorithms, which have been presented in Chapter 2 in the context of active queue management, is to mark packets proactively, that is, before congestion occurs, to limit the amount of lost traffic in the network. For instance, instead of dropping packets, RED can mark packets when the smoothed average of the buffer occupancy, $\overline{Q}_{est}$ is between the two thresholds min_{TH} and max_{TH}. Packets are only dropped if $\overline{Q}_{est} > max_{TH}$. Other algorithms discussed in Chapter 2, e.g., Blue [60], REM [13], or PI [77] can also use ECN marking instead of packet drops when proactive action is taken. Recall that, to mark/drop packets, the PI algorithm [77] uses a feedback-based model for TCP arrival rates [110] to let the buffer occupancy converge to a target value. The TCP model used in PI requires a priori knowledge of the number of flows traversing the router, and of the maximum round-trip time experienced by these flows.

While all of the proactive marking algorithms discussed above can, to some extent, reduce the amount of losses in the network, we try to address a broader question in this chapter. Since ECN provides congestion signals that can be conveyed before any traffic is dropped, we are exploring how ECN can be integrated with scheduling and buffer management into JoBS to extend service differentiation to TCP traffic.

We first explore if it is feasible to devise a marking algorithm which can ensure that the traffic load at a router remains at a level that entirely avoids losses due to buffer overflows at routers, without wasting available network bandwidth. The basic idea is to anticipate the behavior of TCP sources at the routers, by tracking the window size and the round-trip time of flows at the router, and to use ECN marking to have the senders adjust the window size of the flows. More precisely,

when a router predicts future losses, the router sends congestion signals to the sources via ECN with the goal of reducing the sources' sending rates before a loss occurs. To that effect, we first present a reference marking algorithm that tracks and controls all TCP flows at a router to prevent impending buffer overflows. This reference marking algorithm is useful to assess the viability of our design, but violates our design constraint of avoiding to maintain per-flow information. We note that measurement studies indicate that only a small number of TCP flows, which we call "heavy-hitters", contribute to the majority of TCP traffic [9,57,59]. We conjecture that tracking and marking only these heavy-hitters is sufficient for avoiding packet drops. Based on the idea of filtering flows, we present a set of heuristic approximations for the marking algorithm, which do not require to maintain per-flow state information. Then, we examine if ECN can be used to concurrently pursue both objectives of avoiding losses and regulating traffic to meet per-class service guarantees.

This chapter is organized as follows. In Section 7.1, we present the reference marking algorithm for avoiding buffer overflows. In Section 7.2, we describe the heuristic approximations. In Section 7.3, we show how the proposed marking algorithm can be used for traffic regulation in the context of class-based service differentiation. We compare the performance of the reference marking algorithm and of the heuristic approximations to other algorithms proposed in the literature in Section 7.4, and draw brief conclusions in Section 7.5.

7.1 A Reference Marking Algorithm for Avoiding Losses

In this section, we describe a reference algorithm for marking TCP traffic at network routers. The objective of the algorithm is to determine when to mark TCP traffic and which flows to mark in order to completely avoid packet losses due to router buffer overflows, while maximizing the utilization of the network capacity.

Throughout this section, we assume that all traffic uses TCP. While in practice one can expect a mix of flows using different protocols (e.g., TCP, UDP, SCTP [140]), we can make this assumption without loss of generality, since one can always reserve fixed resources at each router for TCP traffic such as a dedicated buffer and a fixed portion of the output link capacity. Furthermore, we assume

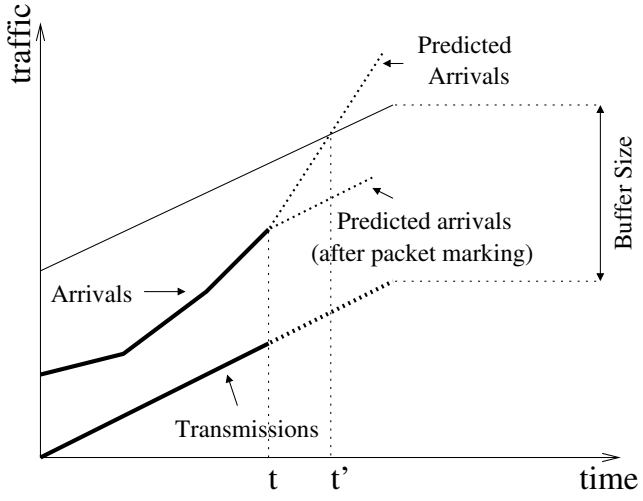


Figure 7.1: **Overview of the marking algorithm.** At time t of a packet arrival, the router predicts future arrivals, by inferring how the TCP source will send traffic. When an impending buffer overflow is predicted, at time t' here, a packet is marked to reduce future arrivals.

that ECN is available in the network. This assumption should hold in future networks, as ECN is being rapidly deployed on the Internet: recent operating systems such as FreeBSD 4.x and 5.x (with KAME) or Linux 2.4 already support ECN. For the description of the marking algorithm in the remainder of this section, we assume that enough resources are available to perform the needed computations.

We next describe the marking algorithm at a single router, for a single greedy TCP flow. For the time being, we assume that there is no other traffic in the network, and that the only cause of packet losses at the router is a buffer overflow. The router estimates the congestion window size and the round-trip time of the TCP flow. With these estimates, future traffic arrivals are predicted, and impending buffer overflows are inferred, as illustrated in Figure 7.1. In the case of Figure 7.1, at time t, a buffer overflow is predicted for time t'. If a packet loss is predicted, the algorithm reduces the congestion window size of the TCP source by marking packets with ECN. By reducing the congestion window size, the sending rate of the TCP source is reduced, and impending packet losses can be avoided. Note that the proposed algorithm does not require any changes to TCP, and only relies on ECN to reduce the traffic load.

The remainder of this section describes the calculations at the router to predict future packet losses. We explain how to use the predictions to mark traffic and avoid packet losses using the simplified model of a single TCP flow. We then generalize the proposed technique to multiple TCP flows with different sources and destinations crossing paths at a same router.

7.1.1 Predicting Traffic Arrivals to Prevent Losses

Let us assume for now that packet losses can only be caused by a buffer overflow at the considered router, and let B denote the size of the router's buffer. Using the notions of input and output curves that we defined in Chapter 3, at any time t, the backlog at the router is equal to $R^{in}(t) - R^{out}(t)$.[1] Hence, we have the following constraint:

$$\forall t : \; R^{in}(t) - R^{out}(t) \leq B \; . \tag{7.1}$$

Assume that the output link capacity of the router has a constant rate C, and that the router uses a work-conserving scheduler. Thus, for any t and $\tau > 0$ such that traffic is always backlogged over $[t, t+\tau]$,

$$R^{out}(t+\tau) = R^{out}(t) + C \cdot \tau \; . \tag{7.2}$$

Since Eqn. (7.2) characterizes R^{out} whenever there is a backlog, the algorithm only needs to infer $R^{in}(t+\tau)$ for $\tau > 0$, to ensure Eqn. (7.1) holds at $t+\tau$, thereby avoiding impending buffer overflows. To clearly distinguish between known, measured values and future, predicted values of the arrivals and of the departures, we will use the notations introduced in Chapter 3, that is, $\widetilde{R}^{in}_t(t+\tau)$ is the value predicted at time t for the input curve at time $t+\tau$. While the prediction of the output curve, $\widetilde{R}^{out}$, is identical to that introduced in Eqn. (3.8), the prediction of the input curve will use the properties of TCP congestion control to provide a more accurate prediction than the prediction proposed in Eqn. (3.7).

To predict future arrivals $\widetilde{R}^{in}_t(t+\tau)$ for $\tau > 0$, we need to examine how traffic is sent at the

[1] Since we consider a single TCP flow here, there is a single class of traffic, and we do not need to use subscripts to denote a class index.

source, so that we can infer how much traffic is received by the router. For this discussion, we consider "segments" and "packets" as synonymous. Furthermore, we ignore the slow-start phase of TCP, since the flow is unlikely to send enough traffic to create a buffer overflow during slow-start, and only focus on the congestion avoidance phase. Every time an acknowledgment is received at the source, the source sends a number of packets equal to the maximum of the receiver's advertised window size, $adv(t)$, and the source's congestion window size, $cwnd(t)$,[2] minus the number of packets sent and not yet acknowledged. $cwnd(t)$ is increased by $\frac{1}{cwnd(t)}$ every time an acknowledgment is received, unless the acknowledgment is marked with the CE-ECHO codepoint or a packet drop is inferred by reception of a triple-duplicate acknowledgment, in which case $cwnd(t)$ is decreased to $\frac{cwnd(t)}{2}$. Last, if the retransmission timer of the TCP source expires, $cwnd(t)$ is reset to one and the flow is back to slow-start.

Since $cwnd(t)$ is conditioned by receiving acknowledgments at the source, the round-trip time (RTT), that is, the time difference between the instant a packet is sent and when its acknowledgment is received at the source, is central to the evolution of $cwnd(t)$. The RTT depends on time, due to variable queueing delays, and/or changing routes. We denote by $RTT(t)$ the value of the RTT at time t, and define a series of "rounds" as follows. The first round starts when the first packet is sent by the source, and ends when the acknowledgment to the first packet sent in the first round is received. The $(k+1)$-th round starts immediately after the k-th round ends. Denoting by s_k the start time of the k-th round at the source, the s_i are linked by the recursive equation

$$s_{i+1} = s_i + RTT(s_i) \ .$$

Now, within the i-th round, i.e., between times s_i and s_{i+1}, a TCP source sends at most $W(s_i)$ packets with $W(s_i) = \max\{adv(s_i), cwnd(s_i)\}$. Furthermore, it can be shown (see [116], or the example in [139], Chap. 21) that, in absence of retransmission timer timeouts, and if the TCP source is not in slow-start mode, $W(s_{i+1})$, the number of packets sent in the $(i+1)$-th round is

[2] At the end hosts, $cwnd(t)$ is internally expressed in bytes, which does not affect our present discussion.

bounded by

$$\frac{1}{2}W(s_i) \leq W(s_{i+1}) \leq W(s_i) + 1 \ . \tag{7.3}$$

The lower bound is given by the consideration that at most one ECN congestion signal is taken into account per round [124], while the upper bound is reached only if all packets sent in the i-th round are successfully acknowledged by the destination. Note that Eqn. (7.3) is general enough to capture the behavior of Delayed-ACKs implementations, which issue on average only one acknowledgment for each two data packets.

Since $W(t)$ and $RTT(t)$ are not known by a router, Eqn. (7.3) tells us that a router that wants to estimate future traffic arrivals must be able to estimate, at any time t, $RTT(t)$, $W(t)$, and s_i for the current round. We denote by $\widehat{W}(t)$, $\widehat{RTT}(t)$, and $\widehat{s}(t)$ the estimates at the considered router of $W(t)$, $RTT(t)$, and of s_i, respectively.

