PROPORTIONAL DIFFERENTIATED SERVICES FOR THE INTERNET

by

Konstantinos Dovrolis

PROPORTIONAL DIFFERENTIATED SERVICES FOR THE INTERNET

by

Konstantinos Dovrolis

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN - MADISON

2000

To my family

$Μαρινος, Καιτη, Αννετα, Τασος$

PROPORTIONAL DIFFERENTIATED SERVICES FOR THE INTERNET

Konstantinos Dovrolis

Under the supervision of Professor Parameswaran Ramanathan

At the University of Wisconsin-Madison

Internet applications and users have diverse Quality-of-Service (QoS) requirements, making the current *common-service-for-all* paradigm inadequate and limiting. It is widely agreed that the Internet architecture should offer some type of *service differentiation*, so that some traffic classes get better QoS than others. This dissertation focuses on *relative* service differentiation. This is a scalable and simple network architecture, because it does not require per-flow state in the routers, admission control, signalling, static routing, or resource reservations. In this context, we develop the *Proportional Differentiated Services (PDS)* architecture. In PDS, the differentiation between classes is *controllable*, allowing the network provider to adjust the QoS spacing between classes, and *predictable*, providing higher classes with better service than lower classes independent of the load conditions.

We first consider the issue of delay differentiation, and the related packet scheduling problem. The proposed *Proportional Delay Differentiation (PDD)* model constrains the average delay ratios between classes. We design three scheduling algorithms for the PDD model. The Proportional Average Delay (PAD) scheduler meets the PDD model when it is feasible; PAD, however, is not predictable in short timescales. The Waiting Time Priority (WTP) scheduler approximates closely the PDD model even in short timescales, but only in heavy load conditions. The Hybrid Proportional Delay (HPD) scheduler combines the PAD and WTP features.

We then consider the issue of loss differentiation, and the related packet dropping problem. The *Proportional Loss Differentiation (PLD)* model constrains the loss rate ratios between classes. We design and evaluate two dropping algorithms for the PLD model. The two droppers, PLR(∞) and PLR(M), differ in the interval over which the loss rates are measured. This

difference causes a trade-off between the two droppers, in terms of implementation complexity, accuracy, and adaptability to varying class loads.

Users with an absolute QoS requirement can dynamically search for the minimum class that meets that QoS. We investigate this *Dynamic Class Selection (DCS)* framework in the context of proportional delay differentiation. We examine whether an acceptable class exists for each user, and show the properties of the resulting distributed DCS equilibria. Simulations provide further insight in the dynamic behavior of DCS, and in the factors that determine the well-provisioning of a network.

Finally, we consider the *class provisioning* problem. The network provider, in this case, knows the rate and average delay requirement of each traffic type in a link. The objective is to compute the required link capacity and the appropriate parameters of the PDD model that meet these requirements. We give a methodology that meets this objective.

Acknowledgements

The last five years were an amazing journey. A journey through hard challenges and unique experiences. Looking back, I remember days of discovery and excitement, sleepless nights of hard work, and moments of satisfaction every time something would be completed. I can also remember days of disappointment and discouragement when I was unable to proceed with a research problem, or when obstacles of various kinds were showing up on the way. It seems now that those ‘darker’ days were perhaps the most valuable, as I learned from them my many weaknesses and mistakes.

I had the great fortune to be guided in this journey by Professor Parmesh Ramanathan. Prof. Ramanathan was not simply an academic or research advisor; he was a true *mentor*, with the special meaning of this word in both the Indian and Greek vocabularies. My memories from the hundreds of hours that we spent together are still too vivid to isolate one or another of the many ways in which he helped me. I only hope that I will manage to become a mentor like him in the years to come.

I am also grateful to a number of people that offered significant help and advice in the last five years. Professor Apostolos Dollas, my undergraduate advisor at the Technical University of Crete, helped this journey to start. Dimitrios Stiliadis was my advisor during the summer of 1998, when I was a summer intern at Bell Labs. It was then that I started working on the subject of this thesis, and Dimitrios helped me significantly to better understand the problem and come up with the first results. I hope that I will have the chance to work with him again in the future. During the summer of 1999, as a summer intern at CAIDA, I was also fortunate to work with Kimberly Claffy and David Moore, and to meet Professor Evi Nemeth that helped me significantly later on.

I would also like to thank the members of my thesis committee, Mary Vernon, Lawrence Landweber, Mikko Lipasti, Kewal Saluja, and David Wood. Professor Vernon took a genuine interest in this thesis, and pointed out an important omission in our original work on DCS. Professor Saluja provided valuable comments on this dissertation, and wise advices throughout my graduate studies. I was also fortunate to take two computer networking courses from Professor Landweber. His unique teaching talent and his great experience on how the Internet works had a major impact on my 'vision' about the future Internet.

This work would not be completed without the support of our sponsors. Specifically, the National Science Foundation (grant No. MIP-9526761), the USENIX association, and Cisco Systems Inc, helped me with their support to pursue this research with absolute independence.

In February of 1998, I met Lori Ring and her three wonderful children Chaz, Coddye, and Chelsea. These four people became my best friends in Madison, and with all their love and care, a second family. I will never forget the nights that Lori was bringing me supper at the office, or all her efforts to make me live a happy and 'normal' life, even in the most stressful moments of the graduate program.

Finally, and most importantly, I thank my family in Greece, whose love and support in every moment of my life have been 'the wind beneath my wings'. This dissertation is dedicated to them.

Contents

List of Figures

Acronyms

ABE: Asymmetric Best Effort (scheduler)

ADD: Additive Delay Differentiation (scheduler)

ADD: Average Drop Distance (dropper)

ATM: Asynchronous Transfer Mode

BPR: Backlog Proportional Rates (scheduler)

CBP: Complete Buffer Partitioning (dropper)

CBQ: Class Based Queueing (scheduler)

COP: Class Operating Point

CSC: Class Selector Compliant (PHBs)

CSV: Class Selection Vector

CT: Cross Traffic

DCS: Dynamic Class Selection

DDP: Delay Differentiation Parameter

DiffServ: Differentiated Services (architecture)

DSCP: Differentiated Services Code Point

D-WFQ: Dynamic Weighted Fair Queueing (scheduler)

EDD: Earliest Due Date (scheduler)

FCFS: First Come First Served (scheduler)

FEC: Forward Error Correction

FTP: File Transfer Protocol

GPS: Generalized Processor Sharing (scheduler)

HPD: Hybrid Proportional Delays (scheduler)

H-PFQ: Hierarchical Packet Fair Queueing (scheduler)

HTTP: HyperText Transport Protocol

IETF: Internet Engineering Task Force

IntServ: Integrated Services (architecture)

IP: Internet Protocol

JoBS: Joint Buffer Management and Scheduling (scheduler/dropper)

LDP: Loss Differentiation Parameter

LHT: Loss History Table

LIFO: Last In First Out (dropper)

LOPD: Local Optimal Proportional Delays (scheduler)

MDP: Mean Delay Proportional (scheduler)

MPLS: Multi-Protocol Label Switching

mRED: multi-class Random Early Detection (dropper)

PAD: Proportional Average Delays (scheduler)

PBS: Partial Buffer Sharing (dropper)

PDD: Proportional Delay Differentiation (model)

PDS: Proportional Differentiated Services (architecture)

PFE: Packet Forwarding Engine

PHB: Per-Hop Behavior

PLD: Proportional Loss Differentiation (model)

PLR: Proportional Loss Rate (droppers)

PMP: Paris Metro Pricing (architecture)

PQCM: Proportional Queue Control Mechanism (scheduler)

QoS: Quality of Service

RED: Random Early Detection (backlog controller)

RIO: RED In/Out (dropper)

RSVP: Resource Reservation Protocol

RTCP: Real-Time Control Protocol

RTD: Round-Trip Delay

SCORE: Scalable Core (architecture)

SP: Strict Priorities (scheduler)

TCP: Transport Control Protocol

USD: User Share Differentiation (architecture)

VLL: Virtual Leased Line (architecture)

WDM: Wavelength Division Multiplexing

WFQ: Weighted Fair Queueing (scheduler)

WTP: Waiting Time Priorities (scheduler)

WWW: World Wide Web

Chapter 1

Introduction

1.1 Background

Packet networks can transfer a wide range of information types, such as E-mail, WWW pages, voice, music, video, and others. In general, if an information unit can be digitally represented, it can also be assembled into *packets* and transferred through a network. The Internet, which is an interconnection of many different networks using the Internet Protocol (IP), was mainly used for E-mail and file transfers in the eighties, for WWW access in the nineties, and more recently, for multimedia streaming and conferencing as well. The ever-increasing diversity in the data types that can be carried over a packet network is a major reason for the unprecedented success of the Internet.

The information transfer in a packet network is accomplished through a sequence of *links* and *routers* from the sender to the receiver(s). The links serve as direct connections between end-users or routers. The routers serve as 'junction' points where packets from an input link are transmitted, or *forwarded*, to one or more output links. The path and the service that packets receive in a network is determined by information stored in the *packet headers*, such as the source or destination addresses and the application port numbers.

Figure 1 shows the basic architectural components of a conventional IP router [53]. Packets arrive at input links (*input interfaces*) and depart from output links (*output interfaces*). A routing protocol creates the routing table, which specifies the output interface(s) that a packet should be sent to. A *packet switch*, which can be a shared bus, a multi-ported memory bank, or a switching fabric, is used to convey packets from input interfaces to output interfaces [80]. Packets

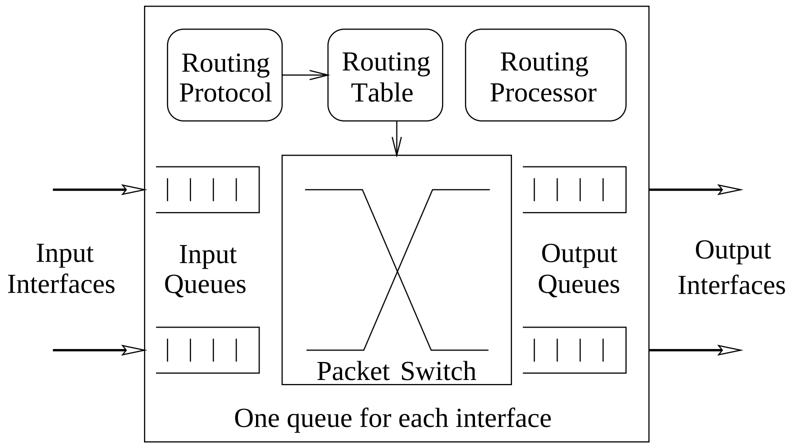


Figure 1: Architectural layout of a conventional IP router.

can arrive at the input interfaces in a higher rate than the switch or the output interfaces can forward. Additionally, packets destined to the same output interface can arrive simultaneously from different input interfaces. These scenarios can cause *bandwidth contention* in the router. Bandwidth contention is dealt with packet buffers that temporarily store (or *queue*) the pending packets until they can be forwarded. If the number of available packet buffers is not adequate, the router also experiences *buffer contention*, and some packets need to be *dropped*. Other types of resource contention in a router can be caused in the main router CPU or in the 'routing-processor', and they are also dealt with packet buffering and dropping.

An important point in this architectural layout of an IP router is that packet queues are inevitable, and they can cause *delays* and *packet losses*. These two effects are the major *performance degradation factors* in packet networks. Since the queueing delays and losses are caused by individual routers or links, they are considered *local*, or *per-hop*, performance factors. A second important point is that the two main *resource types* in a router are the rate with which it can forward and transmit packets (usually called 'bandwidth') and the packet buffers in the router queues. The performance that traffic experiences in a router degrades when there is contention for either of these resources.

In the rest of this chapter, we introduce the notion of *service differentiation* in packet networks. We first discuss the current Internet service model, and highlight its *common-service-for-all* paradigm. Then, we present *IntServ* and *DiffServ*, which are the two major architectures proposed for introducing service differentiation in the Internet. IntServ is based on *flows*, *end-to-end resource reservations*, and *admission control.* DiffServ is based on *traffic aggregation*, *per-hop differentiation*, and *resource provisioning.* This dissertation focuses on the DiffServ architecture. Two major proposals in DiffServ are the *Virtual Leased Line (VLL)* and *Assured* service models. These two models, as well as some more recent works, are reviewed. Our major contribution is to develop an original architecture, called *Proportional Differentiated Services (PDS).* The chapter closes with a summary of the PDS architecture and with an overview of the dissertation.

1.2 The *'Common-Service-For-All'* paradigm

Not all applications have the same QoS requirement from the network. An interactive video-conference session requires lower delays and less losses than a file transfer with FTP. Similarly, the access of a Web page should be completed faster than the transfer of an E-mail message. Additionally, not all users need the same performance. Some individuals and companies use the Internet for important transactions, and rely on it for their everyday life and business. The central point in this discussion is that some degree of *service differentiation* would be beneficial in packet networks, so that the more demanding applications or users get a better performance than the rest. Such a service differentiation capability, of course, would need to be accompanied in practice with some pricing or policy rules, according to which a better service level either costs more, or is limited to certain traffic types or users.

The current Internet does not offer, at least in its vast majority, any service differentiation. This is reflected in Figure 1 by the fact that there is *only one queue* in each input or output interface. In other words, *all packets that follow the same path are treated in the same way, being queued or dropped at the same contention points, independent of the application or the user that generates them.* We refer to this effect as the *common-service-for-all paradigm* of the current Internet. To illustrate the deficiencies that it causes, Figure 2 shows a generic *QoS requirement*

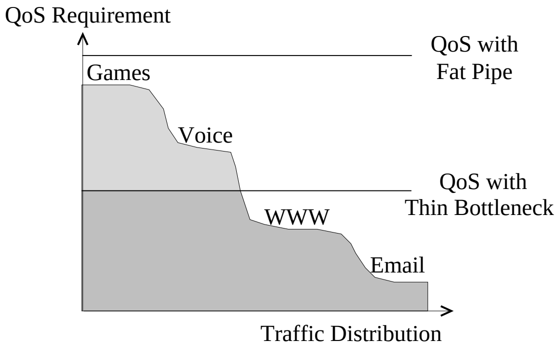


Figure 2: A QoS requirement curve for a network link that services four traffic types.

curve for a network link that services four traffic types. Say that about 20% of the traffic is E-mail or FTP with very low QoS requirements in order to perform well, 35% is WWW traffic with somewhat higher requirements, 20% is voice traffic with interactive requirements, while the rest of the traffic is generated from applications like distributed interactive games that are very demanding. Even though these applications have diverse performance expectations, they would get the same treatment in the router queues, and so they would receive the same QoS in today's Internet.

If the router has enough forwarding resources, it can provide an adequate performance even to the most demanding applications and users. This type of *over-provisioned* routers or links are referred to in practice as *fat pipes*, since they have enough resources to satisfy the requirements of the entire traffic mix. In the example of Figure 2, the fat pipe would offer the stringent QoS of the 'Games' traffic type to all packets, even if they need a much lower performance. The problem with fat pipes is that, from an economic point of view, they do not utilize the network resources efficiently. This is especially important for networks in which the forwarding resources are costly, such as transcontinental links, satellite connections, or access links that are priced based on the bandwidth of the interconnection.

On the other hand, if the router or link does not have enough forwarding resources, it can only provide adequate performance to the less demanding traffic. This type of *under-provisioned* routers and links are referred to in practice as *thin bottlenecks*. In the example of

Figure 2, the thin bottleneck only meets the QoS requirement of the E-mail and WWW traffic types, preventing the more demanding applications from performing well.

In summary, the fact that current routers follow the *common-service-for-all* paradigm leads to either inefficient network operation, or to inadequate performance for some applications and users. Because of this issue, there is a wide consensus that the current Internet service model should be replaced with an architecture that can offer different QoS to different traffic types or users [41]. The desired characteristics or objectives of this new service model, however, are not yet clear. The following two sections describe the two major architectures that have been developed within the Internet Engineering Task Force (IETF) [49] for addressing this issue: the Integrated Services architecture (or *IntServ*), and the Differentiated Services architecture (or *DiffServ*).

1.3 Integrated Services architecture

The *Integrated Services* (*IntServ*) architecture was developed within the IETF in the mid-nineties [15, 118, 115], and has its roots in earlier research results [5, 79, 27, 28, 38]. The basic premise in IntServ is that applications have hard network performance requirements, and they cannot operate effectively unless these requirements are met. For instance, an IP telephony application may require a maximum end-to-end delay of 200msec for each packet, while a video streaming application may need a packet loss rate that is less than 10^{-3}. In the IETF terminology, the IntServ architecture is mainly appropriate for *intolerant* or *unelastic* applications, i.e., applications that do not tolerate or adapt to a performance level that is lower than what they ask for.

Figure 3 shows the major architectural blocks of an IntServ-capable router. The service models that have been discussed or standardized in the IntServ architecture include a maximum end-to-end delay [98], a sufficiently low loss rate [117], and a minimum end-to-end rate [3]. Even though these service models differ in their implementation or usage, they are all based on the following IntServ architectural principle: *the service differentiation is accomplished with per-flow end-to-end resource reservations.* These three terms ('per-flow', 'end-to-end', and 'resource

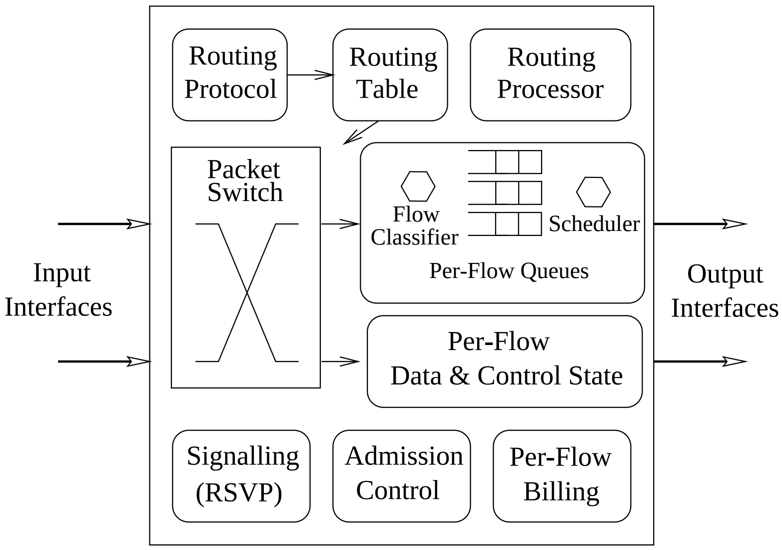


Figure 3: Architectural layout of an IntServ router.

reservations') have important implications on the strengths and the weaknesses of IntServ, and are discussed next.

First, IntServ is *flow-centric*, meaning that the service differentiation router mechanisms apply to individual packet flows, i.e., streams of packets that belong to the same application session. IntServ requires that routers perform the following operations on the packets of a flow:

- *flow classification*, to identify the flow in which a packet belongs to [57, 100],
- *scheduling*, to provide a certain delay deadline or rate to a flow [101, 102],
- *buffer management*, to allocate a number of buffers to a flow [107], and
- *traffic shaping or policing*, to control certain traffic characteristics (e.g., maximum burstiness) of a flow [20, 101].

The previous operations must be performed for all active flows. In addition to the *data-plane per-flow state*, which summarizes the above operations, a router has to also maintain and process information for each flow at the *control-plane*. The control-plane in IntServ is responsible for resource reservations and signalling operations (discussed next), but also for per-flow accounting [37] and policy control [86].

Second, IntServ is an *end-to-end* architecture, and so it requires the cooperation between network providers in order to provide an end-to-end service guarantee. Suppose, for instance, that a network path between two users goes through networks X, Y, and Z. If one of these networks does not support the service type that a user requests, the service cannot be provided in an end-to-end basis. The Internet has a highly hierarchical structure, with major backbone providers connecting smaller regional networks, which then connect even smaller local networks, and so on. In such a hierarchical structure, establishing the required *multilateral agreements* in order to enable end-to-end services has been proven to be quite difficult in practice [41].

Third, IntServ is based on per-flow *resource reservations*, meaning that a certain amount of forwarding resources (bandwidth and buffers) has to be reserved for a flow before the session starts and for the duration of the session. In order for resource reservations to be accomplished in a network, a *signalling protocol* is also required. The signalling protocol conveys the application QoS requirements and traffic characteristics from the end-hosts to the routers along the flow's path. If the resources do not suffice for the new flow in any of the routers, the signalling protocol notifies the end-hosts that the flow request has been denied. This process is called *admission control* [52, 62]. The signalling protocol that has been designed for the IntServ architecture is the *Resource Reservation Protocol (RSVP)* [123, 16]. The significant per-flow processing that RSVP requires at the control-plane raised concerns about the IntServ scalability [97]. Recent measurements with a commercial RSVP implementation, however, showed that the RSVP overhead may not be so important [73].

Despite the fact that IntServ is able to provide strict QoS guarantees to individual flows, its deployment by network providers has been quite limited. In addition to the issues of inter-domain deployment and RSVP overhead, discussed earlier, there are also *scalability* and *manageability* issues in IntServ [64]. The scalability concerns are raised because IntServ requires that routers maintain and process data and control state for every active flow in the router. Gigabit or terabit links carry millions of simultaneously active flows, making it difficult to build IntServ-capable routers. Next generation routers may be able to accommodate millions of flows by maintaining per-flow state only at the edge routers [106, 105], giving to the term 'flow' a more coarse granularity [56], or using approximations of the ideal algorithms [99].

The manageability concerns are raised because of the conventional wisdom that an IntServ network is harder to install, debug, and operate. This is probably true, given that IntServ requires admission control, signalling, per-flow accounting, and the configuration of several router mechanisms. Another important issue is that routes can dynamically change in an IP network. Routing changes may not be a frequent event in the Internet as a whole, but there are certain links in which they occur quite often [81]. If a certain end-to-end QoS is guaranteed to a flow, the architecture should be able to either forbid routing changes for that flow (*route-pinning*) [46], or to reserve the required forwarding resources, while the session is in progress, in the flow's new path. Both these operations are hard to implement in practice. Similar complexities arise when IntServ flows need fault tolerance [32].

Other factors that have contributed to the weak deployment of IntServ is that it requires new Application Programming Interfaces (APIs) at the end-hosts, especially for multimedia applications [45], and that it can provide true end-to-end service guarantees only if similar resource reservation mechanisms are also deployed in the servers [8], and the end-host operating systems [120]. The combination of all these issues led the IETF and the research community to consider simpler and more scalable service differentiation architectures, such as the proposals discussed in the next section.

1.4 Differentiated Services architecture

The *Differentiated Services (DiffServ)* architecture was developed within the IETF in the late nineties [74, 9], and in several aspects it is still work-in-progress. The initial objective in DiffServ was a more scalable, manageable, and easily deployable architecture for service differentiation in IP networks. The major DiffServ premise is that individual flows, or *microflows*, with similar QoS requirements can be *aggregated* in larger traffic sets, called *macroflows*. All packets in a macroflow receive the same 'forwarding behavior' in routers. So, a macroflow is the minimum level of granularity in a DiffServ network in terms of service differentiation. Each macroflow

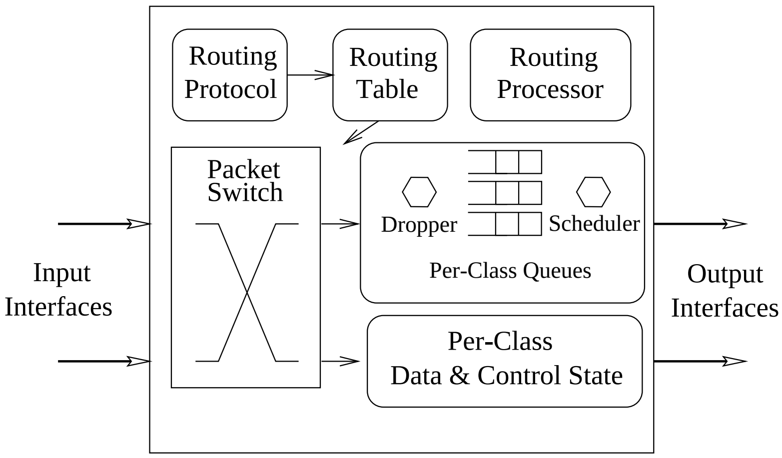


Figure 4: Architectural layout of a DiffServ router.

uses a certain service class, or *Per-hop Behavior (PHB)*. A PHB is identified by a short label (currently six bits) in the IPv4 or IPv6 header, which is called *Differentiated Services Code Point (DSCP)*.

Microflows are aggregated into macroflows at the *edges* of a DiffServ network. The 'edge' can be the host-network interface. In that case, the operating system or the application determines the macroflow that the flow belongs to. Or, the 'edge' can be a router that connects a microflow-aware network to a DiffServ network. The mapping from microflows to macroflows requires flow classification [100]. This is a relatively expensive operation, but since it is only performed at the network edges, where the number of microflows is much smaller, it is expected to not cause scalability problems. Microflows are aggregated in macroflows based on rules set by the network operator. For instance, a macroflow can be all the traffic originating or destined to a certain organization, the traffic that is sent/received by a certain host, or all the traffic of a certain application.

Figure 4 shows the main architectural components in a DiffServ router. The first benefit that aggregation provides is that it addresses the scalability issue, since state is only required for a few service classes. Note in Figure 4 that there is a queue for each class, instead of a queue for each flow. If the number of classes is small, say a few tens, the per-queue operations of classification, scheduling, buffer management, or shaping/policing, become significantly simpler and faster. Second, aggregation simplifies the network management, since the operator needs to only monitor and control the service level of a few classes, rather than of millions of flows. Third, aggregation makes the network pricing/accounting simpler, since users do not have to be billed for each session or microflow, but for the overall traffic they generate in each service class. Obviously, the fundamental drawback of aggregation is that the network cannot guarantee a certain QoS to a microflow.

Besides exploiting the benefits of aggregation, the DiffServ architecture also makes other simplifications. First, the PHBs have *strictly local (per-hop) semantics*. So, they can provide service differentiation even if they are deployed only at individual congestion points, without requiring deployment in an entire network or across different domains. For example, a network provider can decide that the entire network will be over-provisioned, without using any service differentiation mechanisms, and deploy DiffServ only at the transoceanic links where congestion is acceptable due to the high bandwidth cost.

Second, *DiffServ does not require a signalling protocol*, since there are no resource reservations for individual flows. In some proposals, though, such as the *VLL* service (§1.5.1), forwarding resources need to be reserved for a macroflow during a certain time period. Instead of a signalling protocol, the VLL proposal uses a *bandwidth broker*, which is a centralized agent controlling the resource reservations in a network. Other proposals, such as the *Assured* service (§1.5.2), require careful *resource provisioning*. Provisioning refers to determining the necessary amount of forwarding resources for a service class over a large time period (say hours to days) based on the *expected traffic load* in that class. The limitations of bandwidth brokers and provisioning are discussed in Section §1.5.

Third, DiffServ allows both *absolute* (or *quantitative*) and *relative* (or *qualitative*) service differentiation. An absolute service model aims to provide a macroflow with a quantitative performance level, such as a minimum forwarding rate or a maximum loss rate, at a certain link or network path. Examples of absolute services are the VLL and the Assured service proposals. These service models require 'semi-static' bandwidth reservations or provisioning, as mentioned in the previous paragraph. An absolute service model also requires some form of admission control or policing, in order to prevent users from sending traffic at a higher rate than their network contract allows.

Relative services, on the other hand, provide a number of classes with increasing performance. *The network simply guarantees that higher classes will provide better QoS than lower classes* [33]. The exact QoS in each class is not specified, and it depends on the load conditions and the service differentiation mechanisms that the network deploys. If the applications and users have absolute QoS requirements, they can dynamically search for the class that best meets their QoS and pricing constraints. The relative differentiation model does not require resource reservations, admission control, signalling, or fixed routing, and so it is considered as simpler to manage and deploy.

Summarizing the last two sections, Table 1 shows the main differences between the IntServ and DiffServ architectures.

1.5 Previous work on differentiated services

The two DiffServ models that received most attention so far are the *Virtual Leased Line (VLL)* service [51], and the *Assured* service [24]. We briefly present them next, discussing also their limitations. Then, we review some more recent proposals on service differentiation.

1.5.1 Virtual Leased Line (VLL) service

The VLL service, also called *Premium* service, was proposed by Van Jacobson in 1997 [51], and it is widely considered today as the DiffServ starting point. The VLL service aims to implement a *guaranteed peak bandwidth service* with negligible queueing delay and losses, similar to the service that a user would get with a leased line in a circuit-switched network. The VLL traffic is limited at the network ingress to its contracted peak rate, say R, using traffic shaping at the edge routers. The shapers guarantee that packet bursts are not injected into the network, and that the VLL traffic never exceeds the rate R at the network ingress. In the network core, the

	Integrated Services	**Differentiated Services**
Granularity of service differentiation	Microflows	Macroflows (traffic aggregates or classes)
State in routers (scheduling, buffer management, etc)	Per-flow	Per-class
Traffic classification basis	Several header fields	DSCP field (6 bits) in IPv4/v6 header
Admission control	Required	Required for absolute differentiation only
Signalling protocol	Required (RSVP)	Not required for relative schemes. Absolute schemes need semistatic reservations or broker agents.
Coordination for service differentiation	End-to-End	Local (per-hop)
Scalability	Limited by number of flows	Limited by number of classes
Network accounting	Based on flow characteristics and QoS requirement	Based on class usage
Network management	Similar to circuit-switched networks	Similar to existing IP networks
Interdomain deployment issues	Multilateral agreements needed	Bilateral agreements needed

Table 1: A comparison of the IntServ and DiffServ architectures.

rate R is reserved for the VLL macroflow along its path, and additionally the VLL traffic is serviced with the highest priority. The original expectation was that the VLL macroflows would not experience any significant queueing delays or losses.

Recent research, however, has shown that these expectations are not always met. The VLL traffic, even if it is shaped at the network edge and serviced with the highest priority in the network core, can still become bursty as it flows through the network. The reason is that multiplexing of VLL traffic from different input interfaces in the core routers can create packet bursts [21], causing queueing delays and potentially losses. The magnitude of this burstiness depends on the VLL load, the hop count of a VLL macroflow, and the shaping parameters at the network ingress. This implies that in order to *guarantee* reasonably low queueing delays, the VLL load must only be a small fraction of the network capacity [105]. Experimental results have shown that, under certain conditions, it is indeed possible that the VLL service can experience significant queueing delays and losses [42]. More recently, [47] investigated how to address these VLL issues with *re-shaping* at the boundaries between network domains. Re-shaping, though, requires information about the microflows that constitute the VLL macroflow, and so it may not be a feasible operation in high-speed links.