These estimates are computed as follows. The first time a packet is received at the router, the current time, T_1, is recorded. When the second packet arrives at the router, at time T_2, the value of $\widehat{RTT}(T_2)$ is initialized to $T_2 - T_1$,[3] and $\widehat{W}(t)$ is initialized to 1. At time T_2, $\widehat{s}(t)$ is initialized to T_2.

After time T_2, the key idea to update the RTT estimates is to discriminate the rounds. Measurement studies [56, 121] show that the RTT is generally significantly larger than the time needed to receive all packets from a given round.[4] Thus, monitoring the packets' interarrival times at the router can determine alone if a new round has started. More specifically, if, for a constant $\kappa > 1$ chosen by the network operator, T_{i-1} and T_i satisfy

$$T_i - T_{i-1} > \frac{\widehat{RTT}(T_{i-1})}{\kappa} \ , \tag{7.4}$$

the router considers that T_i marks the start of a new round. Empirical results, such as our evaluation in Section 7.4, indicate that using $\kappa \approx 10$ generally manages to accurately distinguish between different rounds.

If Eqn. (7.4) does not hold, T_{i-1} and T_i are part of the same round, and $\widehat{RTT}(T_i)$ is set equal to

[3] This method is equivalent to the SYN-ACK algorithm of [89].

[4] The same assumption is used in [116] for modeling the sending rate of a TCP source, and has been confirmed in experimental measurements.

$\widehat{RTT}(T_{i-1})$, $\widehat{W}(T_i)$ is set to $\widehat{W}(T_{i-1})+1$, and $\widehat{s}(T_i)$ is set equal to $\widehat{s}(T_{i-1})$. Conversely, if Eqn. (7.4) holds, the router updates the estimates as follows:

$$\begin{aligned}
\widehat{s}(T_i) &= T_i \,, \\
\widehat{W}(T_i) &= 1 \,, \\
\widehat{RTT}(T_i) &= \alpha \cdot \widehat{RTT}(\widehat{s}(T_{i-1})) + (1-\alpha) \cdot (\widehat{s}(T_i) - \widehat{s}(T_{i-1})) \,,
\end{aligned}$$

where $0 \leq \alpha \leq 1$ is a constant. The method to estimate $\widehat{RTT}(t)$ described above is similar to the round-trip time estimator at the TCP sources [82], which uses $\alpha = 0.9$, and has shown to provide reasonably accurate results. We point out that except in rare cases of persistent link failure, where packets end up being re-routed, the RTT does not vary significantly over time, and thus, the algorithm should be rather insensitive to the selection of α.

With the estimates of the RTT and the window size, the router can predict future window sizes. Specifically, for any time t and any time $\tau > 0$, denoting by $\widetilde{W}_t(t+\tau)$ the prediction of the window size at time $t+\tau$, the router computes $\widetilde{W}_t(t+\tau)$ as

$$\widetilde{W}_t(t+\tau) = \begin{cases}
\widehat{W}(t) & \text{if } t+\tau < \widehat{s}(t) + \widehat{RTT}(t), \\
\widehat{W}(t)+1 & \text{if } t+\tau \geq \widehat{s}(t) + \widehat{RTT}(t) \\
 & \text{and no packet has been} \\
 & \text{marked (or dropped)} \\
 & \text{in } [\widehat{s}(t), t], \\
\frac{1}{2}\widehat{W}(t) & \text{if } t+\tau \geq \widehat{s}(t) + \widehat{RTT}(t) \\
 & \text{and at least one packet} \\
 & \text{has been marked (or} \\
 & \text{dropped) in } [\widehat{s}(t), t].
\end{cases} \tag{7.5}$$

The router can discover if a packet has been marked (or dropped upstream) in $[\widehat{s}(t), t]$ by checking the ECN bits and the TCP sequence numbers. From Eqn. (7.5), the router predicts that the window size does not change until the end of the current round, and that its value at the beginning of the next

round depends on whether or not a packet has been dropped or marked during the current round. Eqn. (7.5) captures that, at the earliest, ECN signals have an effect only at the beginning of the *next* round.

We shall note that this prediction is correct only when all packets in a round have been received by the router. This may seem a restriction, but since the RTT of a flow is generally larger than the time needed to receive all packets in a given round for this flow [56, 116, 121], the prediction is generally accurate. With $\widetilde{W}_t(t+\tau)$ given by Eqn. (7.5), a router can predict the input curve with the following expression:

$$\widetilde{R}_t^{in}(t+\tau) = R^{in}(t) + MSS \cdot \gamma_t(\tau) \cdot \widetilde{W}_t(t+\tau) \ , \tag{7.6}$$

where *MSS* is the maximum segment size of the TCP flow, and

$$\gamma_t(\tau) = \begin{cases} 1 & \text{if } t+\tau \geq \widehat{s}(t) + \widehat{RTT}(t), \\ 0 & \text{otherwise.} \end{cases}$$

That is, a router can assume that all traffic sent in the next round arrives in a batch right at the start of the next round, which is a conservative assumption. In practice, such bursts of traffic are rarely observed.

Next, we discuss how arrivals are marked. To determine if an arrival at time t must be marked, a router checks that the flow has not already been marked (or has experienced some losses) during the current round. This test is necessary since at most one ECN-marked packet per round has an impact on the arrivals. If the flow has not experienced any losses or packet marking during $[\widehat{s}(t), t]$, the router verifies if the following condition holds:

$$\widetilde{R}_t^{in}(t+\tau) - \widetilde{R}_t^{out}(t+\tau) \leq B \ ,$$

that is, replacing the prediction of the output curve by its expression,

$$\widetilde{R}_t^{in}(t+\tau) - R^{out}(t) - C \cdot \tau \leq B \ . \tag{7.7}$$

This condition tests if a buffer overflow is going to occur at the beginning of the next round. Since ECN feedback does not have any impact until the beginning of the next round, the condition in Eqn. (7.7) is checked for $\tau = \widehat{s}(t) + \widehat{RTT}(t) - t$. If the condition of Eqn. (7.7) is violated, then the router marks the packet at the head of the transmission queue with the CE codepoint. Marking the packet at the head of the queue minimizes the delay needed for the ECN feedback to reach the source.

We conclude with a discussion on the robustness of the above estimators. If the constant κ in Eqn. (7.4) is too small (e.g., $\kappa = 1.01$), or if $W(t)$ is extremely large and data transmission appears continuous, the test described in Eqn. (7.4) may not be able to discriminate between rounds. In the worst-case, the router may never infer the start of a new round, and $\widehat{W}$ grows unbounded. To address this problem, we use a safeguard, based on Eqn. (7.3) as follows. If, at time T_i, we have

$$\widehat{W}(T_i) > \widehat{W}(\widehat{s}(T_i)^-) + 1 \ ,$$

the router infers that T_i marks the start of a new round, *even if Eqn. (7.4) does not hold*. Now, if κ is too large (e.g., $\kappa > 1000$), the router incorrectly infers that each packet arrival marks the start of a new round, and thus, $\widehat{W}$ and $\widehat{RTT}$ underestimate W and RTT. In the worst-case, when $\widehat{W} \to 0$ and $\widehat{RTT} \to 0$, no prediction is performed, thus no traffic is marked, and the algorithm degenerates to Drop-Tail. Our experiments show that the algorithm is quite robust to changes of the parameters. In fact, the experimental results gathered in Section 7.4 with $\kappa = 10$, $\alpha = 0.9$ are almost identical to those obtained with any value $10 \leq \kappa < 100$, and $0.7 < \alpha < 1$.

7.1.2 Generalization to Multiple TCP Flows

We next consider a more general situation with N greedy TCP flows. We use $\widehat{RTT}^j(t)$, $\widehat{W}^j(t)$, MSS^j, and $\widehat{s}^j(t)$ to denote the estimated round-trip time, congestion window size, maximum segment size and start time of the current round for TCP flow j, respectively. Let us assume, for the moment, that the router is able to monitor all N TCP flows and can keep track of all the $\widehat{RTT}^j(t)$, $\widehat{W}^j(t)$, MSS^j and $\widehat{s}^j(t)$.

Now, by defining for each flow j, at any time t,

$$\tau^j = \hat{s}^j(t) + \widehat{RTT}^j(t) - t \ , \tag{7.8}$$

i.e., τ^j is the (estimated) remaining time before the start of the next round for TCP flow j, and by iterating the prediction technique of Section 7.1.1 for all flows, the router first computes the predicted congestion window in the next round, $\widetilde{W}_t^j(t+\tau^j)$ for each flow j, using Eqn. (7.5). Then, for any $\tau > 0$, the predicted arrivals are

$$\widetilde{R}_t^{in}(t+\tau) = R^{in}(t) + \sum_j MSS^j \cdot \gamma_t^j(\tau) \cdot \widetilde{W}_t^j(t+\tau) \ , \tag{7.9}$$

where

$$\gamma_t^j(\tau) = \begin{cases} 1 & \text{if } \tau \geq \tau^j, \\ 0 & \text{otherwise.} \end{cases}$$

If the condition given in Eqn. (7.7) is violated for any of the τ^j's of Eqn. (7.8), the algorithm proactively marks the oldest backlogged packet from flow k with

$$k = \arg\max\{ j \mid \widetilde{W}_t^j(t+\tau^j) = \widehat{W}^j(t) + 1\} \ , \tag{7.10}$$

that is, the algorithm marks the flow with the largest congestion window that has not yet been marked (or experienced a packet drop) in its current round. As soon as the oldest backlogged flow-k packet is marked, $\widetilde{W}_t^k(t+\tau_k)$ is set to $\widehat{W}^k(t)/2$, and the condition of Eqn. (7.7) is reevaluated. The marking process is repeated until Eqn. (7.7) does not hold for any of the τ^j's, or all flows have one packet marked in the current round.

7.2 Emulating the Reference Algorithm without Per-Flow State

The algorithm presented in Section 7.1 maintains the per-flow state information, thereby violating our design constraints. We now present a set of heuristic approximations for the reference

algorithm. The approximations use flow filtering to reduce the number of tracked TCP flows, and employ linear interpolation to reduce the computational complexity of the predictions.

7.2.1 Flow Filtering

As observed in measurement studies [9, 57, 59], only a small percentage of flows ("heavy-hitters") accounts for a large percentage of traffic. These heavy-hitters transmit at a high data rate due to (1) a large congestion window, and (2) a relatively small round-trip time. From the description of the reference algorithm in Section 7.1, these are generally the only flows that need to be marked by the reference algorithm. Thus, by limiting the tracking operations to the heavy-hitters we expect that the reference algorithm can be closely approximated.

To identify the heavy-hitters, we use the serial multistage filter proposed in [54]. The objective of the multistage filter is to identify, at any time t, the flows that have sent more than θ bytes during the time interval $(\lfloor t/\sigma \rfloor \cdot \sigma, t)$, where θ is a given threshold, and $\sigma > 0$ is a fixed time constant denoting the sampling interval used for measurement. The serial multistage filter proposed in [54] works as follows. Every time a packet arrives at the router, a hash function is applied to the source and destination IP addresses and port numbers. Flows are then grouped into buckets depending on the value returned by the hash function. Then, flows in the fullest buckets are hashed by a second, independent, hash function and grouped into second-level buckets. The same type of hashing operation is repeated a third time. Flows belonging to the fullest buckets after the third hash are recorded into memory. The authors of [54] showed that the serial multistage filter minimizes false positives (i.e., only a few flows with a small sending rate are labeled as heavy-hitters) and avoids false negatives (i.e., all flows with a large sending rate are tracked).

We implement flow filtering as follows. We use two linked lists in the router's memory, $\mathcal{L}_1$ for current sampling, and $\mathcal{L}_2$ for flows previously recorded. Initially, both $\mathcal{L}_1$ and $\mathcal{L}_2$ are empty. In the first sampling interval, flows are added to $\mathcal{L}_1$ only if they pass the multistage filter, while $\mathcal{L}_2$ remains empty. At time $t = \sigma$, $\mathcal{L}_1$ is copied into $\mathcal{L}_2$ before being reset.[5] The process is iterated

[5] This operation can be implemented efficiently by swapping the two pointers on $\mathcal{L}_1$ and $\mathcal{L}_2$, and resetting the pointer on $\mathcal{L}_1$.

every σ seconds. At any time t, the router updates the estimates $\widehat{RTT}$, $\widehat{W}$, and $\widehat{s}$ for all flows in $\mathcal{L}_1$ and $\mathcal{L}_2$.

Only the flows in $\mathcal{L}_2$ are used for the predictions, and the prediction of Eqn. (7.9) always underestimates the input curve. To adjust the estimate of the input curve, at any time t, we introduce a correction factor, $\zeta(t)$, whose value is updated at $t = k\sigma$, where k is a positive integer, with

$$\zeta(t) = \frac{R^{in}(t) - R^{in}((k-1)\cdot\sigma)}{\sum_{j\in\mathcal{L}_2}(R^{in,j}(t) - R^{in,j}((k-1)\cdot\sigma))} ,$$

where $R^{in,j}(t)$ denotes here the amount of flow-j traffic received by the router by time t. That is, at any time t, $\zeta(t)$ denotes the ratio of the total amount of traffic received by the router in the previous sampling interval over the amount of traffic that was identified in the previous sampling interval. Note that we always have

$$\zeta(t) \geq 1 .$$

The case $\zeta(t) = 1$ is an extreme case where all flows pass the filter during the previous sampling interval. As an example, for $t = 5.5$ *s*, and $\sigma = 1$ *s*, if $\zeta(t) = 1.1$, we know that 90.9 % of all traffic received by the router in the time interval $(4\text{ s}, 5\text{ s})$ has been identified. At any time t, the prediction of the input curve for the traffic aggregate, $\widetilde{R}_t^{in}$ is set equal to the sum of the prediction of the input curves of the flows in $\mathcal{L}_2$, multiplied by the correction factor $\zeta(t)$, that is

$$\widetilde{R}_t^{in}(t+\tau) = \zeta(t)\cdot\sum_{j\in\mathcal{L}_2}\widetilde{R}_t^{in,j}(t+\tau) .$$

Remark: We note that the selection of the parameters σ and θ presents a trade-off between computational overhead and accuracy of the algorithm. With a larger sampling interval σ, the updates to main memory, $\mathcal{L}_2$, are performed less frequently, at the expense of using possibly obsolete data. With a larger value for θ, the number of recorded flows, $\mathcal{X}$, remains small, but the accuracy of the predictions may be poor. Thus, we infer that both θ and σ should be tuned according to the computational power available. In particular, routers at high-speed access points, and a large number of flows, should be configured with relatively large values for θ and σ.

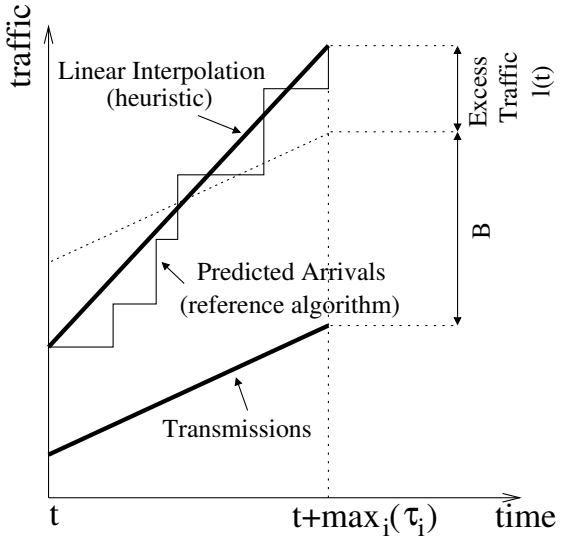


Figure 7.2: **Linear interpolation.** In the heuristic, only the value $\widetilde{R}_t^{in}(t+\max_i\{\tau_i\})$ is computed, and is used to determine the excess traffic that will arrive at the router.

7.2.2 Linear Interpolation

Flow filtering limits the amount of state information recorded at the router, but does not improve the computational overhead of the construction of the predicted input curve. We next describe a technique that reduces the complexity of the prediction of Eqn. (7.9).

First, instead of using individual values of the congestion windows of all recorded flows in the construction of the predicted input curve, we consider that all recorded flows have a congestion window size (in bytes) equal to the mean congestion window size (in bytes), $\bar{\Omega}$, given by

$$\bar{\Omega}(t) = \frac{1}{X} \sum_{j \in \mathcal{L}_2} MSS^j \cdot \widehat{W}^j(t) \ .$$

This approximation is justified by the fact that we use flow filtering and ignore flows with small congestion windows.

Second, we use linear interpolation to reduce the complexity of the construction of the predicted input curve and illustrate our method in Figure 7.2. Rather than constructing the whole predicted input curve, only the value $\widetilde{R}_t^{in}(t+\max_j\{\tau^j\})$ is computed. Intermediary values $\widetilde{R}_t^{in}(t+\tau)$ for

$0 < \tau < \max_j\{\tau^j\}$ are approximated using a linear interpolation, based on the value obtained for $\widetilde{R}_t^{in}(t + \max_j\{\tau^j\})$. The reason for selecting $\max_j\{\tau^j\}$ as the basis for the linear interpolation, instead of, for instance, $\min_j\{\tau^j\}$, is that the prediction can take into account all recorded flows.

Next, Eqn. (7.7) tells us that

$$\ell(t) = \widetilde{R}_t^{in}(t + \max_j\{\tau^j\}) - R^{out}(t) - C \cdot \max_j\{\tau^j\} - B$$

is the amount by which the traffic must be reduced to prevent buffer overflows. From $\ell(t)$ and $\bar{\Omega}(t)$, the algorithm can infer the number of flows that have to be marked, and only update the predicted input curve once, which reduces the worst-case complexity of the prediction operations to $O(1)$. If $\ell(t) > 0$, the marking process performs at most $O(Y)$ operations where Y is the number of backlogged packets. The worst-case occurs when all packets backlogged have to be marked at the same time. In practice, we only expect at most a couple of flows to be marked upon each packet arrival, since predictions are performed over short time intervals.

7.3 Traffic Regulation with ECN Marking in Class-Based Service Architectures

In this section, we build on the algorithm we described in Sections 7.1 and the approximations of 7.2 to describe how ECN marking can be used to extend JoBS to TCP traffic.

Recall Eqn. (5.4) is a sufficient condition for all Class-i traffic to meet its delay and throughput guarantees at any time t when Class i is backlogged. To be able to satisfy Eqn. (5.4) for each class, at any time t, we need to have

$$\sum_i \max\left\{\frac{R_i^{in}(t) - R_i^{out}(t)}{d_i - D_i(t)}, \mu_i \cdot \chi_{B_i(t)>0}\right\} \leq C\ . \tag{7.11}$$

If the condition of Eqn. (7.11) is violated, one can reduce $R_i^{in}(t)$ by dropping traffic, as we did in the previous chapters. Since our objective here is to avoid any traffic drops, we use the predictions described earlier to ensure that the $R_i^{in}(t)$'s always satisfy Eqn. (7.11).

Assuming the throughput guarantees are appropriately chosen, that is,

$$\sum_i \mu_i < C \ ,$$

we propose the following approach. At time t, in addition to the predictions on the input curve of all flows j in Class i, $\widetilde{R}_{i,t}^{in,j}(t+\tau)$, which is given by Eqn. (7.6), and the class-i predicted input curve, given by

$$\widetilde{R}_{i,t}^{in}(t+\tau) = \sum_j \widetilde{R}_{i,t}^{in,j}(t+\tau) \ ,$$

we also predict the Class-i output curve, $\widetilde{R}_{i,t}^{out}(\tau)$ by

$$\widetilde{R}_{i,t}^{out}(t+\tau) = R_i^{out}(t) + \tau \cdot r_i(t) \ ,$$

where $\tau > 0$. Note that the prediction of the output curve is the same as originally proposed in Eqn. (3.8).

If the rate allocation r_i remains unchanged between t and $t+\tau$, this prediction of $\widetilde{R}_{i,t}^{out}(t+\tau)$ is exact. Since we only use the prediction for small values of τ (in the order of a round-trip time) we can assume that the prediction is reasonably accurate, even if r_i changes between t and $t+\tau$. Furthermore, let us assume that the delay of Class i remains roughly constant during $[t, t+\tau]$. With these predictions defined, we can predict the minimum service rates $\widetilde{r}_{i,t}^{\min}(t+\tau)$ needed at time $t+\tau$, so that all service guarantees on throughputs and delays are met:

$$\widetilde{r}_{i,t}^{\min}(t+\tau) = \max\left\{ \mu_i, \frac{\widetilde{R}_{i,t}^{in}(t+\tau) - \widetilde{R}_{i,t}^{out}(t+\tau)}{d_i - D_i(t)} \right\} \ .$$

To ensure that the set of service rates required for meeting service guarantees is always feasible, we must enforce

$$\sum_i \widetilde{r}_{i,t}^{\min}(t+\tau_i^j) \leq C \ , \tag{7.12}$$

for all τ_i^j's defined by Eqn. (7.8) for each class i. If Eqn. (7.12) does not hold, the incoming traffic needs to be reduced. To that effect, we propose to first identify the set of classes where

$\widetilde{r}_{i,t}^{\min}(t+\tau) > \mu_i$, which are the classes where decreasing the traffic arrivals has an effect on the minimum service rate required. Since $\sum_i \mu_i < C$, we know that there is at least one class in that set.

Once the classes whose traffic need to be throttled have been identified, the marking process is carried out in the same manner as in the case of an impending buffer overflow, by merely replacing the condition given in Eqn. (7.7) by the condition given in Eqn. (7.12).