The VLL service requires some form of semi-static bandwidth reservations that a 'bandwidth broker' protocol or agent has to setup in each domain [51]. Interdomain reservations have to be agreed upon and coordinated between the individual bandwidth brokers of each domain [111]. The requirement for a centralized bandwidth broker in each domain causes concerns about the scalability and fault-tolerance of the architecture, while distributed broker architectures raise consistency issues that are not yet solved [105]. The VLL service also requires some form of *route-pinning* for holding the VLL traffic in the links where the bandwidth reservations have been setup, despite of route changes that can occur in IP networks. Despite these issues, the VLL service is the most popular DiffServ model today, and there are routers that support it. Also, there has been some experimentation with VLL in the QBone [110] and in other networks. These efforts are still in progress.

1.5.2 Assured service

The *Assured* service was originally proposed by David Clark in [25], under the name *Expected Capacity framework*, and it was later refined in [75] and [24]. The basic idea in the Assured service is quite simple. A user purchases from the network a certain bandwidth 'profile', say R. As long as the traffic that the user generates has a lower rate than R, it is marked as *IN*; otherwise, it is marked as *OUT*. In times of congestion, the network drops OUT traffic with a significantly higher probability than IN traffic. Consequently, users are allowed to obtain a higher throughput than their profile R in times of low network load, but they are limited to their IN traffic in times of congestion. How does the network guarantee that the IN traffic will not be dropped, and that each user will get the contracted bandwidth profile? The original conjecture was that if the network is sufficiently *provisioned*, the IN traffic should not be dropped. 'Provisioning', here, means that the network operator keeps track of the profiles of different users *and* of the routes that the IN traffic goes through, in order to reserve an adequate bandwidth capacity in each network link. The Assured service was further studied in [24], in the context of TCP transfers, focusing on specific marking procedures for TCP traffic.

Recent results, however, have shown that it is difficult to design provisioning algorithms that achieve simultaneously good service quality with large spatial granularity and high resource utilization [104]. Some form of route-pinning is also necessary to implement the Assured service, since the provisioning procedure assumes fixed routing and steady load in each network link. In the context of TCP transfers using the Assured service, [121] and [93] showed that under certain conditions it is impossible to provide a certain throughput to a TCP connection, independent of how well-provisioned the network is. The Assured service has been also analytically studied in [66] and [92] with simple queueing models, attempting to quantify the 'assurance level' of the provided bandwidth profiles.

1.5.3 Other proposals

The early works in the context of differentiated services focused on the VLL and Assured service models. More recently, however, there has been a large interest in other service models as well. Some of these proposals are based on the DiffServ principle of flow aggregation, while others attempt to provide per-flow service guarantees (as in IntServ) but without maintaining per-flow state in the core routers.

The Asymmetric Best Effort (ABE) architecture [14] provides two service classes: one for delay-sensitive applications (such as IP telephony) and another for throughput-sensitive applications (such as data transfers). The User Share Differentiation (USD) scheme, proposed in [114], guarantees that the per-hop distribution of bandwidth is performed proportionally to the profile that each user purchases. USD is an example of relative differentiation. A similar relative service model has been studied in [4]. A refinement of the Assured service appeared in [40], proposing that the IN/OUT marker should adapt to the measured flow throughput.

A simple case of relative service differentiation is the Paris Metro Pricing (PMP) scheme [76]. PMP is based on pricing, instead of special router forwarding mechanisms, to provide relative service differentiation. It is based on the assumption that larger prices for the higher classes will lead to lower loads, and thus, better performance. Pricing mechanisms, however, can only be effective over relatively long timescales, especially when the class tariffs cannot be frequently modified. When higher classes become overloaded (because, for example, many 'rich' users become active at the same time), they will offer worse performance than lower classes. This would be an instance of *inconsistent* or *unpredictable* relative differentiation, as we discuss in §2.3.

The *SCORE (Scalable Core)* architecture provides per-flow service guarantees, as in IntServ, but without per-flow state in the core routers [106]. In SCORE, the QoS-related information is carried in the packet headers (similar to the MPLS technology [113]), instead of being stored in the routers. This architecture can provide fair-queuing [106] or guaranteed delay [105] services. SCORE requires per-flow state only at the ingress routers. When a flow goes through multiple domains however, which is the common case in the Internet, the *border routers* between networks should also maintain per-flow state, which may be a scalability problem in high-bandwidth links. In a similar architecture, [18] proposed a scheme in which resource management and admission control are performed only at egress routers, without any coordination between backbone routers and per-flow state in the core.

1.6 Thesis scope and overview

The broad subject of this dissertation is *service differentiation in packet networks*. Even though most of our contributions are applicable to any packet network technology, we focus on IP networks, and especially on the Internet, as the major IP-based interconnection of packet networks today. Within the broad subject of service differentiation, *we focus on the DiffServ architecture*, which is based on a few classes of service, local differentiation, and provisioning, rather than on the more 'heavy-weight' IntServ architecture of per-flow and end-to-end resource reservations. In most of this thesis we consider the *relative differentiation* model, which is more scalable and simpler to deploy than the absolute differentiation model, because it does not require admission control, signalling, bandwidth brokers, or static routing.

Having limited the scope and focus of this dissertation on relative differentiation, we propose and study an original architecture, called *Proportional Differentiated Services (PDS)*. PDS refines the basic premise of relative differentiation in two ways. First, it makes the differentiation between classes *controllable*, allowing the network provider to *adjust the performance spacing between classes*. This is achieved with simple *proportional constraints* on the average delays and loss rates between classes. Second, it makes the differentiation between classes *predictable*, providing the assurance to users and applications that higher classes will lead to better performance, independent of the load conditions.

As shown in Figure 5, the PDS architecture consists of three components: router mechanisms for packet scheduling and dropping, user/application dynamic class selection algorithms, and class provisioning methodologies. These components, in more detail, are:

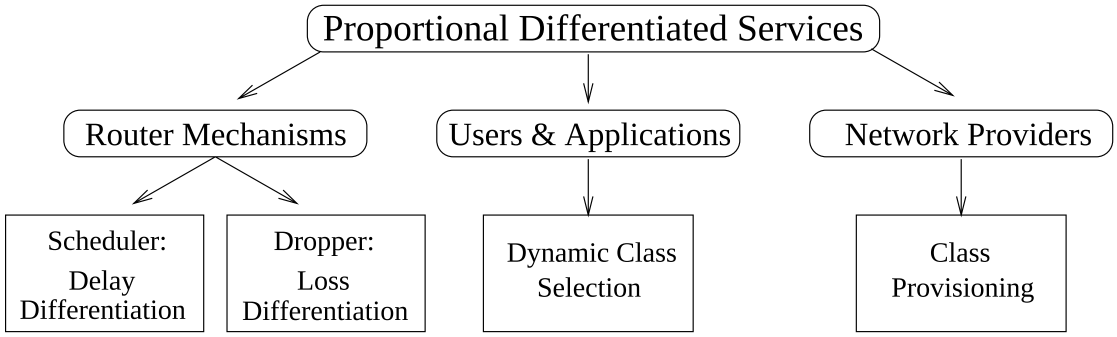


Figure 5: The structure of the PDS architecture.

1. *Router mechanisms and PDS*: the routers have to use special forwarding mechanisms for providing proportional delay and loss rate differentiation. These mechanisms are the scheduler and the dropper, respectively.

2. *Users/Applications and PDS*: the users and applications that need an absolute QoS can perform dynamic class selection over a PDS network in order to dynamically select the minimum acceptable class that meets their requirements.

3. *Class provisioning and PDS*: the network operator can use the PDS architecture in order to provision an absolute QoS level in each class. The class provisioning methodology determines the class differentiation parameters and the capacity of a link.

The structure of this dissertation is as follows. The general framework of relative differentiated services, the requirements for controllability and predictability, and the proportional differentiation model are presented in Chapter 2. The implementation of the PDS architecture requires the design of special scheduling and buffer management router mechanisms. The scheduler in the PDS context has to provide proportional delay differentiation, which is the subject of Chapter 3. Similarly, the packet dropper in the PDS context has to provide proportional loss rate differentiation, which is the subject of Chapter 4. Chapter 5 has two parts. In the first part, we study a dynamic class selection algorithm that users and applications can use to meet an absolute QoS requirement. In the second part, we propose a class provisioning methodology that can provide an absolute QoS level to each class. The dissertation is summarized and suggestions for future work are given in Chapter 6.

Chapter 2

Relative Differentiation

This chapter provides the background for the *Proportional Differentiated Services* architecture. We first state the *relative differentiation premise*, as the basic requirement between a number of graded classes of service. The important concepts of *dynamic class selection* and *class provisioning* are discussed. We then impose two crucial requirements on the relative differentiation premise. The *controllability* requirement aims to provide the network provider with the appropriate 'knobs' for adjusting the performance spacing between classes. The *predictability* requirement states that higher classes should provide better performance than lower classes, independent of the load conditions. Finally, we propose the *Proportional Differentiation* model as a means for controllable and predictable differentiation, in terms of the per-hop queueing delays and packet losses.

2.1 Relative differentiation premise

As discussed in §1.4, there are two distinct models in the DiffServ architecture: *absolute differentiation* and *relative differentiation*. The former uses some form of admission control in order to provide an *absolute* , or *quantitative*, performance level to each class or macroflow. The latter offers a number of classes of service that are *relatively differentiated*, such that higher classes receive better performance than lower classes, but at a higher cost. The actual performance of a class is *not* known a priori. Consequently, users and applications that have an absolute QoS requirement have to search dynamically for an acceptable class, depending on their QoS and cost requirements.

The relative differentiation model does not require admission control, bandwidth brokers and resource reservations, or any form of signalling between the users and the network. It also does not require route pinning or provisioning. So, it is considered as a simpler architecture to deploy and manage overall [33]. Since there is no admission control, however, the offered load in a link cannot be controlled or predicted. This is the fundamental reason why absolute QoS guarantees cannot be provided in this architecture.

The central premise of relative differentiation is that the N classes of service are *ordered* in the following sense:

> Class i provides better (or at least no worse) performance than class j, for $i > j$, in terms of the per-hop queueing delays and packet losses.

The elucidation 'or no worse' is required, since all classes may experience the same performance level in low load conditions. The IETF has standardized eight relative differentiation classes, called *Class Selector Compliant Per-Hop-Behaviors (CSC PHBs)*, or simply *Class Selectors* [74]. The three bits of the IPv4 header that identify the Class Selectors correspond to the Precedence bits in the original IPv4 packet header [84]. Even though the use of the Precedence bits has been limited in the past, there have been certain links or networks that deployed simple DiffServ schemes (such as providing higher priority to Telnet traffic) [67].

An important point about the Class Selectors is that higher classes offer better performance *in terms of both queueing delays and packet losses*. In other words, the Class Selectors provide 'one-dimensional' differentiation. This simplification is justified by the fact that the Precedence bits in the original IPv4 header were not discriminating between delays and losses. It is likely that a 'two-dimensional' differentiation model, in which a class causes higher delays but less losses than another class, would be more useful for some applications. An example of two-dimensional differentiation is given in §4.3.

The mapping of packets to a certain Class Selector can be done either at the application level, at the operating system of the end-hosts, or at the edge routers of a network. For example, a WWW server can classify the requested HTTP transactions based on the user-subscription level,

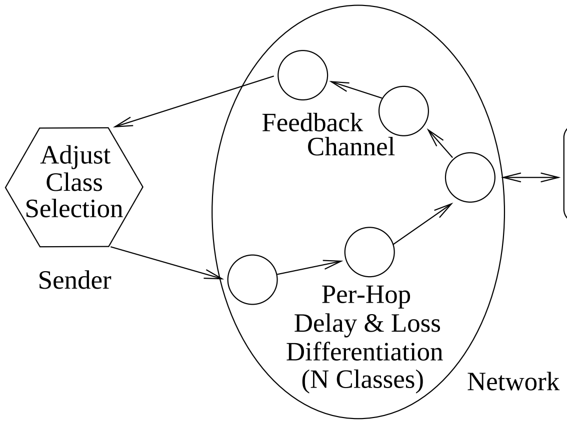
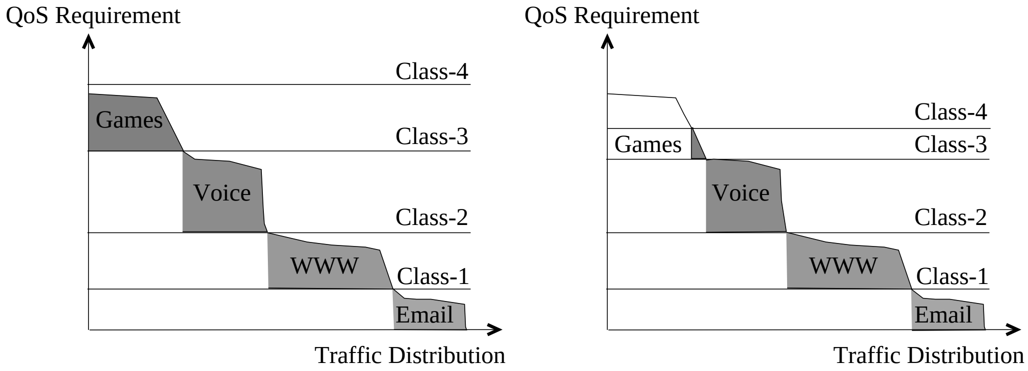


Figure 6: Dynamic class selection in a relative differentiation network.

so that 'non-members' are mapped to the lowest class, while 'members' get the better service of a higher class [8]. Or, in the case of an academic organization, a policy-based classification of packets may be performed at the local hosts, so that faculty use the highest service class, graduate students a middle service class, while the undergraduate students the lowest class. Or, a commercial network may classify the ingress traffic at the corresponding edge router based on the class prices and the maximum tariff that customers are willing to pay. The mapping of packets to different classes can be performed using flow classification techniques [6, 100].

Applications and users that do not have absolute QoS requirements can select their class based on the performance versus cost trade-off. In this case, a user would choose the highest possible class that she is willing to pay for. We believe that a large part of the Internet workload falls in this category. On the other hand, for applications and users that have an absolute QoS requirement, the relative differentiation architecture requires that they *dynamically search for the class that provides them with an acceptable QoS level.* This dynamic class selection model is illustrated in Figure 6.

The network, in this model, offers per-hop delay and loss differentiation, providing higher classes with a better performance. The application, on the other hand, aims for an operating range in which it performs well in terms of the end-to-end delay and loss rate. In this example, the sender maps the packet flow to a Class Selector, while the receiver monitors the end-to-end



(a) Well-provisioned differentiation

(b) Under-provisioned differentiation

Figure 7: Well-provisioned and under-provisioned differentiation in a link with four classes and four traffic types.

performance. The receiver notifies the sender about the current performance through a *feedback channel*. Based on this feedback, the sender decides whether to stay in the same class, or switch to a higher or lower class. If the user is also interested in minimizing the cost of the session, the application searches for the *least expensive (i.e., minimum)* class that offers acceptable delays and losses. The dynamic class selection process, outlined here, is studied in detail in Chapter 5.