7.4 Evaluation

In this section, we evaluate our proposed marking algorithm via simulation, using the *ns-2* network simulator. The evaluation has three objectives. First, we compare the performance of the reference algorithm and of the algorithm with the heuristic approximations. Second, we compare the performance of our proposed algorithm to state-of-the-art active queue management algorithms. Third, we illustrate how JoBS performance is improved with the marking algorithm presented in this chapter. We present two simulation experiments. The first experiment evaluates the efficiency of the proposed approach with respect to buffer management, while the second experiment evaluates the performance of a combination of the heuristic approximations of the marking algorithm and the closed-loop algorithm for rate allocation and buffer management of Chapter 5 to provide service guarantees.

7.4.1 Experiment 1: Active Queue Management

In the first simulation experiment, we consider a bottleneck link with capacity $C = 10$ Mbps, and buffer size of $B = 150{,}000$ bytes. All traffic at this single bottleneck link is TCP (NewReno), and is generated by 60 greedy FTP flows, and 180 on-off flows, aiming at emulating HTTP connections. The sources of the on-off flows send on average 300 packets during an "on" period, and pause on average for one second between two "on" periods. The actual number of packets sent and the wait time between two transmissions are exponentially distributed. All packets have a size of 500 bytes. In the absence of queueing and transmission delays in the network, the RTTs of all flows are independent identically distributed random variables uniformly distributed between 24 *ms* and

180 *ms*, and to avoid synchronization effects, sources start transmitting at different times, uniformly distributed between 0 *s* and 5 *s*. The experiment lasts for 70 seconds of simulated time, and ECN is available in the entire network. The load is roughly constant and equal to the link capacity. We compare the performance of six different algorithms at the router governing the bottleneck link:

- **Drop-Tail.** We use Drop-Tail to have an estimate of the loss rates encountered without active queue management. With Drop-Tail, incoming packets are discarded only when the buffer is full.

- **RED [68].** We use RED with the `gentle_` variant [67], with a minimum threshold $min_{TH} = 37{,}500$ bytes, and a maximum threshold $max_{TH} = 75{,}000$ bytes. The parameter max_P is set to 0.1, and the weight used in the computation of the average queue size is set to $w_q = 0.002$. While min_{TH} and max_{TH} are chosen so that traffic is dropped with a probability of one only if the buffer is full, other parameters are the default RED parameters in *ns-2*, and are therefore expected to cover a large range of operating conditions. RED is set to use ECN marking instead of packet dropping whenever possible.

- **PI [77] with approximate parameter tuning.** To account for the uncertainty on estimates of the RTTs and of the number of flows at router configuration time, we configure here the PI algorithm with crude estimates of the RTTs and of the number of flows. That is, we use a lower bound on the number of flows of $N = 50$, and a maximum RTT $R^+ = 300$ *ms*, with a sampling frequency of 160 Hz, yielding parameter values of $a = 0.2395e-4$ and $b = 0.2388e-4$. The target queue length is set to Q_{ref}=100,000 bytes.

- **PI with exact parameter tuning.** We configure the PI algorithm with the exact RTTs and number of flows we use in our simulation. That is, we use a lower bound of $N = 60$ on the number of flows, and a tight upper bound on the round-trip times $R^+ = 180$ *ms*, with a sampling frequency of 160 Hz, and get $a = 1.643e-4$ and $b = 1.628e-4$. The target queue length Q_{ref} is set to 100,000 bytes. Note that such an exact parameter imposes a priori knowledge of the number of flows and of the round-trip times of the flows that will traverse the router at router configuration time, which may be difficult to obtain in practice.

- **Reference algorithm.** This is the reference marking algorithm described in Section 7.1. Results are obtained for $\kappa = 10$, $\alpha = 0.9$. We achieved similar results with parameter settings in the range $10 \leq \kappa < 100$ and $0.7 < \alpha < 1$, which tends to show that our proposed marking algorithm is relatively insensitive to the selection of these parameters.

- **Heuristic approximations.** This is the marking algorithm using the heuristic approximations described in Section 7.2. The multistage filter consists of 3 stages of 8 buckets. The admission threshold is set to θ =200,000 bits and $\sigma = 1$ *s*.

For each algorithm, we monitor the loss rates over a sliding window of size 0.25 *s*, and present our results in Figure 7.3. We start monitoring at time $t = 10$ *s* to ignore transient effects linked to the initial empty state of the network. Figure 7.3(a) tells us that, without active queue management, one can expect loss rates in the order of 12 %. Figure 7.3(b) and (c) show that RED with the default parameters, which turn out to be unsuitable for the traffic mix at hand, and a crudely configured PI algorithm, drop almost as much traffic as Drop-Tail. Conversely, a perfectly tuned PI algorithm manages to avoid most packet drops. The reference algorithm completely avoids packet losses, and the heuristic rarely drops any packets, delivering results close to those of the reference algorithm.

Next, we monitor the aggregate throughput and goodput observed at the receivers, averaged over a moving window of size 0.25 *s* and present our results in Figure 7.4. The throughput characterizes the total amount of traffic received by the transport layer at the destination, while the goodput characterizes the amount of traffic that is passed by the transport layer to the application layer. The main observation is that all schemes manage to achieve an aggregate throughput roughly equal to the capacity of the bottleneck link. Furthermore, we note that, by avoiding packet losses, an exactly tuned PI, and both the reference algorithm and the heuristic approximations manage to achieve a goodput close to the throughput.

Last, we monitor the queue size, averaged over a moving window of size 0.25 *s*, and we present our results in Figure 7.5. Not surprisingly, the Drop-Tail queue is almost always full, which explains the relatively high loss rates. RED manages to stabilize the queue length around $max_{TH} = 75{,}000$ bytes. (This observation coupled with the result presented in Figure 7.3 indicates that RED drops

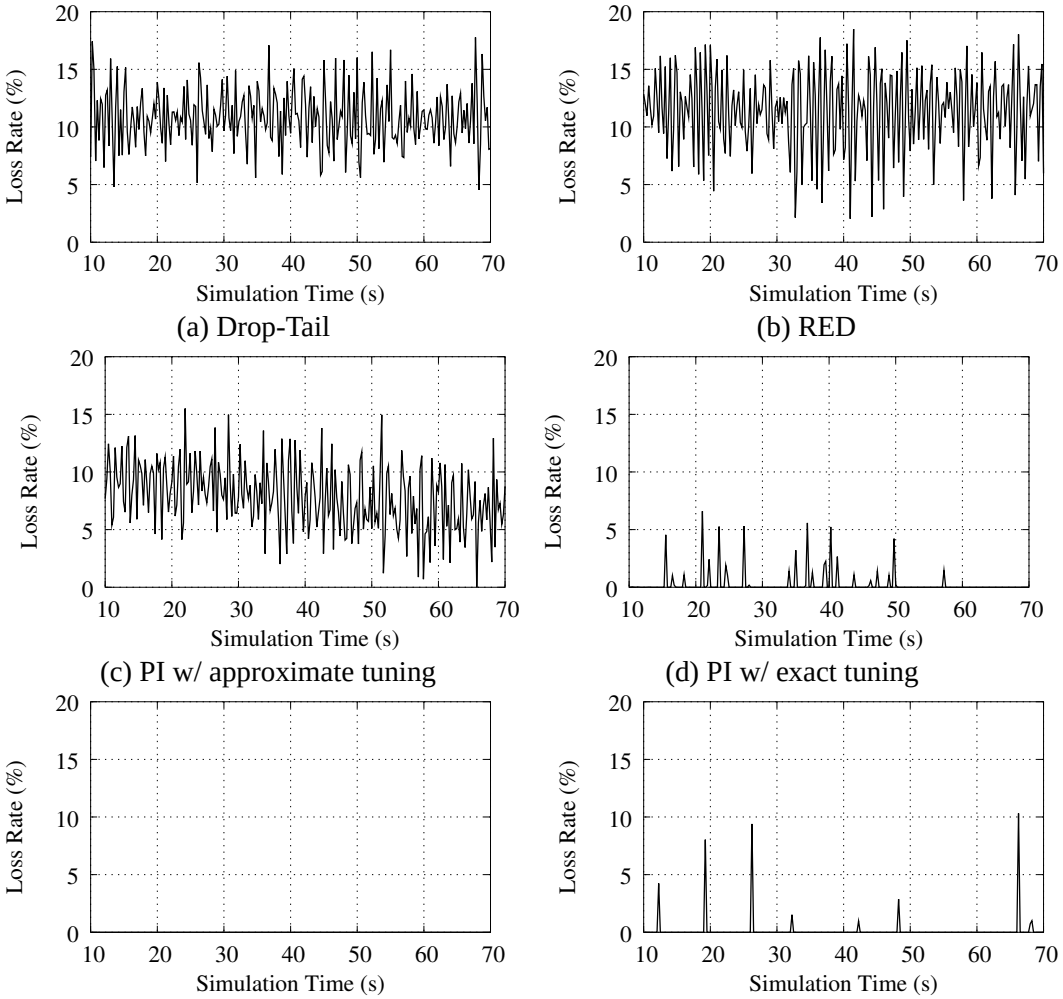


Figure 7.3: **Loss rates**. The figures compare the loss rates obtained with all six algorithms. Note that the reference algorithm does not discard any traffic.

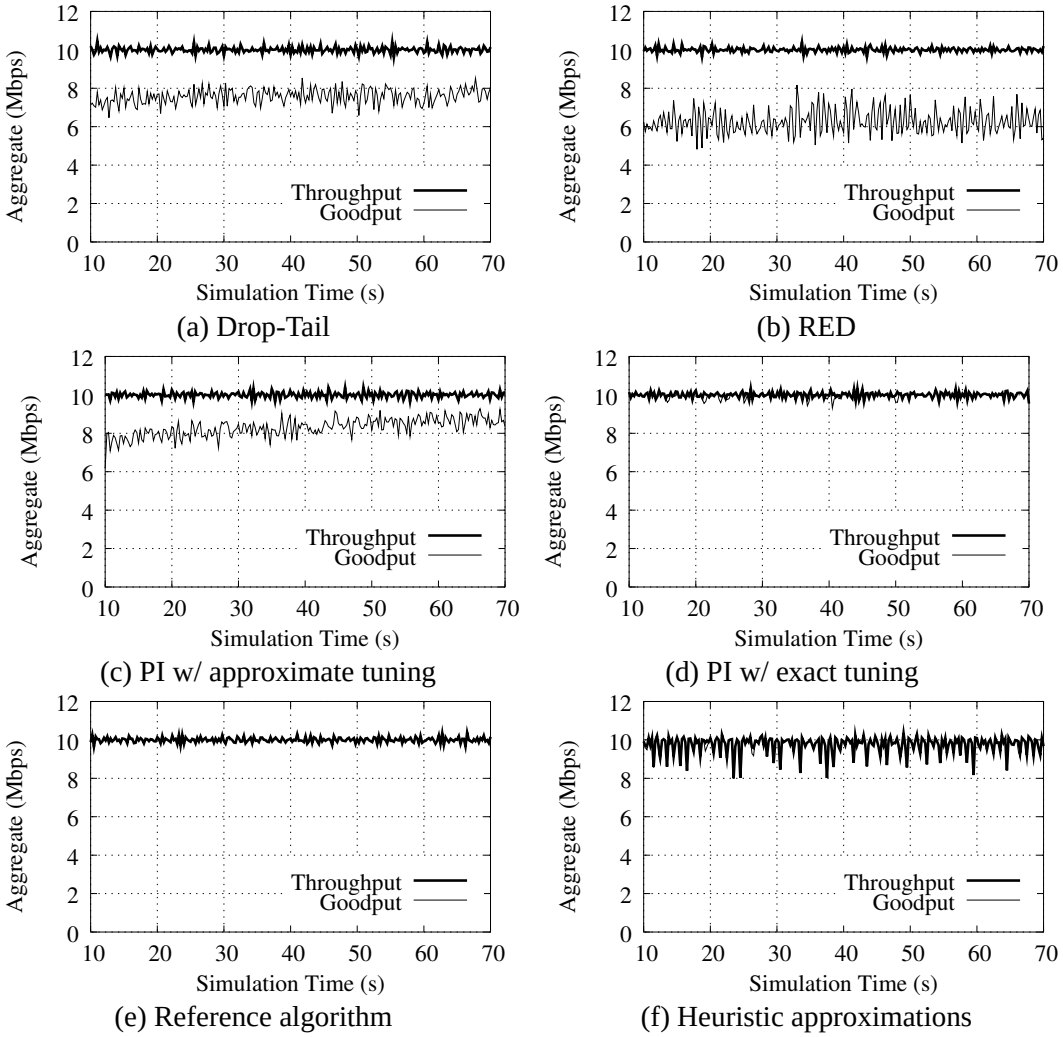


Figure 7.4: **Measured throughput and goodput at the receivers.** The figures compare the aggregate throughput and goodput seen at the receivers with all six algorithms. All schemes are efficient at maximizing the utilization of the bottleneck link (10 Mbps). A perfectly tuned PI, and both the reference algorithm and its heuristic approximations have a goodput (amount of traffic passed to the application layer at the destinations) almost equal to the throughput (total amount of traffic received at the destinations) by avoiding packet drops.

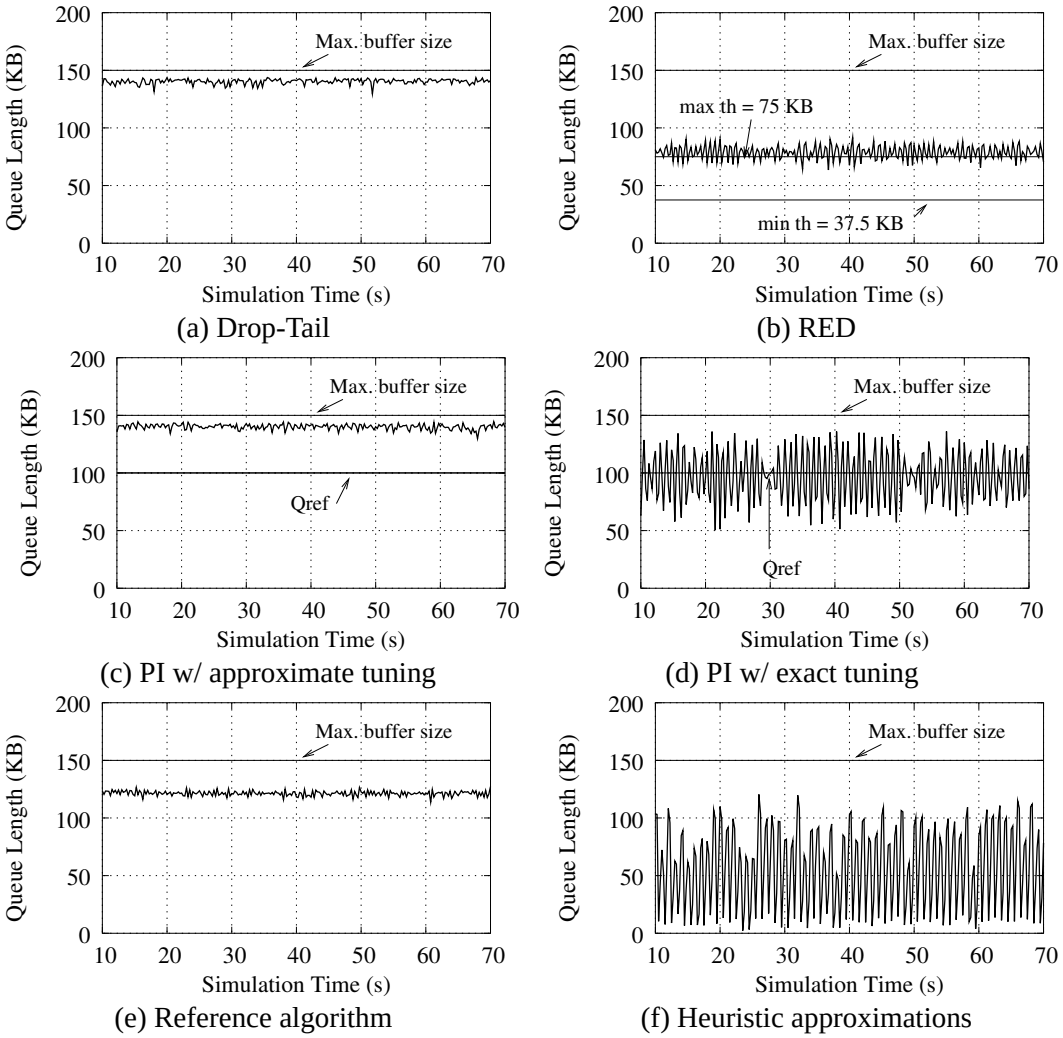


Figure 7.5: **Queue lengths.** The figures compare the queue lengths at the router for all six algorithms.

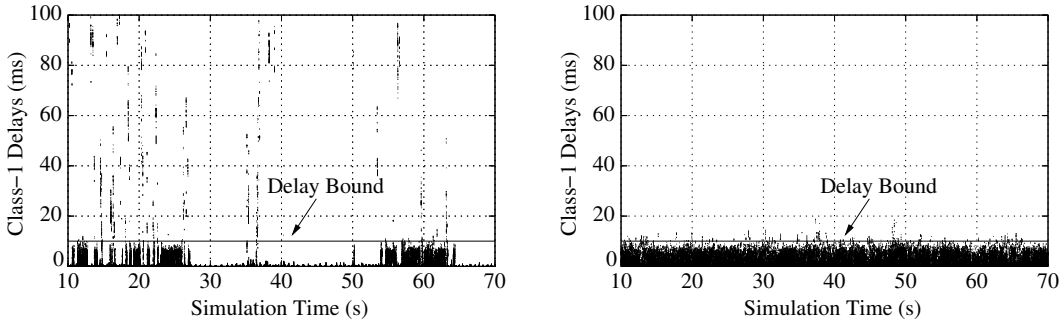
Class	Number of on-off flows	Number of greedy flows	QoS Guarantees		
			Delay	Loss Rate	Throughput
1	5	3	$\leq 10\ ms$	$\leq 1\ \%$	≥ 5 Mbps
2	10	3	$\approx \frac{1}{4}D_3$	$\approx \frac{1}{2}p_3$	–
3	15	3	$\approx \frac{1}{4}D_4$	$\approx \frac{1}{2}p_4$	–
4	20	3	–	–	–

Table 7.1: **Traffic mix and service guarantees.** The second column indicates the number of on-off flows, the third column the number of greedy flows, and in the third and fourth rows, as in previous chapters, p_i denotes the loss rate of Class i over a busy period, and D_i denotes the delay of Class i.

some packets proactively even when ECN is available.) With an approximate tuning of the configuration parameters, PI does not manage to track the desired queue length $Q_{ref} = 100{,}000$ bytes, and instead, the queue is almost always full. Conversely, a properly tuned PI algorithm manages to achieve the target Q_{ref}, albeit with some oscillations around the target value. While stabilizing the queue length is not the primary objective of our algorithm, the reference algorithm manages to keep the queue length almost constant around 120,000 bytes. The heuristic approximations keep the queue length in the vicinity of 50,000 bytes, with oscillations of a magnitude comparable to those of a well-configured PI controller. These oscillations are mostly due to the choice of a one second sampling interval, and are reduced for higher sampling frequencies, at the expense of a higher computational overhead.

7.4.2 Experiment 2: Providing Service Guarantees

Next, we assess the effectiveness of our algorithms at regulating traffic in the context of our proposed service architecture. To that effect, we run a second experiment, with a bottleneck link with capacity $C = 45$ Mbps, and a buffer size of $B = 250{,}000$ bytes. All traffic at the bottleneck link is TCP (NewReno), and consists of 12 greedy TCP flows, and 50 on-off TCP flows, following the same on-off pattern as in the first experiment. The RTTs of all greedy TCP flows are equal to 44 *ms*, and the RTTs of the on-off flows, in the absence of propagation and transmission delays, are uniformly distributed between 44 *ms* and 80 *ms*. All sources start transmitting at time $t = 0$ for 70 seconds of simulated time, and ECN is available.



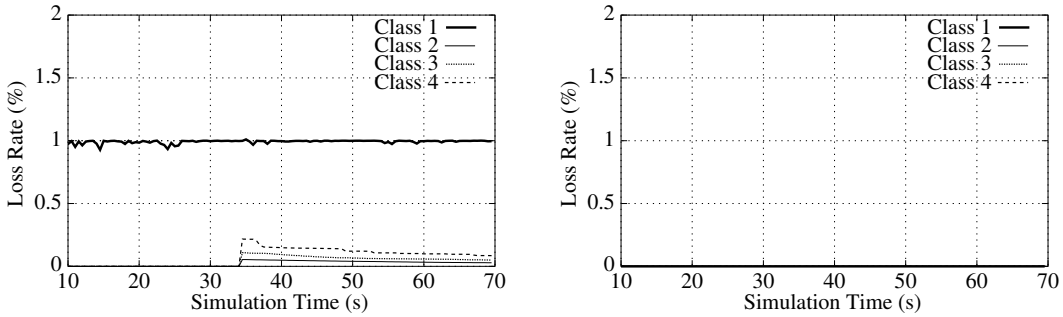
(a) without marking algorithm (b) with marking algorithm

Figure 7.6: **Class-1 packet delays.** Note that the violations are much smaller in magnitude with the marking algorithm.