It is possible that an application cannot find an acceptable class, even if the highest class is selected. This depends on the forwarding resources in the link (bandwidth and buffers), and on the volume and QoS requirements of the traffic mix. Intuitively, if the link is adequately *provisioned*, i.e., if it has enough forwarding resources for the given workload, there will be an acceptable class for each traffic type. A detailed study of a specific instance of this provisioning problem is given in §5.4. We next give a graphical example to illustrate the provisioning issue.

Figure 7 shows a QoS requirement curve for a link with four traffic types: Email, WWW, Voice, and Games (as in Figure 2). The link offers four classes of service. In the *well-provisioned* case (Figure 7-a), the performance level in each class is such that all traffic types can find an acceptable class. For instance, the 'Games' traffic type can meet its requirements in Class-4, while the WWW traffic type would use Class-2. In the *under-provisioned* case (Figure 7-b), on the other hand, the forwarding resources in the link are not adequate for Class-4 to meet its QoS requirement. The result is that a large part of the 'Games' traffic type does not find an acceptable performance, even in the highest class.

2.2 Controllability requirement

The relative differentiation premise of §2.1 simply states that a higher class should provide better performance than a lower class. This is a minimalistic requirement, similar to the Precedence field specification in the original IPv4 header [84]. We advocate that in order for a relative differentiation model, such as the Class Selectors, to be effective in practice, it has to meet some additional requirements on the relation between classes. The first of these requirements is presented in this section.

A requirement, from the perspective of the network provider, is that *the relative QoS spacing between classes should be controllable, based on appropriate performance 'knobs' that the router provides and the operator can adjust.* A separate set of performance knobs can exist for adjusting the queueing delay differentiation, and another for controlling the packet loss differentiation. For example, using these knobs the operator can make the highest class much better than the lower N-1 classes, and restrict its usage only to routing and network management traffic. Additionally, the operator can use the performance knobs to create a smoother differentiation between the lower N-1 classes, aiming to capture the wide range of QoS and cost requirements that users have.

Note that, ideally, a differentiation scheme should be able to adjust not only the relative spacing between classes, but also the *absolute* QoS level in each class. For instance, instead of controlling how much better a class is relative to another class, a network provider would prefer to specify a maximum delay and/or a maximum loss rate for each class. Since the aggregate load or the class load distribution is not known, however, due to the lack of admission control, such absolute performance levels cannot be provided in this architecture.

As an example of a relative differentiation model that is *not controllable*, consider the

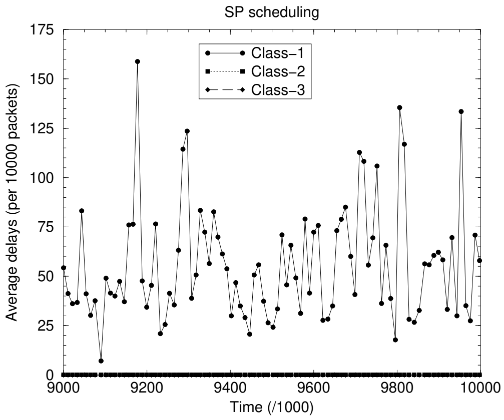


Figure 8: Short-term average delays (K=10000pks) with a three-class SP scheduler.

Strict Priority (SP) packet scheduler. SP services the packet from the highest backlogged class first. Even though SP satisfies the relative differentiation premise, providing higher classes with lower delays, it has some important drawbacks. First, the lower classes can experience long 'starvation' periods in which the queueing delays are excessive. Second, SP does not offer any differentiation knobs that the network provider can adjust in order to control the performance spacing between classes. Instead, the delays in each class depend exclusively on the load conditions and the load distribution between classes.

These drawbacks of SP are illustrated in Figure 8. The graph resulted from a simulation of a three-class lossless Strict Priority (SP) scheduler. Note that all simulation procedures and parameters in this dissertation are described in §7.1; the reader can refer to that appendix for all the simulation details not given here. The utilization of the scheduler is u=90%, and the class load distribution is $(\lambda_1, \lambda_2, \lambda_3)$=(60,30,10)[1]. The sample path shows the average delays for the three classes, measured in every K=10000 packet departures of the aggregate traffic, (i.e., it shows *short-term average delays in a timescale of K packets*). The queueing delays are measured in (fixed-size) packet transmission times, e.g., a queuing delay of 10 units means that the packet has to wait in the queue for 10 other packets to be transmitted first.

Returning to Figure 8, note that only Class-1 encounters significant queueing delays,

[1]This notation means that 60% of the traffic is Class-1 packets, 30% Class-2, and 10% Class-3.

while Class-2 and Class-3 have only minor delays and they are practically indistinguishable[2]. This implies that the network operator does not have a way to differentiate the two higher classes, users have no reason to select Class-3 instead of Class-2, and Class-1 receives the entire performance penalty.

As an example of a controllable scheme, consider the *capacity differentiation* model. The basic idea in this scheme is to *provide more forwarding resources to higher classes, relative to the class input loads.* The amount of forwarding resources allocated to each class controls the relative differentiation in this model. To illustrate the capacity differentiation model, we focus next on the allocation of bandwidth between classes and on the corresponding queueing delay differentiation.

A scheduling discipline that has been proposed in [48] for providing relative delay differentiation is the Generalized Processor Sharing (GPS) scheduler [29] and its many packet-based approximations, such as Weighted Fair Queueing (WFQ) (see [103] and references therein). GPS provides a minimum service rate $r_i(t)$ to each backlogged class i at time t, based on the class weights $\{w_j, j = 1 \ldots N\}$

$$r_i(t) = C \frac{w_i}{\sum_{j \in B(t)} w_j} \tag{2.1}$$

where C is the link bandwidth (capacity), and $B(t)$ is the set of backlogged classes at time t. The GPS weights w_i are the performance knobs that determine the delay differentiation between classes. If the GPS weights are selected so that a class has a larger share of the link capacity than the lower classes, relative to the class input loads, higher classes are expected to encounter lower queueing delays. The delay differentiation that results with the GPS scheduler is further discussed in §3.5.1.

[2] The curves for the Class-2 and Class-3 delays are also indistinguishable in the graph.

2.3 Predictability requirement

The second requirement that we impose on the relative differentiation model is that the differentiation between classes should be *predictable*. By 'predictable', we mean that *the performance of higher classes should be better than the performance of lower classes independent of the aggregate load, the class load distribution, or the timescales in which the performance is measured.*

Obviously, the value of relative differentiation would be limited if the class ordering is only met in a certain load range (e.g., when the utilization is more than 90%), or in a class load distribution (e.g., uniform load between classes). It is also important to provide predictable class ordering in all timescales. Saying that the average delay in class i is lower than the average delay in class j, implying the use of *long-term average delays*, does not provide any information on the relation between the delays of the two classes in short timescales. A predictable differentiation in short timescales is particularly important because most flows in the Internet are rather short-lived. For instance, an HTTP connection often consists of less than ten data packets [112].

The predictability requirement is important for two reasons. First, since users would pay a larger tariff for higher classes, they need to know that the performance in higher classes is better than the performance in lower classes. Factors such as the load conditions, the load distribution, or the timescales in which the performance is measured, are important for the network provider but not for the users. Such factors should not affect the 'class ordering contract' between the network and the users.

Second, as will be discussed in more detail in Chapter 5, a predictable class differentiation in all load conditions and timescales allows users and applications to dynamically select an acceptable class, when such a class exists. Intuitively, if the differentiation is not predictable, the class ordering may vary as the class load distribution varies. In that case, users and applications may end up moving between classes in an erratic manner, without converging to an acceptable class, even if such a class exists.

To illustrate the predictability requirement, and to also show that the WFQ (or GPS) scheduler of §2.2 is *not* predictable, Figure 9 shows a sample path of the queueing delays in

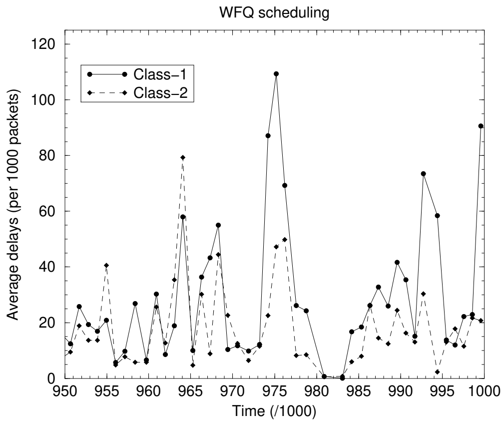


Figure 9: Short-term average delays (K=1000pks) with a two-class WFQ scheduler.

a two-class WFQ scheduler. The queueing delays shown are short-term averages measured in every K=1000 packet departures (i.e., the performance measuring timescale here is 1000 packet departures). The load distribution is uniform and the utilization is 90%. The WFQ weights are selected as (w_1, w_2)=(0.475,0.525), so that the *long-term average delay* is about 50 time units in Class-1, and about 20 time units in Class-2. Note that even though the class differentiation is predictable in terms of long-term averages, the differentiation between the two classes in the shorter timescale of Figure 9 is not predictable. At several time periods (with a duration of K packet departures), the delays in Class-2 are *higher* than the delays in Class-1, meaning that the performance ordering between classes has been inverted. The fact that GPS is not predictable can also be illustrated showing its dependence on the load distribution between classes. This is is shown in a more extensive discussion of GPS in §3.5.1.

As an example of a predictable differentiation model, consider the Strict Prioritization scheme discussed in §2.2. The SP scheduler services the highest backlogged class *always* first, independent of the load conditions, and so it provides predictable delay differentiation. SP is predictable, but as shown in the previous section it is not controllable. On the other hand, the GPS scheduler is controllable, but as shown in this section it is not predictable. The crucial question is whether there exists a relative differentiation model that is both controllable and predictable. Such a model is proposed in the next section.

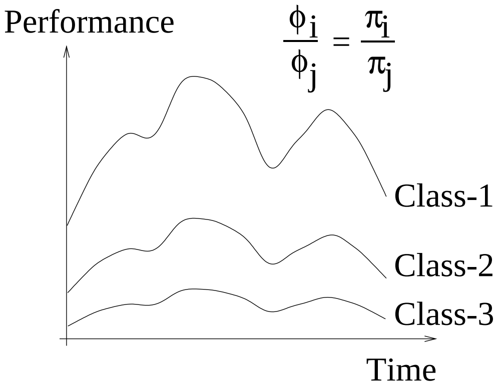


Figure 10: Proportional differentiation in terms of a general performance metric ϕ.

2.4 Proportional differentiation model

In this section, we propose the *Proportional Differentiation model* as a means for controllable and predictable differentiation between the N offered classes. The idea of proportional differentiation is quite simple: *the spacing between classes should follow proportional constraints on the class performance levels.* More formally, if ϕ_i is a metric for the performance of class i (e.g., average delay, loss rate), the proportional differentiation model requires that

$$\frac{\phi_i}{\phi_j} = \frac{\pi_i}{\pi_j} \qquad 1 \leq i, j \leq N \tag{2.2}$$

where π_i is the *differentiation parameter* for class i. Note that if lower values of ϕ_i lead to better performance, we must have that $\pi_i < \pi_j$ if $i > j$.

Figure 10 illustrates the basic idea of proportional differentiation in terms of a generic performance metric ϕ. If higher classes are assigned lower differentiation parameters, they will get better performance than the lower classes. The performance spacing between classes can be adjusted through the differentiation parameters π_i, which makes the proportional differentiation model *controllable* (§2.2). Notice that the proportional differentiation model is independent of the aggregate load or the class load distribution. Consequently, the network operator can provide a certain spacing between classes, even if the load conditions are not a priori known, or even when they dynamically vary.

The performance spacing between classes can be expressed in terms of the per-hop queueing delays and/or in terms of packet losses. Since these two effects are the major performance degradation factors in packet networks, *we apply the proportional differentiation model in this dissertation in terms of both queuing delays and packet losses.* Specifically, let $\bar{d}_i$ be the average queueing delay of the serviced packets in class i. The proportional differentiation model in the context of queueing delays requires that

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j} \qquad 1 \leq i, j \leq N \tag{2.3}$$

where δ_i is the *Delay Differentiation Parameter (DDP)* for class i. The DDPs are ordered as $\delta_1 > \delta_2 > \ldots > \delta_N > 0$. The Proportional Delay Differentiation (PDD) model of (2.3) is studied in detail in §3.1.

Similarly, let $\bar{l}_i$ be the packet loss rate in class i, i.e., the long-term fraction of packet arrivals in class i that are dropped. The proportional differentiation model in the context of packet losses (or drops) requires that

$$\frac{\bar{l}_i}{\bar{l}_j} = \frac{\sigma_i}{\sigma_j} \qquad 1 \leq i, j \leq N \tag{2.4}$$

where σ_i is the *Loss Differentiation Parameter (LDP)* for class i. The LDPs are ordered as $\sigma_1 > \sigma_2 > \ldots > \sigma_N > 0$. The Proportional Loss Differentiation (PLD) model of (2.4) is studied in detail in §4.1. For example, the network provider can specify that the average delay and the packet loss rate in each class is twice as large, compared to the immediately higher class, setting $\delta_i = 2\delta_{i+1}$ and $\sigma_i = 2\sigma_{i+1}$ for $i = 1 \ldots N-1$.

A crucial point about the proportional differentiation model is that it specifies a QoS spacing between classes that holds *independent of the aggregate load or the class load distribution.* This is a radically different approach than other differentiation schemes that *expect* certain load conditions, and provision an absolute QoS level in each class that can only be met in those load conditions. In those differentiation schemes, when the load prediction fails, the resulting differentiation becomes unpredictable in the sense that a class can get worse performance than

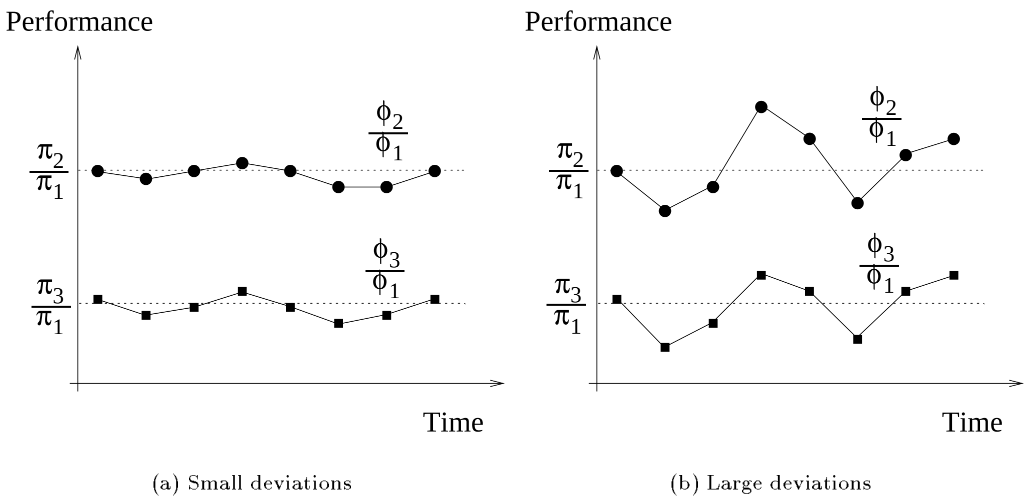


Figure 11: Proportional differentiation in short timescales.

some lower classes. An example of this behavior is shown in §3.5.1 for the GPS scheduler. The proportional differentiation model does not have this problem, because it specifies an *invariant* performance spacing between classes that is independent of the load conditions.

A large part of this thesis focuses on router mechanisms that can implement the proportional differentiation model in terms of queueing delays (Chapter 3) and packet losses (Chapter 4). We show there that there are fundamental issues that prevent the proportional differentiation model from being always *feasible*. Given that it may be impossible to always achieve the constraints of (2.3) or (2.4), we have to accept some *approximation errors* or *deviations* from the ideal proportional constraints that the differentiation parameters specify. Consequently, when we compare different mechanisms for packet scheduling or dropping, the magnitude of these errors is used to decide which mechanism is better.

To illustrate this issue, Figure 11 shows two cases of proportional differentiation between three classes, resulting from two different mechanisms. In both cases, the proportional constraints are met when we measure the class performance ratios over the entire time axis (i.e., in terms of the *long-term average behavior*). In the first case (Figure 11-a), that results say from a mechanism X, the differentiation between classes is not always as specified by the ratios π_i/π_j, but it is quite close to those constraints. In the second case (Figure 11-b), that results from a mechanism Y, the errors in meeting the proportional differentiation constraints are significantly larger. In terms of choosing a candidate mechanism, we would choose X instead of Y because the former causes smaller deviations from the proportional differentiation model.

Note that we choose to not quantify these errors using second-order statistics (such as the variance or standard deviation of the resulting performance ratios). Instead, we evaluate the errors in the time-domain; this is done by measuring the performance ratios between classes *in short timescales*, say in every K packet departures or arrivals, and then computing the resulting class ratio distributions. This time-domain technique will be illustrated in Chapter 3 for short-term average queueing delays, and in Chapter 4 for short-term packet loss rates.

The evaluation of proportional differentiation in the time domain provides us with two benefits. First, as discussed in the previous paragraph, it quantifies the deviations from the ideal proportional constraints that the differentiation parameters specify. Second, it allows us to *investigate the predictability* of the resulting relative differentiation in short timescales. The measurement of the performance ratios between classes in a range of timescales, from a few tens or hundreds of packets to thousands or millions of packets, shows whether the higher classes encounter better performance independent of the performance monitoring timescales.

2.5 Summary

The main strength of the relative differentiation model is its simplicity. It does not require admission control, signalling, provisioning, or bandwidth brokers. Also, it does not require the knowledge or control of the offered load in each network link, and so it is insensitive to dynamic routing changes. The fundamental drawback, though, is that it only provides a relative ordering between classes, and not a per-class absolute QoS level. We stated two requirements that a relative differentiation model should meet: controllability and predictability. The former requires that the network operator should be able to adjust the class spacing between classes. The latter requires that the relative ordering between classes should be met independent of load conditions and timescales. Existing differentiation models, such as strict prioritization or capacity differentiation, are either not controllable or not predictable. We proposed an original service model, called proportional differentiation, that is both controllable and predictable. The model is stated in terms of both queueing delay and loss rate differentiation. The next two chapters investigate these two models, and design router mechanisms that can implement them.

Chapter 3

Proportional Delay Differentiation

The subject of this chapter is the queueing delay differentiation between traffic classes, and the related packet scheduling problem. We first propose the Proportional Delay Differentiation (PDD) model as a means for controllable and predictable delay differentiation. Starting from the PDD model, we derive the average queueing delay in each class, show the dynamics of the class delays under the PDD constraints, and state the conditions under which the PDD model is feasible. In the second part of the chapter, we focus on packet scheduling algorithms that can implement the PDD model, when the model is feasible. The Proportional Average Delay (PAD) scheduler meets the PDD constraints, when they are feasible, but it exhibits unpredictable behavior in short timescales. The Waiting Time Priority (WTP) scheduler, on the other hand, approximates closely the PDD model even in the shortest timescales of a few packet departures, but only in heavy load conditions. PAD and WTP serve as motivation for the third scheduling algorithm, called Hybrid Proportional Delay (HPD). HPD combines the features of PAD and WTP. The chapter closes with a review of other schedulers in the context of relative or proportional delay differentiation.

3.1 Proportional Delay Differentiation (PDD) model

Consider a *Packet Forwarding Engine (PFE)* in a router. The PFE consists of a *router output interface* (or output), which transmits (or services) packets with a constant capacity (or service rate). Packets arrive in the PFE from other PFEs or *router input interfaces*, which generally have different capacities. Packets that cannot be transmitted immediately upon their arrival, because the output services other packets, are queued in the PFE. A separate First-Come First-Serve (FCFS) *packet queue* is maintained for each of the N classes of service[1]. The order in which packets are selected for transmission, from the head of each backlogged queue, is determined by the *packet scheduler*. It is the scheduler that provides the queueing delay differentiation between traffic classes, which is the subject of this chapter. In Chapter 4, we will extend the PFE model with two other modules, the *backlog controller* and the *packet dropper*, used when packets are occasionally dropped from the PFE queues. For now, we assume that the PFE queues are *lossless*.

In more detail, the PFE model is as follows. A *lossless*, *work-conserving*, and *non-preemptive* PFE with transmission capacity C (bytes per second) services N queues, one for each traffic class. Let λ_i be the *average input rate*, or simply the average rate, in class i (packets per second), and $\lambda{=}\sum_{i=1}^{N} \lambda_i$ the average input rate in the PFE. Let $\bar{L}_i$ be the *average size* (bytes) of class i packets, and $\bar{L}{=}\sum_{i=1}^{N}(\lambda_i \bar{L}_i)/\lambda$ the average size among all packets. The *utilization* of the PFE is $u{=}\bar{L}\lambda/C$. The lossless property requires that $u < 1$ (i.e., the PFE is stable), and that there are enough buffers for packets that need to be queued. The work-conserving property means that the scheduler does not idle when there are waiting packets; this is a common practice in routers in order to minimize the queueing delays. The non-preemptive property means that a packet transmission is always carried out to completion, which is also the standard practice in packet networks.

The *Proportional Delay Differentiation (PDD)* model aims to control the *ratios* of the average queueing delays between classes based on the *Delay Differentiation Parameters (DDPs)* $\{\delta_i, i = 1, \ldots, N\}$. Specifically, let $\bar{d}_i$ be the *average queueing delay*, or simply average delay, of the class i packets. The PDD model requires that the ratio of average delays between two classes i and j is fixed to the ratio of the corresponding DDPs

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j} \qquad 1 \leq i, j \leq N \tag{3.1}$$

[1] We use the terms 'class' and 'queue' interchangeably.

Following the convention that higher classes provide better service, $\delta_1 > \delta_2 > \ldots > \delta_N > 0$. In the following, we choose Class-1 as the 'reference class' and set δ_1=1. Then, the PDD model requires that the average delay of each class i is a certain fraction δ_i of the average delay of Class-1,

$$\bar{d}_i = \delta_i \, \bar{d}_1 \qquad i = 2, \ldots, N \tag{3.2}$$

independent of the aggregate load, or the class load distribution.

3.1.1 Per-class average delays in the PDD model

In given load conditions, the N-1 ratios of the PDD model specify uniquely the average delays of the N classes. The key additional relation in the mapping from delay ratios to class delays is the *conservation law* [54, 13], which constrains the average class delays in any work-conserving scheduler $\mathcal{S}$. The conservation law holds under arbitrary distributions for the packet interarrivals and packet sizes, as long as the first moment of these distributions (λ_i and $\bar{L}_i$) and the second moment of the packet size distribution exist, and the packet scheduling discipline $\mathcal{S}$ is independent of the packet sizes.

Specifically, the conservation law requires that for any work-conserving scheduler $\mathcal{S}$ of capacity C that causes an average delay $\bar{d}_i$ in each class i, we must have that

$$\sum_{i=1}^{N} \lambda_i \, \bar{L}_i \, \bar{d}_i = \lambda \, \bar{L} \, \bar{d}_{ag} = \bar{q}_{ag} \tag{3.3}$$

where $\bar{d}_{ag}$ and $\bar{q}_{ag}$ is the average delay (in seconds) and the average backlog (in bytes), respectively, in a First-Come First-Served (FCFS) scheduler of capacity C that services the same aggregate traffic stream as $\mathcal{S}$. The conservation law implies that even though a scheduler $\mathcal{S}$ can affect the relative magnitude of the class delays, making the delay of one class lower than the delay of another class, this is a 'zero-sum' game because the weighted sum of (3.3) has to be equal to the average backlog $\bar{q}_{ag}$ of the aggregate traffic stream. The average backlog $\bar{q}_{ag}$ is a significant *invariant* in this balance, as it does not depend on the scheduling discipline $\mathcal{S}$, but only on the aggregate traffic stream and the service rate C.

Suppose now that the scheduler $\mathcal{S}$ satisfies the PDD model of (3.1). With the additional constraint (3.3) that the conservation law imposes, we can show that the average delay in class i is

$$\bar{d}_i = \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda_n \bar{L}_n} \qquad i = 1, \ldots, N \tag{3.4}$$

Consequently, even though the PDD model consists of N-1 *relative* constraints, the average delay in each class is absolutely specified when the PDD model is applied to a certain traffic stream with load distribution $\{\lambda_i\}$, average packet sizes $\{\bar{L}_i\}$, and aggregate backlog $\bar{q}_{ag}$.

When all classes have the same average packet size, i.e., $\bar{L}_i = \bar{L}$ for all i, we can set $\bar{L} = 1$, and normalize the backlog measures to *average packet size* units. In that case, the average delay in each class can be simplified to

$$\bar{d}_i = \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda_n} \qquad i = 1, \ldots, N \tag{3.5}$$

We assume that *all classes have the same packet size distribution*, and so[2], *the same average packet size* $\bar{L} = 1$.

3.1.2 Delay dynamics in the PDD model

Based on Equation (3.5), we can now investigate the variations in the average class delays, under the constraints of the PDD model as the aggregate load, the class load distribution, or the DDPs vary. We refer to the following properties as the *delay dynamics in the PDD model.* The proofs of these properties are given in §7.2.

Property 1: Increasing the input rate of a class, increases (in the wide sense)[3] the average delay of all classes.

[2] The reason for the stronger assumption on the packet size distribution will become clear in §3.1.3.
[3] '*Increasing in the wide sense*' means that the corresponding function is non-decreasing.

This property shows that in the PDD model there is no segregation between classes, that is expected due to the relative differentiation nature of the model. When the delay of a class increases, due to additional load in that class, the delays of all classes will also encounter an increase.

Property 2: Increasing the rate of a higher class causes a larger increase in the average class delays than increasing the rate of a lower class.

In the case of two classes, for instance, suppose that the class rates become either $\lambda_1' = \lambda_1 + \epsilon$ and $\lambda_2' = \lambda_2$, or $\lambda_1'' = \lambda_1$ and $\lambda_2'' = \lambda_2 + \epsilon$ $(\epsilon > 0)$. Even though the conservation law requires that the weighted average of the class delays is the same in both cases $(\lambda_1'\bar{d}_1' + \lambda_2'\bar{d}_2' = \lambda_1''\bar{d}_1'' + \lambda_2''\bar{d}_2'')$, the class average delays in the second case are larger, i.e., $\bar{d}_1'' > \bar{d}_1'$ and $\bar{d}_2'' > \bar{d}_2'$, because of the PDD constraints. This property shows that higher classes cost more, in terms of queueing delay, than lower classes.

Property 3: Decreasing the delay differentiation parameter of a class increases (in the wide sense) the average delay of all other classes, and decreases (in the wide sense) the average delay of that class.

This property implies that if the delay of a class is reduced, by lowering its DDP, then the delay of all other classes will increase.

The following two properties are important in the Dynamic Class Selection (DCS) model (see §5.1 and §5.2). In the DCS model, users dynamically choose the class they use, searching for the minimum class that provides them with an acceptable delay. The first property states that when one ore more users move to a higher class, the delay of all classes increases. The class delays decrease, on the other hand, when one or more users move to a lower class. The second property shows that when a user switches from one class to another, the user still observes a consistent class ordering, i.e., the higher class provides a lower delay.

Suppose that the class load distribution changes from $\{\lambda_n\}$ to $\{\lambda_n'\}$, with $\lambda_i' = \lambda_i - \epsilon$, $\lambda_j' = \lambda_j + \epsilon$, and $\lambda_k' = \lambda_k$ for all $k \neq i, j$ $(\epsilon > 0)$. Let $\bar{d}_n'$ be the average delay in class n when the class load distribution is $\{\lambda'_n\}$.

Property 4: If $i > j$ then $\bar{d}'_n \leq \bar{d}_n$ for all $n = 1 \ldots N$. Similarly, if $i < j$ then $\bar{d}'_n \geq \bar{d}_n$.

Property 5: If $i > j$ then $\bar{d}'_j \geq \bar{d}_i$. Similarly, if $i < j$ then $\bar{d}'_j \leq \bar{d}_i$.

3.1.3 Feasibility of the PDD model

We have assumed so far that the PDD model is *feasible*, i.e., that there exists a work-conserving scheduler that can meet the constraints of (3.2). However, this is not always possible. Given the load distribution and the average backlog of the aggregate traffic, the PDD model not only specifies the N-1 delay ratios, but also the N average delays of Equation (3.5). The important point, though, is that there may not exist a work-conserving scheduler that can set the average delay of each class to a certain value. Intuitively, the reason is that the average delay of a class (or of any set of classes) has a lower bound due to the inherent load in that class (or set of classes). This lower delay bound would result if that class (or set of classes) was given strict priority over the rest of the traffic.

Formally, given the input rates $\{\lambda_i\}$ and the average backlog $\bar{q}_{ag}$ of the aggregate traffic stream, we say that *a set of DDPs $\{\delta_i,\ i = 2, \ldots, N\}$ is feasible when there exists a work-conserving scheduler that can set the average delay of each class as in (3.5).* So, the set of DDPs $\{\delta_i,\ i = 2, \ldots, N\}$ is feasible when the set of class delays $\{\bar{d}_i = \delta_i \bar{q}_{ag}/(\sum_{n=1}^{N} \lambda_n \delta_n),\ i = 1, \ldots, N\}$ is feasible.

The necessary and sufficient conditions for the feasibility of a set of N average class delays, given the N class loads, were derived by Coffman and Mitrani in [26] (see also the follow-up work [69]). Under general assumptions[4], a set of N average delays $\{\bar{d}_i,\ i = 1, \ldots, N\}$ is

[4] Most of [26] assumes Poisson arrivals. The particular result that we include here holds, as [26] also states, for general arrival and packet size distributions.

feasible if and only if the following 2^N-2 inequalities hold,

$$\sum_{i \in \phi} \lambda_i \, \bar{d}_i \; \geq \; \bar{d}_{\phi}^{SP} \sum_{i \in \phi} \lambda_i = \bar{q}_{\phi}^{SP} \qquad \text{for all} \, \phi \in \Phi \tag{3.6}$$

where Φ is the set of 2^N-2 nonempty proper subsets of the set $\{1, 2, \ldots, N\}$. $\bar{q}_{\phi}^{SP}$ and $\bar{d}_{\phi}^{SP}$ are the average backlog and the average delay, respectively, of the traffic in the set ϕ, *if ϕ was given strict priority over all other classes.* The Coffman-Mitrani inequalities state that any set of classes ϕ has a lower bound on its average backlog and this bound results when the traffic in ϕ is serviced with the highest priority in a Strict Priority (SP) scheduler.