We consider four classes of traffic, with the service guarantees and traffic mix described in Table 7.1. In addition to the on-off flows, each class contains three greedy TCP flows. We compare the performance of two algorithms in this experiment. The first algorithm is the closed loop algorithm for buffer management and rate allocation described in Chapter 5, without any ECN marking. The second algorithm combines the closed-loop algorithm of Chapter 5 with the marking algorithm described in Section 7.3 and the heuristic approximations described in Section 7.2, using a multistage filter of 3 stages of 8 buckets, $\sigma = 0.1$ *s*, $\theta = 200,000$ bits.

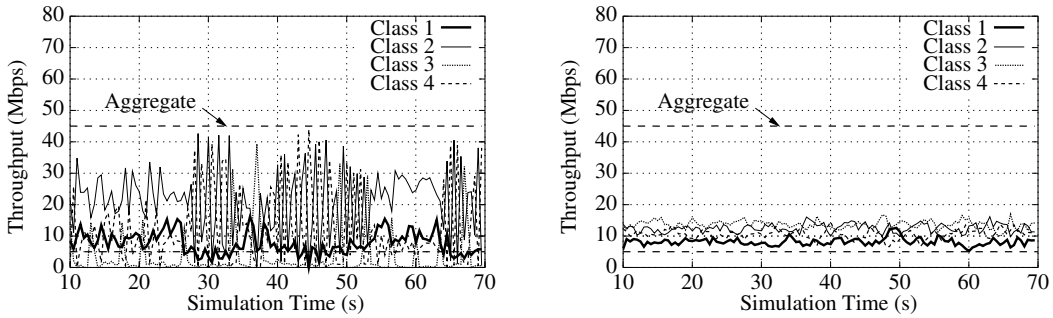
We plot the delays encountered by each Class-1 packet at the bottleneck link in Figure 7.6. Figure 7.6(a) shows that, given the traffic mix considered, about 11 % of all Class-1 packets exceed the delay bound of 10 *ms*, with queueing delays going as high as 100 *ms*. This is due to the order chosen for relaxing service guarantees, which gives precedence to the loss guarantee and relaxes the delay bound. Clearly, in this experimental setup, traffic regulation is urgently needed, because the buffer size at the transmission queue (250 KB) is small compared to the output link capacity (45 Mbps), and stringent loss rate bounds (1%) and delay bounds (10 *ms*) are offered to a same class of traffic, often resulting in an infeasible set of service guarantees. Conversely, Figure 7.6(b) shows that when the marking algorithm we described in this chapter is used, violations rarely happen (< 2 %), and the delay does not exceed 20 *ms*.

Next, in Figure 7.7, we plot the loss rates averaged over the length of the current busy period.



(a) without marking algorithm (b) with marking algorithm

Figure 7.7: **Loss rates.** The marking algorithm prevents any traffic from being dropped.



(a) without marking algorithm (b) with marking algorithm

Figure 7.8: **Per-class throughputs.** Without the marking algorithm, we observe oscillations and sporadic violations of the Class-1 throughput guarantee. The marking algorithm stabilizes these oscillations and ensures the throughput guarantees are respected.

Figure 7.7(a) show that all loss guarantees are respected, notably the 1 % bound on Class-1 losses. However, as we have seen in Figure 7.6(a), the loss rate bound is respected at the expense of the delay bound. Figure 7.7(b) shows that, with the addition of the algorithm of Section 7.3, no packets are lost, and therefore, the objective of completely avoiding packet drops to meet service guarantees is met.

Finally, in Figure 7.8 we present the throughput obtained by each class at the bottleneck link. Figure 7.8(a) shows without the marking algorithm, severe oscillations of the throughput can be observed. These oscillations are due to TCP sources reacting to packet losses. The throughput bound on Class-1 is sometimes violated, due to the fact that there is not enough Class-1 traffic

present in the router. Figure 7.8(b) shows that our marking algorithm stabilizes these oscillations in throughput, and that the throughput guarantee on Class 1 is always respected. We also note that the aggregate throughput is equal to the capacity of the bottleneck link whether or not the marking algorithm is used. This means that the stabilization in the throughputs provided by the marking algorithm algorithm does not come at the expense of under-utilization.

7.5 Summary and Remarks

We investigated whether marking algorithms for ECN can be used for regulating traffic in the context of class-based service architectures, while avoiding packet losses due to buffer overflows. To that effect, we first described a reference marking algorithm for IP routers, which attempts to eliminate packet losses in TCP flows. The proposed algorithm infers how traffic is sent by TCP sources, by tracking the window size and RTT of large flows, and accordingly makes the marking decisions. We then showed how the proposed algorithm can be used for traffic regulation in the context of QoS architectures, in lieu of traffic policing or admission control. Experimental results illustrated the potential of the approach.

We note that the techniques used in the algorithms can be further improved by more accurate and robust estimators of the RTT values, e.g., [89], and of the congestion window sizes. Another area for improvement resides in the type of filter used in the heuristic approximations. While the serial multistage filter [54] we use for our algorithm appears to exhibit good performance, a follow-up work described in [55] indicates that parallel multistage filters typically perform better than serial multistage filters, and are more amenable to mathematical analysis of their properties, such as probabilities of false negatives. Using a parallel multistage filter could therefore open the door for an analytical evaluation of our proposed algorithms, and help quantify the trade-offs in parameter selection.

Furthermore, our current approach assumes TCP Reno or NewReno; extending it to other flavors of TCP such as SACK [109], or Vegas [25] could be of interest.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

For the past decade, a significant amount of research in data networks and telecommunications has been devoted to providing different levels of service to applications on the Internet. In this dissertation, we presented a novel service architecture for the Internet, which reconciles application demand for strong service guarantees with the need for low computational overhead in network routers.

The main contribution of this dissertation is the definition and realization of a new service, called Quantitative Assured Forwarding, which offers absolute and relative differentiation of loss, service rates, and packet delays to classes of traffic. The Quantitative Assured Forwarding can be viewed as a generalization of all previous class-based service models. We devised and analyzed mechanisms that implement the proposed service, and demonstrated the effectiveness of the approach through analysis, simulation and measurement experiments in a testbed network.

8.1.1 Scheduling and Buffer Management

A key scheme to realize the proposed service architecture is to combine service rate allocation and buffer management in a single step. This scheme is called JoBS, short for Joint Buffer Management and Scheduling. Based on JoBS, we first presented a reference algorithm that dynamically solves

a non-linear optimization to allocate service rates and drop traffic, and we proposed a heuristic algorithm to approximate the solution to the non-linear optimization.

We then devised a closed-loop algorithm relying on feedback control theory, that provides a very close match to the non-linear optimization while relying on relatively simple operations.

We described the design of IP routers, called QoSboxes, that realize the QAF service. We presented the details of our implementation in PC-routers of the closed-loop algorithm, which is distributed as part of the recent ALTQ-3.1 package and of the KAME snapkits, and is also available from `http://qosbox.cs.virginia.edu`. The outcomes of this research include the following:

- By combining service rate allocation and buffer management in a single algorithm, one can provide a service architecture that subsumes all other per-hop, per-class service architectures in terms of service guarantees, without resorting to admission control or traffic policing. We showed by simulation that our approach matched the performance of algorithms specifically designed for proportional differentiation, while being able to enforce absolute guarantees at the same time.

- The non-linear optimization problem that characterizes the service rate allocation and traffic drops can be closely approximated by a closed-loop algorithm based on linear feedback control theory. We can derive stability conditions on the linearized feedback loops. The approximations made for linearizing the feedback loops appear to be valid in practice, since the feedback loops exhibit stability, as we demonstrated through a number of simulations and testbed experiments.

- The closed-loop algorithm can be implemented at link speeds in the order of a few hundreds of megabits per second in 1-GHz PC-routers running variants of the 4.4-BSD operating system. We described the approximations necessary for an implementation at higher speeds.

8.1.2 Extending JoBS to TCP

We proposed extensions to the JoBS scheme to reconcile the per-hop, per-class guarantees of the QAF service, with the properties of TCP, which is an end-to-end transport protocol sensitive to

losses. In particular, we considered the design of a packet marking algorithm that attempts to avoid losses by using the feedback capabilities of TCP traffic, and of the recently proposed Explicit Congestion Notification.

Also, even though combining scheduling and buffer management manages to enhance per-class service differentiation, controlling the amount of traffic to enter the network remains a necessity to prevent cases where sudden bursts of traffic render a system of service guarantees infeasible. We showed our marking algorithm was a potential alternative to admission control and traffic policing.

Our research on extending a QoS architecture to take into account TCP traffic indicated that:

- It is possible to design ECN marking algorithms that completely eliminate losses in TCP/IP networks. Simulation experiments illustrated that a reference algorithm using per-flow information was able to completely avoid packet drops, thereby maximizing the goodput of the TCP flows.

- Because of the asymmetry of Internet traffic, where 90% of the traffic is carried by 10% of the flows, one can use flow filtering and avoid maintaining per-flow state information for all flows, while providing a reasonable approximation of the reference algorithm.

- In addition to avoiding packet losses, using ECN marking is a possible alternative to admission control for traffic regulation in the context of class-based service differentiation.

8.2 Future Work

We conclude here by outlining areas which can be of potential interest for future research.

A thorough inspection of the interaction of traffic engineering techniques with our proposed service model could be of interest. Specifically, path replication techniques, such as one-to-one protection, are increasingly used in the context of load-balancing (see [151], for instance) and core provisioning, and can be efficiently implemented using emerging technologies such as MPLS [128]. One can investigate how service guarantees, and particularly throughput guarantees, should be selected when path replication and load-balancing are used in a network.

Another question that can be worth pursuing regards the class selection in our proposed service architecture. Note that class selection is highly dependent on the type of guarantees offered, and as such, some assumptions regarding the specific type of service guarantees provided would need to be made. Partial answers to the class selection problem are already available in the literature. For instance, Dovrolis and Ramanathan show in [49] that, for a network providing proportional delay differentiation, having the end applications dynamically select their class of service enables to obtain end-to-end delay bounds.

Letting the end applications select the class of traffic they want to use requires to enforce collaboration between the different applications to avoid cases where each application marks all packets with the best class of service available. The other option, which is to let the network select the class of traffic assigned to different flows, requires cooperation between the different domains, so that different network domains agree on some common semantics for the services offered.

We note that the class selection problem, which requires collaboration between different entities, is only an instance of a much larger problem, which is to extend QoS architectures such as proposed in this dissertation to provide economic incentives. As noted in a recent NSF workshop report, "one of the impediments to the deployment of new services on the Internet is the lack of market incentives to improve network services and applications and to use them efficiently." Trying to provide market incentives is likely to foster some additional technological challenges. For instance, in an economic context, the end applications need to be able to verify with certainty the quality of the service they receive. Devising good service verification mechanisms is still an open problem, which may be worth pursuing.

Bibliography

[1] The FreeBSD project. http://www.freebsd.org.