If all classes have the same packet size distribution, which is also our assumption[5], Regnier [88] extended the Coffman-Mitrani results, showing that the necessary and sufficient feasibility conditions can be reduced from 2^N-2 to the following N-1 inequalities

$$\sum_{i=k}^{N} \lambda_i \, \bar{d}_i \; \geq \; \bar{d}_{k,N}^{SP} \sum_{i=k}^{N} \lambda_i = \bar{q}_{k,N}^{SP} \qquad k = 2, \ldots, N \tag{3.7}$$

where $\bar{q}_{k,N}^{SP}$ and $\bar{d}_{k,N}^{SP}$ are the average backlog $\bar{q}_{\phi}^{SP}$ and average delay $\bar{d}_{\phi}^{SP}$, respectively, for $\phi = \{k, \ldots, N\}$. When $\phi = \{k\}$, we simply write $\bar{q}_{k}^{SP}$ and $\bar{d}_{k}^{SP}$. Regnier's result imposes a lower bound on the average backlog of only the N-1 subsets of the k highest classes ($k = 1, \ldots, N-1$).

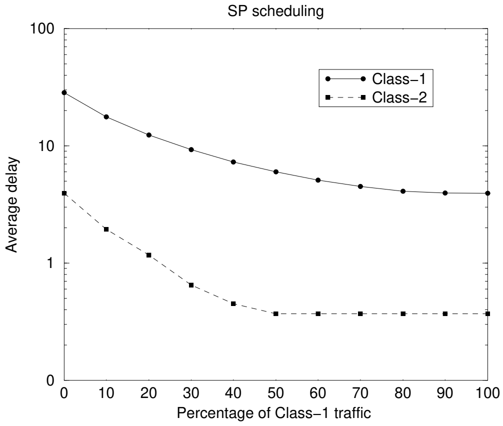
From (3.7) and (3.5), we can now show that the necessary and sufficient condition for the feasibility of the PDD model when $N = 2$ is

$$\bar{d}_2 = \frac{\delta_2 \bar{q}_{ag}}{\lambda_1 + \delta_2 \lambda_2} \geq \bar{d}_2^{SP} \tag{3.8}$$

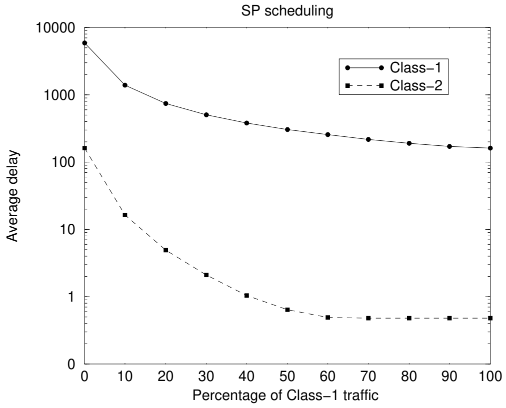
with $\delta_2 < \delta_1 = 1$. Since $\bar{q}_{ag} = \lambda_1 \bar{d}_1^{SP} + \lambda_2 \bar{d}_2^{SP}$, it is easy to show that the feasibility condition becomes

$$\delta_2 \geq \frac{\bar{d}_2^{SP}}{\bar{d}_1^{SP}} \quad \text{or} \quad \frac{1}{\delta_2} \leq \frac{\bar{d}_1^{SP}}{\bar{d}_2^{SP}} \tag{3.9}$$

[5] Note that this is stronger than assuming that all classes have the same average packet size, required for (3.5).



(a) Average delays, u=75%



(b) Average delays, u=95%

Figure 12: Average delays with SP as a function of the load distribution.

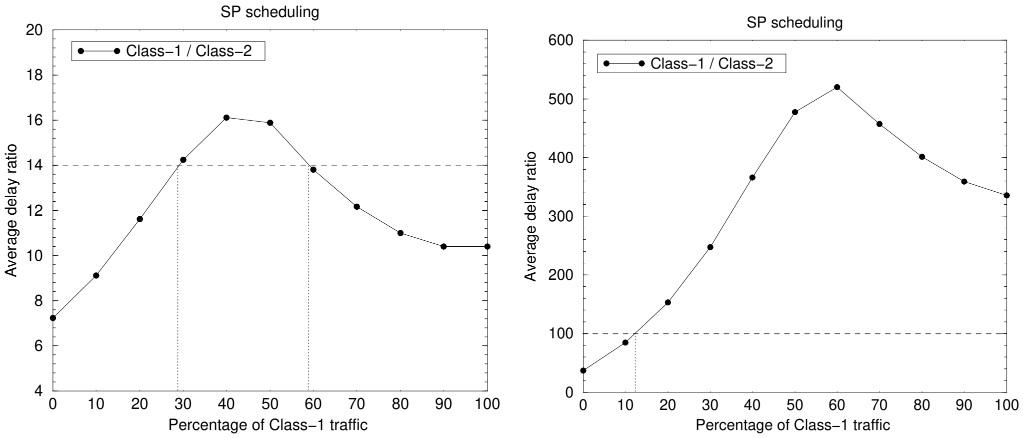
So, *a given delay ratio $\delta_2 < 1$ between Class-2 and Class-1 is feasible if and only if the corresponding average delay ratio in the SP scheduler is not higher than δ_2.* Note that the average delays of the two classes in SP, and so their ratio, are a function of the input rate λ, the load distribution (λ_1, λ_2), and the aggregate average backlog $\bar{q}_{ag}$.

When $N > 2$, the feasibility conditions for the PDD model are the following $N-1$ inequalities

$$\sum_{i=k}^{N} \lambda_i \delta_i \geq \frac{S}{\bar{q}_{ag}} \sum_{i=k}^{N} \lambda_i \bar{d}_i^{SP} \qquad k = 2, \ldots, N \tag{3.10}$$

where $S = \sum_{i=1}^{N} \lambda_i \delta_i$ and $\bar{q}_{ag} = \sum_{i=1}^{N} \lambda_i \bar{d}_i^{SP}$.

To illustrate the feasibility conditions graphically, Figure 12 shows for the case of two classes the per-class average delays with SP in two utilization points (u=75% and u=95%) over a range of load distributions. Also, Figure 13 shows the average delay ratios $\bar{d}_1^{SP}/\bar{d}_2^{SP}$ for the same load conditions. These graphs resulted from a simulation of the SP scheduler. When u=75% (Figure 13-a) and the load distribution is (λ_1, λ_2)=(70,30), the DDP $1/\delta_2$=8 is feasible, while the DDP $1/\delta_2$=16 is infeasible. On the other hand, if $1/\delta_2$=14, the load distribution must be such that $0.3 < \lambda_1/\lambda < 0.6$ in order for the PDD model to be feasible. When u=95% (Figure 13-b),



(a) Ratios of average delays, u=75%

(b) Ratios of average delays, u=95%

Figure 13: Average delay ratios with SP as a function of the load distribution.

the DDP $1/\delta_2$=32 is always feasible, while the DDP $1/\delta_2$=100 is only feasible when at least 12% of the traffic belongs to Class-1.

There are two important points shown in Figures 12 and 13. First, for a certain load distribution, the higher the utilization is, the larger the ratio $\bar{d}_1^{SP}/\bar{d}_2^{SP}$ becomes. So, *as the utilization becomes higher, the feasible range for the DDP δ_2 increases.* Second, *for a certain utilization, the feasible range for δ_2 decreases as the fraction of Class-1 or Class-2 traffic tends to zero.* The reason for this effect is as follows. When $\lambda_1 \to 100\%$ (and $\lambda_2 \to 0\%$), the average delay in Class-1 decreases to the average aggregate delay $\bar{d}_{ag}$. The average delay in Class-2, on the other hand, does not decrease to zero, because it is lower bounded from the (non-preemptive) service time of Class-1 packets. The end-result is that the ratio $\bar{d}_1^{SP}/\bar{d}_2^{SP}$ decreases, as $\lambda_1 \to$ 100%. Similarly, when $\lambda_1 \to 0$, the average delay in Class-2 increases to $\bar{d}_{ag}$. The average delay in Class-1 does not increase to infinity, because it is upper bounded from the (finite) duration of the Class-2 busy periods. The end-result is that the ratio $\bar{d}_1^{SP}/\bar{d}_2^{SP}$ decreases, as $\lambda_1 \to 0$.

If the number of classes is $N > 2$, such a graphical interpretation of the feasibility conditions is not straightforward. One can still use (3.10) to examine numerically if the N-1 conditions are satisfied for the given DDPs, load distribution, and aggregate backlog. The major issue, however, in examining the feasibility of the PDD model in a practical setting is to know the average delays $\bar{d}_k^{SP}$ that would result if a class was serviced with the highest priority by an SP scheduler. If the traffic does not conform well to a mathematically tractable queueing model, for which the average class delays are known analytically (see [13] for instance), one needs to estimate or measure them experimentally. Most existing routers provide an SP scheduler. So, a network provider can measure the class average delays $\bar{d}_k^{SP}$ deploying the SP scheduler for short time intervals in the actual workload. Alternatively, if the SP deployment is considered harmful due to its starvation effects (§2.2), the delays $\bar{d}_k^{SP}$ can be estimated from emulating an SP scheduler with the actual workload of the link.

3.2 Proportional Average Delay (PAD) scheduling

Another way to interpret the PDD model is that the *normalized average delays*, defined as $\tilde{d}_i = \bar{d}_i/\delta_i$, must be equal in all classes, i.e.,

$$\tilde{d}_i = \frac{\bar{d}_i}{\delta_i} = \frac{\bar{d}_j}{\delta_j} = \tilde{d}_j \qquad 1 \leq i, j \leq N \tag{3.11}$$

A scheduler that aims to equalize the normalized average delays among all classes is described next. We refer to this algorithm as *Proportional Average Delay (PAD)* scheduling.

Let $B(t)$ be the set of backlogged classes at time t, $D_i(t)$ be the sequence of Class-i packets that departed before time t (in order of their departure), and d_i^m be the delay of the m'th packet in $D_i(t)$. Assuming that there was at least one departure from class i before t, the normalized average delay of class i at t is

$$\tilde{d}_i(t) = \frac{1}{\delta_i} \frac{\sum_{m=1}^{|D_i(t)|} d_i^m}{|D_i(t)|} = \frac{1}{\delta_i} \frac{S_i}{P_i} \tag{3.12}$$

where S_i is the sum of queueing delays of all packets in $D_i(t)$, and P_i is the number of packets in $D_i(t)$ (or cardinality of $D_i(t)$).

Suppose that a packet has to be selected for transmission at time t. *PAD chooses the backlogged class j with the maximum normalized average delay at t,*

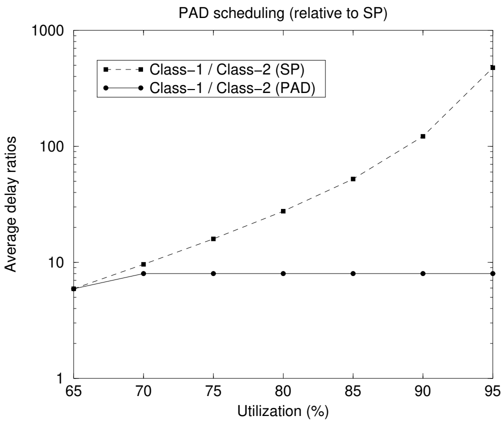
$$j = \arg \max_{i \in B(t)} \tilde{d}_i(t) \tag{3.13}$$

The packet at the head of queue j is transmitted, after a queueing delay $d_j^{P_j+1}$. The variables S_j and P_j are then updated as $S_j = S_j + d_j^{P_j+1}$ and $P_j = P_j + 1$, and the new normalized average delay $\tilde{d}_j(t)$ is computed from (3.12).

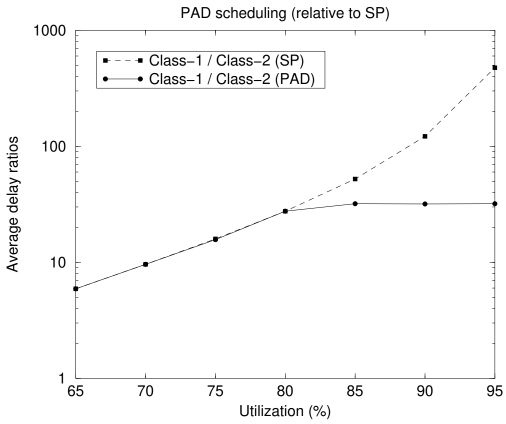
The selection of the maximum $\tilde{d}_i(t)$ term, as in (3.13), requires at most N-1 comparisons, which is a minor overhead for the small number of classes that we consider here. The main computational overhead of PAD is a division, for the calculation of S_j/P_j, after each packet departure[6]. With current commodity microprocessors ($\approx$ 500MHz), a floating-point division takes less than 100ns ($\approx$ 50 cycles). This delay would not be an issue for network interfaces of up to 1Gbps, which transmit a (minimum-size) 40-byte packet in about 320ns. For even faster network interfaces, one can avoid the division operation and compute $\tilde{d}_i(t)$ using an exponential running-average of the form $\tilde{d}_i(t) = (1 - 2^{-w})\tilde{d}_i(t) + 2^{-w} d_i^{P_i+1}/\delta_i$, where w is a positive integer. Also, during 'cold-start', or when the counters S_i and P_i are reset to zero due to an overflow, some classes will not have a history of departed packets. The scheduler can then start servicing packets in a FCFS order, until $P_i \neq 0$ for all i.

The basic idea in PAD is that if a packet is serviced from class j with the maximum normalized average delay, the delay of that packet will not increase any more (since it is serviced), and thus the increase of S_j due to that packet will be minimized. So, servicing a packet from class j tends to reduce the difference of $\tilde{d}_j(t)$ from the normalized average delays of the other classes. In the long-run, if the scheduler always minimizes the differences between the normalized average delays in this manner, we expect that the normalized average delays will be about the same. As will be illustrated next, this is the case with PAD when the selected DDPs are feasible.

[6] Obviously, the factor $1/\delta_i$ can be precomputed and does not require a division. If it is a power of two, it does not require a multiplication either.



(a) $1/\delta_2 = 8$

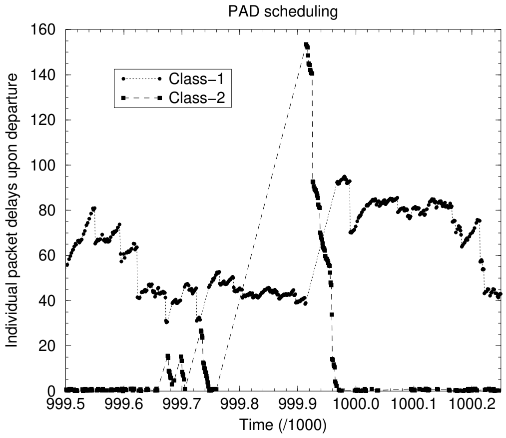


(b) $1/\delta_2 = 32$

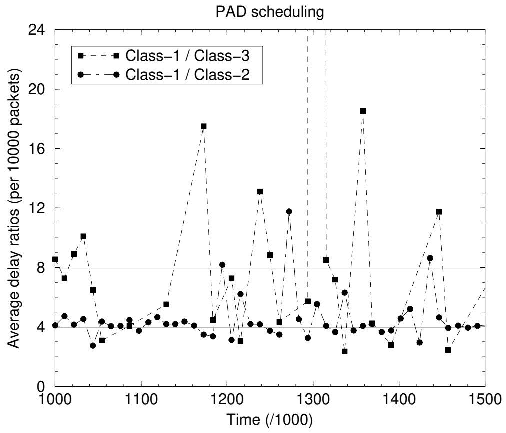
Figure 14: Average delay ratios with PAD and SP as a function of the utilization.

PAD is an excellent scheduler in terms of meeting the PDD model, when the chosen DDPs are feasible. We were unable to prove that PAD is *optimal*, in the sense that it can always meet a set of feasible DDPs. Simulation results, though, have provided us with evidence that PAD is quite close to such an optimal scheduler.

To illustrate this point, Figure 14 shows typical simulation results for N=2 classes. The two graphs show the ratio $\bar{d}_1/\bar{d}_2$ of the average delays with PAD and SP for two DDP selections and for a uniform load distribution (λ_1=λ_2), as a function of the utilization u. Recall from Equation (3.9) that in the case of two classes, a given DDP $1/\delta_2$ is feasible if and only if it is lower than the ratio $\bar{d}_1/\bar{d}_2$ in SP. As shown in Figure 14-a, PAD achieves the specified delay ratio $1/\delta_2$=8 in all utilization points that this DDP is feasible. The resulting delay ratio is less than eight only when u <70%, but SP does not lead to a larger delay ratio either, meaning that this DDP is infeasible in that utilization range. Similarly, in Figure 14-b, PAD achieves the specified delay ratio $1/\delta_2$=32 when the utilization is higher than about 80%, which is also when this DDP is feasible. *Because PAD can achieve the PDD model when the given DDPs are feasible, at least based on our simulation results, we consider PAD as an optimal scheduler for the PDD model.*



(a) Individual delays ($1/\delta_2$=5)



(b) Short-term (K=10000pks) delay ratios ($1/\delta_2$=4, $1/\delta_3$=8)

Figure 15: Individual packet delays and short-term delay ratios with PAD.

Is PAD also a predictable scheduler however? In order to answer this question we have to examine the behavior of PAD in short timescales, and check whether higher classes always encounter lower delays. To get a deeper insight on the behavior of PAD in short timescales, Figure 15-a shows the queueing delays of individual packets at their departure instant. The utilization in this simulation is u=90% and the load distribution to $(\lambda_1, \lambda_2) = (70, 30)$. Note that during most of the time, the Class-2 packets depart with minor queueing delays. Occasionally though, as it occurs shortly after time 999.900, some Class-2 packets depart with significantly larger queueing delays, even larger than the Class-1 delays in the corresponding time frame.

Let us analyze in detail what happens in the time period 999.760-999.960. Just before 999.760, the Class-2 queue is almost empty and its queueing delays are close to zero. From 999.760 to 999.920 (phase-1), $\tilde{d}_2$ is less than $\tilde{d}_1$, and there are no departures from Class-2. The arriving Class-2 packets during phase-1 accumulate large waiting times, up to the duration of this phase (160 time units), but the normalized average delay $\tilde{d}_2$ does not change since there are no departures from that class. At about 999.920, $\tilde{d}_1$ becomes less than $\tilde{d}_2$, after servicing exclusively packets of that class during phase-1. From that point and until 999.960 (phase-2), all the backlogged Class-2 packets from phase-1 depart back-to-back without interventions from Class-1 packets. At about 999.960, the two normalized average delays become comparable, and the scheduler starts servicing packets again in a more uniform manner. This example shows that *because PAD is unaware of the waiting times of backlogged packets, it occasionally allows higher classes to experience much larger queueing delays than their long-term average delays, or the queueing delays of lower classes.* This is a manifestation of unpredictable delay differentiation that should be avoided.

It is also instructive to examine the ratios of short-term average delays between classes with PAD, in relatively short timescales. Figure 15-b shows the ratios of average delays, measured in every K=10000 successive packet departures, for N=3 classes. The utilization is u=90%, and the load distribution is $(\lambda_1, \lambda_2, \lambda_3)$=(50,30,20). Note that PAD does a rather poor job in meeting the PDD constraints in this short timescale. This should be expected, since PAD attempts to equalize the *long-term* normalized average delays and not the normalized average delays in the *last K departures*. Another important point in this graph is that there are several time periods in which the ratio $\bar{d}_1/\bar{d}_2$ is larger than the ratio $\bar{d}_1/\bar{d}_3$, meaning that Class-3 experiences higher delays than Class-2. These are incidents, again, of unpredictable delay differentiation. *A good scheduler for the PDD model should not only achieve or closely approximate the constraints of (3.2), but it should also provide predictable delay differentiation in short timescales.* A scheduler that emphasizes on this predictability issue is studied next.

3.3 Waiting Time Priority (WTP) scheduling

The Waiting Time Priority scheduling algorithm was first studied by L.Kleinrock in 1964 [54] under the name *Time-Dependent Priorities*. A packet is assigned a priority that increases proportionally to the packet's waiting time. Higher classes have larger priority-increase factors. The packet with the highest priority is serviced first (in non-preemptive order). In the following, we describe WTP in a different way to highlight its similarities and differences with PAD.

Suppose that class i is backlogged at time t, and that $w_i(t)$ is the *head waiting time* of class i at t, i.e., the waiting time of the packet at the head of class i at t. We define the *normalized head waiting time* of class i at t as

$$\tilde{w}_i(t) = w_i(t)/\delta_i \tag{3.14}$$

Every time a packet is to be transmitted, the WTP scheduler selects the backlogged class j with the *maximum normalized head waiting time*,

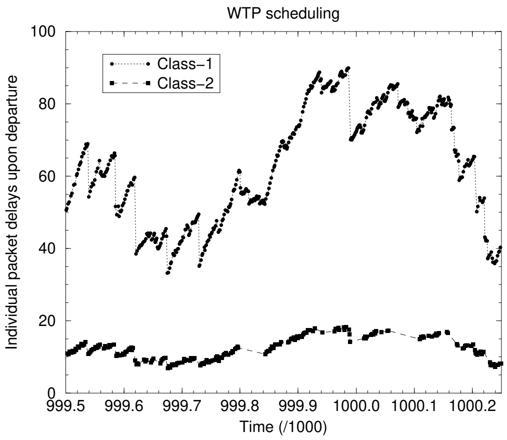
$$j = \arg\max_{i \in B(t)} \tilde{w}_i(t) \tag{3.15}$$

The similarities with PAD are now obvious. In the same way that PAD chooses for service the class with the maximum normalized average delay, WTP chooses for service the class with the maximum normalized head waiting time. PAD attempts to minimize in this manner the differences of the class normalized average delays, while *WTP attempts to minimize the differences between the normalized waiting times of successively departing packets.* Suppose, hypothetically, that WTP manages to always reduce these differences to zero. The queueing delays of successively departing packets will be then proportional to the given DDPs,

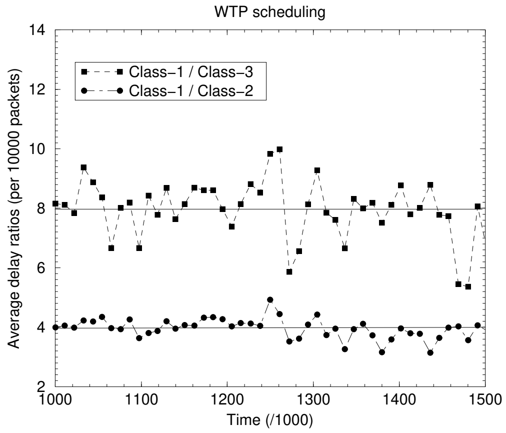
$$\frac{d^m}{d^{m+1}} = \frac{\delta_{c(m)}}{\delta_{c(m+1)}} \tag{3.16}$$

where d^m and $c(m)$ are the queueing delay and the class, respectively, of the m-th departing packet. Of course, this goal is not always feasible. Equation (3.16) shows, though, that *WTP attempts to achieve a proportional differentiation between the delays of successive packet departures.*

In terms of scalability and performance, WTP requires at most N-1 comparisons for each packet transmission, which is a minor overhead for the eight classes or so that we consider. To avoid the multiplication of the waiting time $w_i(t)$ with the factor $1/\delta_i$, the waiting time can be increased by $1/\delta_i$ after each time unit (instead of incrementing it). An implementation requirement, which also applies to PAD, is that packets have to be timestamped upon arrival



(a) Individual delays ($1/\delta_2$=5)



(b) Short-term (K=10000pks) delay ratios ($1/\delta_2$=4, $1/\delta_3$=8)

Figure 16: Individual packet delays and short-term delay ratios with WTP.

so that their delays can be measured; we do not expect this requirement to be an important difficulty in practice.

WTP is an excellent scheduler in terms of providing higher classes with lower delays in short timescales. To illustrate the behavior of WTP in short timescales, the two graphs (a) and (b) of Figure 16 show the individual delays of successive packet departures, and the ratios of short-term class delays averaged in every K=10000 packets, respectively. To compare WTP and PAD, these graphs refer to the same time interval and the same traffic stream as the graphs in Figure 15.

First, note that in Figure 16-a WTP services the Class-2 packets with lower queueing delays, even in the shortest timescales of successive packet departures. A more careful examination of that graph also shows that the individual packet delays in Class-2 are approximately proportional to the corresponding Class-1 packet delays, and that the proportionality factor is about five, which is the specified DDP ($1/\delta_2$=5). In other words, as stated in Equation 3.16, *WTP attempts to provide proportional delay differentiation between successive packet departures, and this objective is closely approximated when the delays are sufficiently large.*

Second, note that in Figure 16-b the correct class ordering is also maintained when the delays are measured over short timescales, in this graph every K=10000 packets. Figures 22 and 23 in the next section, show that this is also true in a wide span of timescales, ranging from a few packets to many thousands of packets. So, *with WTP the average queueing delays in higher classes are smaller than the average delays in lower classes, independent of the location or the duration of the averaging time window.* Also note that the measured delay ratios in Figure 16-b vary around the specified DDPs, $1/\delta_2$=4 and $1/\delta_3$=8, even though the deviations are sometimes significant.

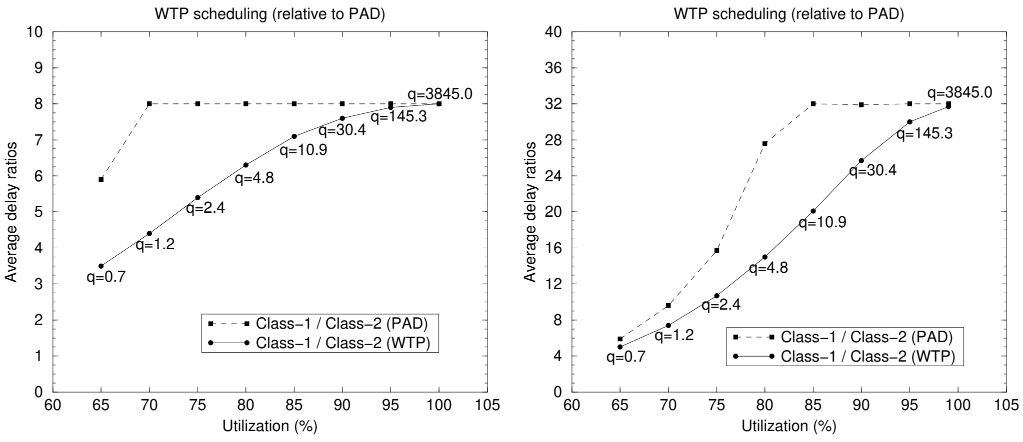
In summary, WTP meets the goals that we have set for predictable delay differentiation, and it approximates the PDD model in short timescales. Is WTP also controllable, however, and specifically does it meet the PDD constraints of (3.1)?

In the special case of Poisson arrivals, we can show analytically that WTP converges to the PDD model as the utilization approaches 100% (heavy load conditions). The following result is proved in §7.3.

Proposition 3.1: If the packet arrivals in each class are generated from a Poisson distribution, the WTP scheduler meets the PDD model of (3.1) as the utilization u tends to 100%,

$$\frac{\bar{d}_i^{WTP}}{\bar{d}_j^{WTP}} \rightarrow \frac{\delta_i}{\delta_j} \qquad \text{as} \quad u \rightarrow 1 \tag{3.17}$$

Since the Poisson assumption has been shown to be often invalid in packet networks [82, 116], we have also examined the validity of Equation (3.17) with simulations, using the infinite-variance Pareto interarrival distribution. The general observation from this empirical study is that *WTP meets the PDD model as the aggregate backlog in the PFE $\bar{q}_{ag}$ tends to infinity, but not always when the utilization tends to 100%.* In the case of Poisson arrivals, the aggregate backlog tends to infinity as the utilization tends to 100%, which is in agreement with Proposition (3.1). For other arrival distributions, however, it is not always true that the aggregate backlog tends to infinity as the utilization increases. In the extreme case that the traffic is periodic, the aggregate backlog is zero even when the utilization is 100%.



(a) $1/\delta_2 = 8$ (b) $1/\delta_2 = 32$

Figure 17: Average delay ratios with WTP and PAD as a function of the utilization.

How close to ‘infinity’ does the aggregate backlog have to be, though, in order for WTP to closely approximate the PDD model? Figure 17 shows the achieved delay ratios for the case of two classes and for a uniform load distribution, as a function of the utilization. The traffic is generated from an infinite variance Pareto distribution, as described in §7.1. The aggregate backlog for each utilization point is also shown (denoted by q in the graphs). Note that the aggregate backlog does not depend on the load distribution or on the specified DDPs. The corresponding curves for PAD are also shown, for comparison purposes.

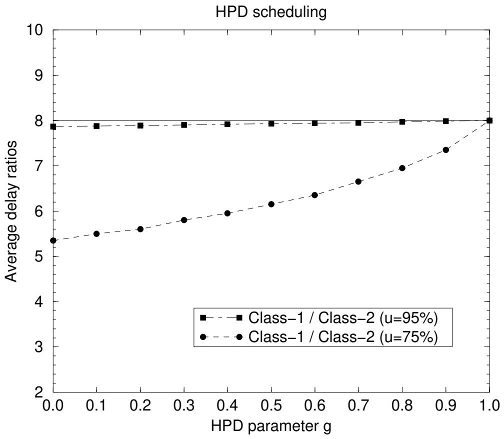
Figure 17 shows that WTP tends to the PDD model as the utilization approaches 100% and the aggregate backlog increases. For a moderate delay ratio, such as $1/\delta_2$=8, the required average backlog for a close approximation of the specified delay ratio is a few tens of packets. This is not an unusual average backlog for heavily utilized, high-speed links [90]. In order to closely approximate a more drastic delay differentiation with WTP however, such as $1/\delta_2$=32, an average backlog of hundreds of packets is needed. This may be impractical, especially when there are not enough buffers, or when the network operators prefer to drop packets (and so limit their maximum delay) instead of queueing them for hundreds of packet transmission times. In moderate load conditions, say below 85%, WTP shows a large deviation from the PDD model, either because the specified DDP is infeasible (as in Figure 17-b for $u \approx$65%), or simply because it is not a particularly good scheduler in terms of meeting the long-term average delay ratios of the PDD model.

To summarize, WTP is an excellent scheduler in terms of providing predictable delay differentiation even in short timescales. It also approximates the PDD model in heavy load conditions, when the average backlog is sufficiently large for the given DDPs. For lower load conditions, on the other hand, WTP deviates significantly from the PDD model. In the next section, we combine the operation of PAD and WTP, in order to create a hybrid scheduler that combines the features of these two schedulers.