[2] Intel's IXP 1200 network processor. http://developer.intel.com/design/network/products/npfamily/ixp1200.htm.

[3] The KAME project. http://www.kame.net.

[4] The NetBSD project. http://www.netbsd.org.

[5] *ns-2* network simulator. http://www.isi.edu/nsnam/ns/.

[6] The OpenBSD project. http://www.openbsd.org.

[7] Napster protocol specification, April 2000. http://opennap.sourceforge.net/napster.txt.

[8] The Gnutella protocol specification v0.4, June 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[9] Packet sizes and sequencing, May 2001. http://www.caida.org/outreach/resources/learn/packetsizes.

[10] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. IETF RFC 2581, April 1999.

[11] W. Almesberger, J. H. Salim, and A. Kuznetsov. Differentiated services on Linux, June 1999. IETF draft, draft-almesberger-wajhak-diffserv-linux-01.txt. See also http://diffserv.sourceforge.net.

[12] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, 11(4):319–352, November 1993.

[13] S. Athuraliya, D. Lapsley, and S. Low. An enhanced random early marking algorithm for internet flow control. In *Proceedings of IEEE INFOCOM 2000*, pages 1425–1434, Tel-Aviv, Israel, April 2000.

[14] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang. The Tenet real-time protocol suite: design, implementation, and experiences. *IEEE/ACM Transactions on Networking*, 4(1):1–10, 1996.

[15] J. Bennett, K. Benson, A. Charny, W. Courtney, and J.-Y. Le Boudec. Delay jitter bounds and packet scale rate guarantee for expedited forwarding. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1502–1509, Anchorage, AK, April 2001.

[16] J. Bennett and H. Zhang. WF^2Q: worst-case weighted fair queueing. In *Proceedings of IEEE INFOCOM '96*, volume 1, pages 120–128, San Francisco, CA, March 1996.

[17] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk-Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, August 1994.

[18] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. IETF RFC 2475, December 1998.

[19] R. Bless and K. Wehrle. Evaluation of differentiated services using an implementation under Linux. In *Proceedings of IWQoS '99*, pages 97–106, London, UK, June 1999.

[20] S. Bodamer. A scheduling algorithm for relative delay differentiation. In *Proceedings of the IEEE Conference on High Performance Switching and Routing (ATM 2000)*, pages 357–364, Heidelberg, Germany, June 2000.

[21] U. Bodin, A. Jonsson, and O. Schelen. On creating proportional loss differentiation: predictability and performance. In *Proceedings of IWQoS 2001*, pages 372–386, Karlsruhe, Germany, June 2001.

[22] J.-Y. Le Boudec and A. Charny. Packet scale rate guarantee for non-FIFO nodes. In *Proceedings of INFOCOM 2002*, volume 1, pages 84–93, New York, NY, June 2002.

[23] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. IETF RFC 1633, July 1994.

[24] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP). IETF RFC 2205, September 1997.

[25] L. Brakmo and L. Peterson. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[26] Z. Cao, Z. Wang, and E. Zegura. Rainbow fair queueing: fair bandwidth sharing without per-flow state. In *Proceedings of IEEE INFOCOM 2000*, volume 2, pages 922–931, Tel-Aviv, Israel, April 2000.

[27] C.-S. Chang. Stability, queue length, and delay of deterministic and stochastic queueing networks. *IEEE Transactions on Automatic Control*, 39(5):913–931, May 1994.

[28] C.-S. Chang. *Performance Guarantees in Communication Networks*. Springer Verlag, London, UK, 1999.

[29] A. Charny and J.-Y. Le Boudec. Delay bounds in a network with aggregate scheduling. In *Proceedings of QoFIS 2000*, pages 1–13, Berlin, Germany, September 2000.

[30] K. Cho. A framework for alternate queueing: towards traffic management by PC-UNIX based routers. In *Proceedings of USENIX '98 Annual Technical Conference*, pages 247–258, New Orleans, LA, June 1998.

[31] N. Christin. QoSbox visualization applet. `http://qosbox.cs.virginia.edu/snooplet.html`.

[32] N. Christin and J. Liebeherr. The QoSbox: A PC-router for quantitative service differentiation in IP networks. Technical Report CS-2001-28, University of Virginia, November 2001. `ftp://ftp.cs.virginia.edu/pub/techreports/CS-2001-28.pdf`. In revision.

[33] N. Christin and J. Liebeherr. A QoS architecture for quantitative service differentiation. *IEEE Communications*, 41(6):38–45, June 2003.

[34] N. Christin, J. Liebeherr, and T. F. Abdelzaher. A quantitative assured forwarding service. In *Proceedings of IEEE INFOCOM 2002*, volume 2, pages 864–873, New York, NY, June 2002.

[35] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input-output queued switch. In *Proceedings of IEEE INFOCOM '99*, volume 3, pages 1169–1178, New York, NY, March 1999.

[36] K. Claffy, G. Miller, and K. Thompson. The nature of the beast: recent traffic measurement from an Internet backbone. In *Proceedings of INET '98*, Geneva, Switzerland, July 1998.

[37] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4):362–373, August 1998.

[38] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM'92*, pages 12–24, Baltimore, MD, 1992.

[39] A. Clerget and W. Dabbous. Tag-based fair bandwidth sharing for responsive and unresponsive flows. In *Proceedings of IEEE INFOCOM 2001*, volume 1, pages 498–507, Anchorage, AK, April 2001.

[40] Compaq Computer Corporation. *Alpha Architecture Handbook*, 4th edition, 1998.

[41] R. L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[42] R. L. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

[43] B. Davie, A. Charny, J. Bennett, K. Benson, J.-Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An expedited forwarding PHB. IETF RFC 3246, March 2002.

[44] S. Deering and R. Hinden. Internet protocol version 6 (IPv6) specification. IETF RFC 2460, December 1998.

[45] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. In *Proceedings of ACM SIGCOMM '89*, pages 1–12, Austin, TX, September 1989.

[46] C. Dovrolis. *Proportional differentiated services for the Internet*. PhD thesis, University of Wisconsin-Madison, December 2000.

[47] C. Dovrolis and P. Ramanathan. A Case for Relative Differentiated Services and the Proportional Differentiation Model. *IEEE Networks*, 13(5):26–34, September 1999. Special issue on Integrated and Differentiated Services on the Internet.

[48] C. Dovrolis and P. Ramanathan. Proportional differentiated services, part II: Loss rate differentiation and packet dropping. In *Proceedings of IWQoS 2000*, pages 52–61, Pittsburgh, PA, June 2000.

[49] C. Dovrolis and P. Ramanathan. Dynamic class selection: From relative differentiation to absolute QoS. In *Proceedings of ICNP 2001*, pages 120–128, Riverside, CA, November 2001.

[50] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *Proceedings of ACM SIGCOMM '99*, pages 109–120, Boston, MA., August 1999.

[51] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. *IEEE/ACM Transactions on Networking*, 10(1):12–26, February 2002.

[52] K. Cho (editor). Manpage for `altq.conf(5)`. See `http://netbsd.gw.com/cgi-bin/man-cgi?altq.conf`.

[53] J. Elischer and A. Cobbs. The Netgraph networking system. Technical report, Whistle Communications, January 1998. `http://www.elischer.com/netgraph/`.

[54] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2001 ACM SIGCOMM Internet Measurement Workshop*, pages 75–80, San Francisco, CA, November 2001.

[55] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM '02*, pages 323–336, Pittsburgh, PA, August 2002.

[56] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communications Review*, 26(3):5–21, July 1996.

[57] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Proceedings of IEEE GLOBECOM '99*, volume 3, pages 1859–1868, Rio de Janeiro, Brazil, December 1999.

[58] A. Feldmann, A. Gilbert, W. Willinger, and T. Kurtz. The changing nature of network traffic: Scaling phenomena. *Computer Communication Review*, 28(2):5–29, April 1998.

[59] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: methodology and experience. In *Proceedings of ACM SIGCOMM '00*, pages 257–270, Stockholm, Sweden, August 2000.

[60] W.-C. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: A new class of active queue management algorithms. Technical Report CSE-TR-387-99, University of Michigan, April 1999.

[61] W.-C. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic fair blue: a queue management algorithm for enforcing fairness. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1520–1529, Anchorage, AK, April 2001.

[62] W.-C. Feng, D. Saha, D. Kandlur, and K. Shin. A self-configuring RED gateway. In *Proceedings of IEEE INFOCOM '99*, volume 3, pages 1320–1328, New York, NY, March 1999.

[63] D. Ferrari. Real-time communication in an internetwork. *Journal of High-Speed Networks*, 1(1):79–103, January 1992.

[64] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[65] V. Firoiu and A. Casati. Amendments to the Assured Forwarding PHB group, November 1998. IETF Internet Draft. Copy available from `http://www.globecom.net/ietf/draft/draft-firoiu-diffserv-af-amend-00.html`.

[66] V. Firoiu, X. Zhang, and E. Gündüzhan. Feedback output queueing: a novel architecture for efficient switching systems. In *Proceedings of Hot Interconnects X*, Stanford, CA, August 2002.

[67] S. Floyd. Recommendation on using the `gentle_` variant of RED, March 2000. See `http://www.icir.org/floyd/red/gentle.html`.

[68] S. Floyd and V. Jacobson. Random early detection for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, July 1993.

[69] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[70] C. Fraleigh, F. Tobagi, and C. Diot. Provisioning ip backbone networks to support latency sensitive traffic. In *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, March 2003.

[71] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital control of dynamic systems*. Addison-Wesley, Menlo Park, CA, 3rd edition, 1998.

[72] B. Gaidioz, M. Goutelle, and P. Primet. Implementation of IP proportional differentiation with waiting-time priority and proportional loss rate dropper in Linux. Technical Report RR-4511, INRIA, July 2002.

[73] B. Gaidioz, P. Primet, and B. Tourancheau. Differentiated fairness: Service model and implementation. In *Proceedings of HPSR 2001*, pages 260–264, Dallas, TX, May 2001.

[74] D. Hartmeier. Openbsd packet filter. See `http://www.openbsd.org/faq/pf/`.

[75] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group. IETF RFC 2597, June 1999.

[76] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. A control-theoretic analysis of RED. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1510–1519, Anchorage, AK, April 2001.

[77] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1726–1734, Anchorage, AK, April 2001.

[78] P. Hurley, J.-Y. Le Boudec, P. Thiran, and M. Kara. ABE: providing low delay service within best effort. *IEEE Networks*, 15(3):60–69, May 2001. See also `http://www.abeservice.org`.

[79] N. Hutchinson and L. Peterson. The *x*-kernel: an architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[80] RealNetworks Inc. Real video technical white paper, January 1999. `http://www.realnetworks.com/devzone/library/whitepapers/g2overview.html`.

[81] Intel Corporation. *Pentium Pro Family Developer's Manual. Volume III: Operating System Writer's Guide*, 1995.

[82] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[83] V. Jacobson. Modified TCP congestion avoidance algorithm. Note sent to end2end-interest mailing list, April 1990. `ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail`.

[84] V. Jacobson and S. McCanne. vat: LBNL audio conferencing tool. `http://www-nrg.ee.lbl.gov/vat/`.

[85] V. Jacobson and S. McCanne. vic: A video conferencing tool. `http://www-nrg.ee.lbl.gov/vic/`.

[86] V. Jacobson and S. McCanne. wb: LBNL whiteboard tool. `http://www-nrg.ee.lbl.gov/wb/`.

[87] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB. IETF RFC 2598, June 1999.

[88] V. Jacobson, K. Nichols, and K. Poduri. The "virtual wire" per domain behavior. IETF draft, draft-ietf-diffserv-pdb-vw-00.txt, July 2000.

[89] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM Computer Communication Review*, pages 75–88, July 2002.

[90] R. Jones. *netperf*: a benchmark for measuring network performance - revision 2.0. Information Networks Division, Hewlett-Packard Company, February 1995. See also `http://www.netperf.org`.

[91] J. Kaur and H. M. Vin. Core-stateless guaranteed rate scheduling algorithms. In *Proceedings of INFOCOM 2001*, volume 3, pages 1484–1492, Anchorage, AK, April 2001.

[92] L. Kleinrock. *Queueing Systems. Volume II: Computer Applications*. John Wiley & Sons, New York, NY, 1976.

[93] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[94] H. Kroner, G. Hebuterne, P. Boyer, and A. Gravey. Priority management in ATM switching nodes. *IEEE Journal in Selected Areas in Communications*, 9(3):418–427, April 1991.

[95] A. Kumar, J. Kaur, and H. M. Vin. End-to-end proportional loss differentiation. Technical Report TR-01-33, University of Texas, February 2001.

[96] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. In *Proceedings of ACM SIGCOMM 2001*, pages 123–134, San Diego, CA, August 2001.

[97] M. A. Labrador and S. Banerjee. Packet dropping policies for ATM and IP networks. *IEEE Communications Surveys*, 2(3), 3rd Quarter 1999. `http://www.comsoc.org/pubs/surveys`.

[98] T.V. Lakshman, A. Neidhardt, and T. Ott. The drop from front strategy in TCP and in TCP over ATM. In *Proceedings of IEEE INFOCOM '96*, pages 1242–1250, San Francisco, CA, March 1996.

[99] G. Lapiotis. *Stochastic analysis of joint buffer management and service scheduling in high-speed nodes*. PhD thesis, Polytechnic University, 1999.

[100] R. R.-F. Liao and A. T. Campbell. Dynamic core provisioning for quantitative differentiated service. In *Proceedings of IWQoS 2001*, pages 9–26, Karlsruhe, Germany, June 2001.

[101] J. Liebeherr and N. Christin. Buffer management and scheduling for enhanced differentiated services. Technical Report CS-2000-24, University of Virginia, August 2000.

[102] J. Liebeherr and N. Christin. JoBS: Joint buffer management and scheduling for differentiated services. In *Proceedings of IWQoS 2001*, pages 404–418, Karlsruhe, Germany, June 2001.

[103] J. Liebeherr and N. Christin. Rate allocation and buffer management for differentiated services. *Computer Networks*, 40(1):89–110, September 2002.

[104] A.-M. Lin and J.A. Silvester. Priority queueing strategies and buffer allocation protocols for traffic control at an ATM integrated broadband switching system. *IEEE Journal on Selected Areas in Communications*, 9(9):1524–1536, December 1991.

[105] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM '97*, pages 127–137, Cannes, France, September 1997.

[106] D. Liu, N. Ansari, and E. Hou. QLP: A joint buffer management and scheduling for input-queued switches. In *Proceedings of the IEEE Workshop on High Performance Switching and Routing (HPSR 2001)*, pages 164–168, May 2001.

[107] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real Time Systems*, 23(1–2), July 2002.

[108] Y. Lu, A. Saxena, and T. F. Abdelzaher. Differentiated caching services; A control-theoretical approach. In *Proceedings of the 21st International Conference on Distribute d Computing Systems*, pages 615–624, Phoenix, AZ, April 2001.

[109] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. IETF RFC 2018, April 1996.

[110] V. Misra, W. Gong, and D. Towsley. A fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of ACM SIGCOMM 2000*, pages 151–162, Stockholm, Sweden, August 2000.

[111] J. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15(3):217–252, 1997.

[112] Y. Moret and S. Fdida. A proportional queue control mechanism to provide differentiated services. In *Proceedings of the International Symposium on Computer and Information Systems (ISCIS)*, pages 17–24, Belek, Turkey, October 1998.

[113] T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Barghavan. Delay differentiation and adaptation in core stateless networks. In *Proceedings of IEEE INFOCOM 2000*, pages 421–430, Tel-Aviv, Israel, April 2000.

[114] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. IETF RFC 2474, December 1998.

[115] K. Nichols, V. Jacobson, and L. Zhang. Two-bit differentiated services architecture for the Internet. IETF RFC 2638, July 1999.

[116] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *Proceedings of ACM SIGCOMM '98*, pages 303–314, August 1998.

[117] R. Pan, B. Prabhakar, and K. Psounis. CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE INFOCOM 2000*, volume 2, pages 942–951, Tel-Aviv, Israel, April 2000.

[118] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thirian, F. Tobagi, and C. Diot. Analysis of measured single-hop delay from an operational backbone network. In *Proceedings of IEEE INFOCOM 2002*, volume 2, pages 535–544, New York, NY, June 2002.

[119] A. K. Parekh and R. G. Gallagher. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[120] S. Parekh, N. Gandhi, J.L. Hellerstein, D. Tilbury, T.S. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Journal of Real-Time Systems*, 23(1–2), July 2002.

[121] V. Paxson. Automated packet trace analysis of TCP implementations. In *Proceedings of ACM SIGCOMM '97*, pages 167–179, Cannes, France, August 1997.

[122] J. Postel. Internet protocol. IETF RFC 791, September 1981.

[123] P. Rajvaidya and K. Almeroth. Analysis of routing characteristics in the multicast infrastructure. In *Proceedings of IEEE INFOCOM 2003*, pages 219–230, San Francisco, CA, April 2003.

[124] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. IETF RFC 3168, September 2001.

[125] D. Reed. Ip filter. See `http://coombs.anu.edu.au/~avalon/`.

[126] D. M. Ritchie. A stream input-output system. *AT&T Bell Laboratories Technical Journal*, 63(8):1897–1910, October 1984.

[127] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, January 1997.

[128] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. IETF RFC 3031, January 2001.

[129] S. Sahu, P. Nain, D. Towsley, C. Diot, and V. Firoiu. On achievable service differentiation with token bucket marking for TCP. In *Proceedings of ACM SIGMETRICS 2000*, pages 23–33, Santa Clara, CA, June 2000.

[130] S. Saroiu, P Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of SPIE/ACM Multimedia Computing and Networking (MMCN) 2002*, pages 156–170, San Jose, CA, January 2002.

[131] K. Schittkowski. NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5:485–500, 1986. Edited by Clyde L. Monma.

[132] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in computer networks: reshaping the research agenda. *ACM Computer Communication Review*, 26(2):19–43, April 1996.

[133] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. IETF RFC 2212, September 1997.

[134] S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements. IETF RFC 2215, September 1997.

[135] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, June 1996.

[136] J.-J. E. Slotine and W. Li. *Applied nonlinear control*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[137] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *Proceedings of ACM SIGCOMM '99*, pages 135–146, Boston, MA, August 2001.

[138] R. Stallman. *Using and Porting the GNU Compiler Collection (gcc)*. iUniverse Inc., 2000.

[139] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Reading, MA, 1998.

[140] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, l. Zhang, and V. Paxson. Stream control transmission protocol. IETF RFC 2960, October 2000.

[141] 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 Transactions on Networking*, 11(1):33–46, January 2003.

[142] I. Stoica and H. Zhang. Providing guaranteed services without per-flow management. Technical Report CMU-CS-99-133, Carnegie Mellon University, May 1999.

[143] I. Stoica and H. Zhang. Providing guaranteed services without per-flow management. In *Proceedings of ACM SIGCOMM '99*, pages 81–94, Boston, MA, August 1999.

[144] I. Stoica, H. Zhang, and T. S. E. Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. In *Proceedings of ACM SIGCOMM '97*, pages 249–262, Cannes, France, August 1997.

[145] I. Stoica, H. Zhang, and S. Shenker. Self-verifying CSFQ. In *Proceedings of IEEE INFOCOM 2002*, volume 1, pages 21–30, New York, NY, June 2002.

[146] A. Striegel and G. Manimaran. Packet scheduling with delay and loss differentiation. *Computer Communications*, 25(1):21–31, January 2002.

[147] B. Suter, T.V. Lakshman, D. Stiliadis, and A.K. Choudhury. Buffer management schemes for supporting TCP in gigabit routers with per-flow queueing. *IEEE Journal on Selected Areas of Communications*, 17(6):1159–1170, September 1999.

[148] Cisco Systems. Distributed WRED. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.htm`.

[149] N. Taft, S. Bhattacharyya, J. Jetcheva, and C. Diot. Understanding traffic dynamics at a backbone POP. In *Proceedings of SPIE ITCOM Workshop on Scalability and Traffic Control in IP Networks*, number 4526, Denver, CO, August 2001.

[150] T. Turletti. The INRIA videoconferencing system (IVS). *ConneXions - The Interoperability Report Journal*, 8(10):20–24, October 1994.

[151] J. Wang, S. Patek, H. Wang, and J. Liebeherr. Traffic engineering with AIMD in MPLS networks. In *Proceedings of PfHSN 2002*, pages 192–210, Berlin, Germany, April 2002.

[152] D. E. Wrege, E. W. Knightly, H. Zhang, and J. Liebeherr. Deterministic delay bounds for VBR video in packet-switching networks: fundamental limits and practical trade-offs. *IEEE/ACM Transactions on Networking*, 4(3):352–362, June 1996.

[153] J. Wroclawski. Specification of the controlled-load network element service. IETF RFC 2211, September 1997.

[154] I. Yeom and A. Narasimha Reddy. Modeling TCP behavior in a differentiated services network. *IEEE/ACM Transactions on Networking*, 9(1):31–46, February 2001.

[155] Q. Yin and S. Low. On stability of REM algorithm with uniform delay. In *Proceedings of IEEE GLOBECOM 2002*, pages 2649–2653, Taipei, Taiwan, November 2002.

[156] H. Zhang and D. Ferrari. Rate-controlled static-priority queueing. In *Proceedings of IEEE INFOCOM '93*, volume 1, pages 227–236, San Francisco, CA, March 1993.

[157] L. Zhang. Virtual clock: A new traffic control algorithm for packet switched networks. *ACM Transactions on Computer Systems*, 9(2):101–125, May 1991.