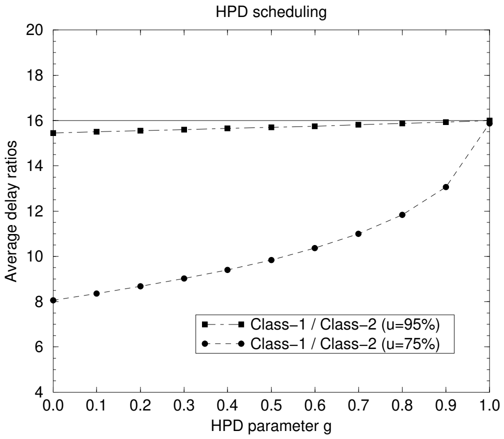
3.4 Hybrid Proportional Delay (HPD) scheduling

As shown in §3.2, PAD attempts to minimize the differences between the normalized average class delays. This objective makes PAD capable of meeting the PDD model when the given DDPs are feasible. WTP, on the other hand, attempts to minimize the differences between the normalized head waiting times. This objective makes WTP capable of approximating the PDD model in successive packet departures. As a consequence of this behavior, WTP provides higher classes with lower delays even in the shortest timescales. Can we combine the operation of these two scheduling algorithms in order to create a scheduler that is both close to the PDD model, when the DDPs are feasible, and also that provides a predictable delay differentiation in short timescales? This is the objective of the *Hybrid Proportional Delay (HPD)* scheduling discipline, described next.

HPD, just like PAD and WTP, maintains a delay metric for each class i, normalized by the corresponding DDP δ_i. Specifically, if $\tilde{d}_i(t)$ is the normalized average delay in class i at t, as defined in (3.12), and $\tilde{w}_i(t)$ is the normalized head waiting time in class i at t, as defined in



(a) $1/\delta_2$=8



(b) $1/\delta_2$=16

Figure 18: Effect of the HPD parameter g on the average delay ratios.

(3.14), the *normalized hybrid delay* in HPD is

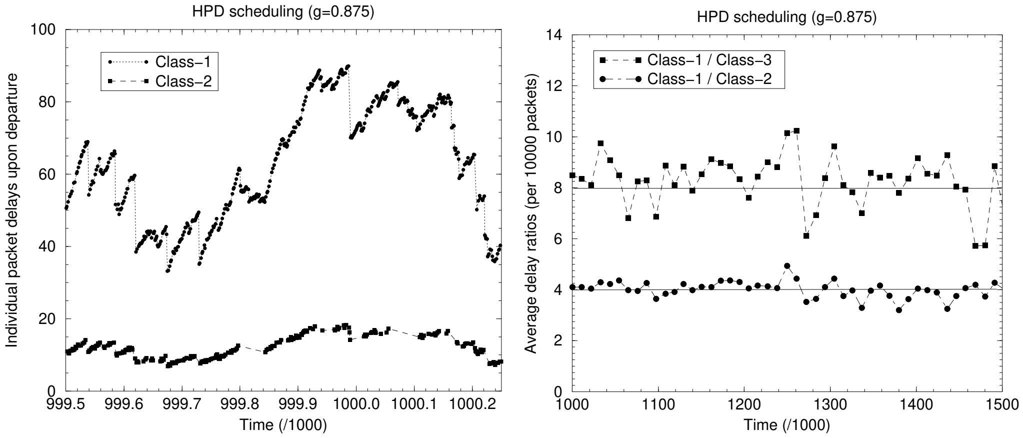
$$\tilde{h}_i(t) = g\tilde{d}_i(t) + (1-g)\tilde{w}_i(t) \tag{3.18}$$

where g is the *HPD parameter* ($0 \leq g \leq 1$). When a packet is to be transmitted, HPD chooses the backlogged class j with the maximum normalized hybrid delay,

$$j = \arg \max_{i \in B(t)} \tilde{h}_i(t) \tag{3.19}$$

When g is zero HPD becomes equivalent to WTP, while when g is one HPD becomes equivalent to PAD. For other values of g, HPD combines the operation of PAD and WTP resulting in a hybrid behavior. In the following, we examine different aspects of the HPD behavior and compare it with PAD and WTP using simulation results.

Choosing a 'good' HPD parameter value: Figure 18 shows the effect of the HPD parameter g on the ratio of average delays for two classes. In heavy load conditions (u=95%), the selection of g does not matter much, and the PDD model is closely approximated for practically any g.



(a) Individual delays ($1/\delta_2$=5)

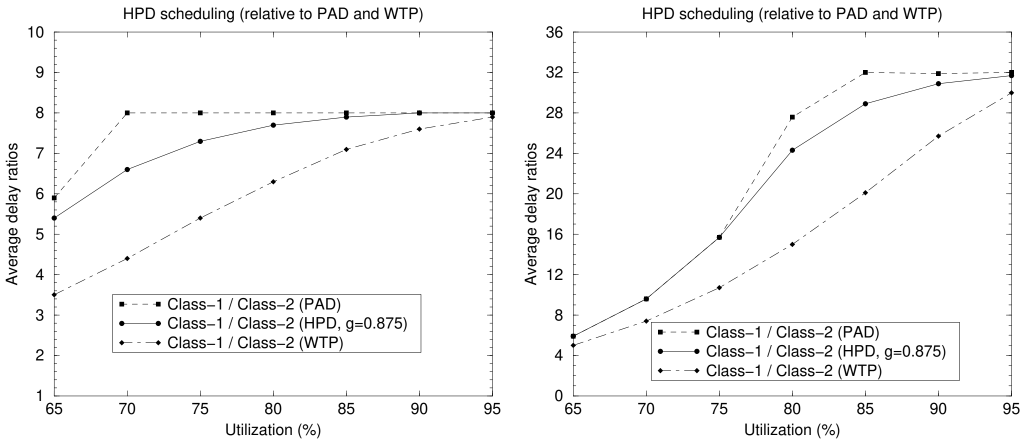
(b) Short-term (K=10000pks) delay ratios ($1/\delta_2$=4, $1/\delta_3$=8)

Figure 19: Individual packet delays and short-term delay ratios with HPD.

The reason is that in heavy load conditions both PAD and WTP meet the PDD model, and so does HPD. In lower load conditions (u=75%) though, g needs to be close to one in order for HPD to approximate the PDD model well. Note that the dependency of the delay ratio on g is convex, meaning that the approximation error decreases faster as g approaches one. Of course, a good value of g has to also take into account the predictability of HPD, which improves as g decreases and HPD behaves more like WTP.

Figure 19 shows the same sample paths of individual delays and short-term delay ratios for HPD, as Figure 16 shows for WTP, and Figure 15 shows for PAD. The HPD parameter is set to g=0.875. Comparing carefully the two graphs in Figures 19-a and 16-a shows that, with g=0.875, HPD provides almost indistinguishable results with WTP. Similar experiments and comparisons of different sample paths for other values of g have shown that a lower value of g does not add to HPD's predictability. A much higher value of g, on the other hand, larger than 0.95 or so, causes occasional predictability problems, similar to the PAD behavior shown in Figure 15-a.

In summary, we have found that g=0.875 is a 'good' value in the trade-off between the



(a) $1/\delta_2$=8 (b) $1/\delta_2$=32

Figure 20: Average delay ratios with HPD as a function of the utilization.

behaviors of PAD and WTP. In the following, g is set to 0.875 unless stated otherwise. Note that 0.875=1-2^{-3}, which also means that the multiplication with g and $1-g$ can be replaced with shift and subtract operations.

Effect of the aggregate load on the average delay ratios: Figure 20 shows the effect of the utilization u on the HPD average delay ratios. Results are shown for two classes, when the specified delay ratio is either $1/\delta_2$=8 or $1/\delta_2$=32. The load distribution between the two classes is uniform. For comparison purposes we also show the PAD and WTP curves.

As the utilization increases, all three schedulers tend to the specified delay ratios. The differences between HPD and PAD in heavy load conditions, above 90% or so, are minor. In moderate load conditions, between 70% to 90%, HPD is significantly closer to the PDD constraints than WTP. The largest relative error between HPD and PAD in this operating region is about 20% in these graphs. This error can be further reduced with a higher HPD parameter, if the network operator values the importance of meeting the specified DDPs more than providing a consistent delay ordering between classes in short timescales. In light load conditions, less than about 70% or 75%, the PDD model becomes infeasible, and so the PAD scheduler cannot achieve the specified delay ratios either.

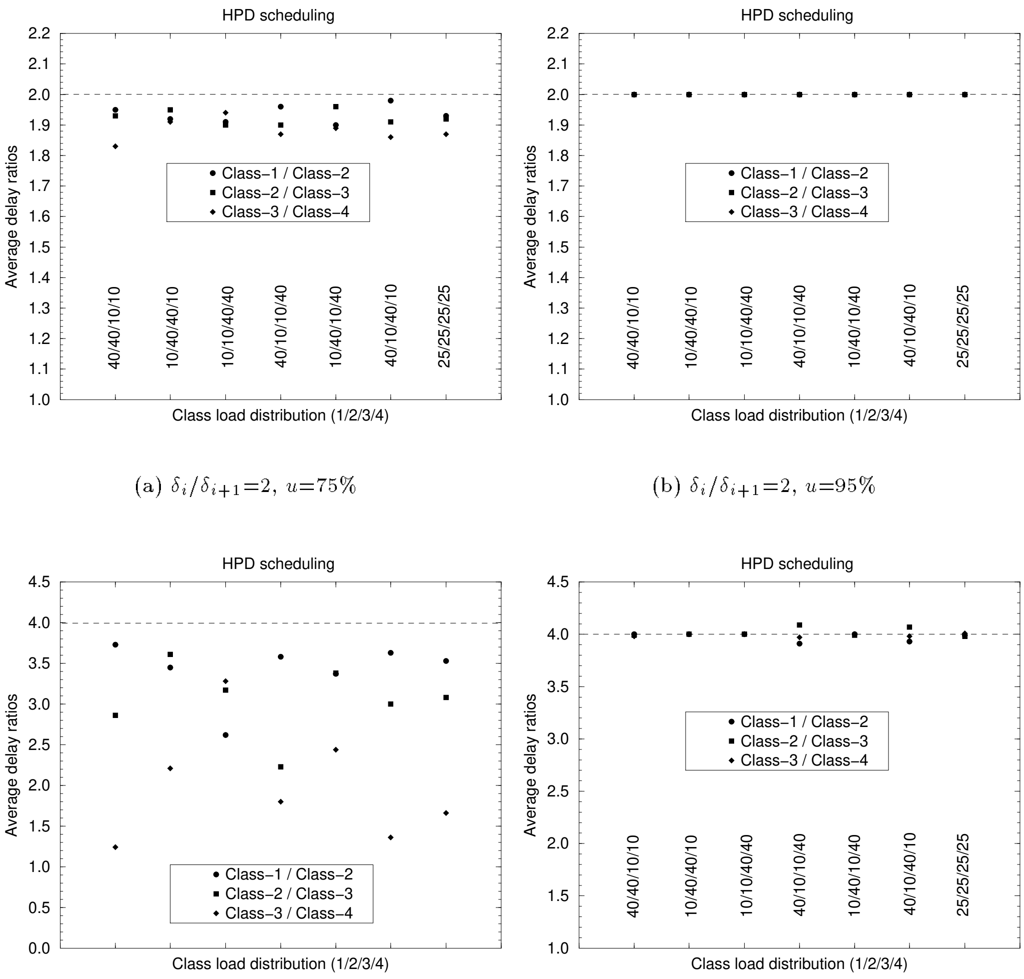
Effect of the load distribution on the average delay ratios: Figure 21 shows the effect of the class load distribution on the HPD average delay ratios. Results are shown for four classes and seven class load distributions in light load conditions (u=75%) and in heavy load conditions (u=95%). In Figures 21-a and 21-b the DDPs are: $\delta_1/\delta_2 = \delta_2/\delta_3 = \delta_3/\delta_4$=2, i.e., each class should have a twice as large average delay than the next higher class. In Figures 21-c and 21-d, on the other hand, the required delay differentiation is set to a factor of four between successive classes, i.e., $\delta_1/\delta_2 = \delta_2/\delta_3 = \delta_3/\delta_4$=4.

Ideally, HPD should meet the PDD model independent of the class load distribution. The first set of DDPs (δ_i/δ_{i+1}=2, i=1,2,3) is feasible in both load conditions, because PAD (not shown here) can meet the PDD model. In moderate load conditions (Figure 21-a), HPD closely approximates the specified delay ratio between successive classes, and the deviations from the PDD model are less than 10%. In heavy load conditions (Figure 21-b), HPD accurately meets the specified delay ratios in all load distributions.

The second set of DDPs (δ_i/δ_{i+1}=4, i=1,2,3) is infeasible when u=75%, because PAD cannot meet the PDD model. Since the DDPs are infeasible in that case (Figure 21-c)[7], HPD exhibits large deviations from the specified PDD model; it is noted that PAD exhibits similar deviations. In heavy load conditions (Figure 21-d), on the other hand, the DDPs are feasible and HPD manages to closely approximate the PDD model without any significant deviations. In general, HPD manages to closely approximate the PDD model, when it is feasible, *independent of the class load distribution.* The approximation errors increase as the utilization decreases, and as the required delay differentiation between classes becomes more extreme.

Delay differentiation with HPD and WTP in short timescales: In evaluating the predictability of a scheduler, our primary focus is to examine whether higher classes encounter lower delays in all timescales. It has already been illustrated in the previous graphs that HPD provides higher classes with lower delays, but only in terms of long-term averages. Is the delay ordering

[7] The class load distributions in this graph are as in the graph of Figure 21-d.



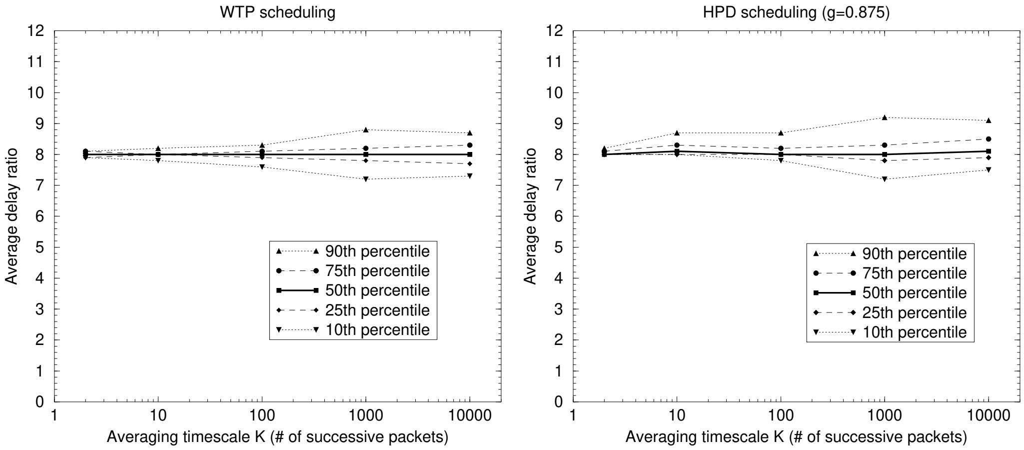
(a) δ_i/δ_{i+1}=2, u=75%

(b) δ_i/δ_{i+1}=2, u=95%

(c) δ_i/δ_{i+1}=4, u=75%

(d) δ_i/δ_{i+1}=4, u=95%

Figure 21: Average delay ratios with HPD as a function of the load distribution.



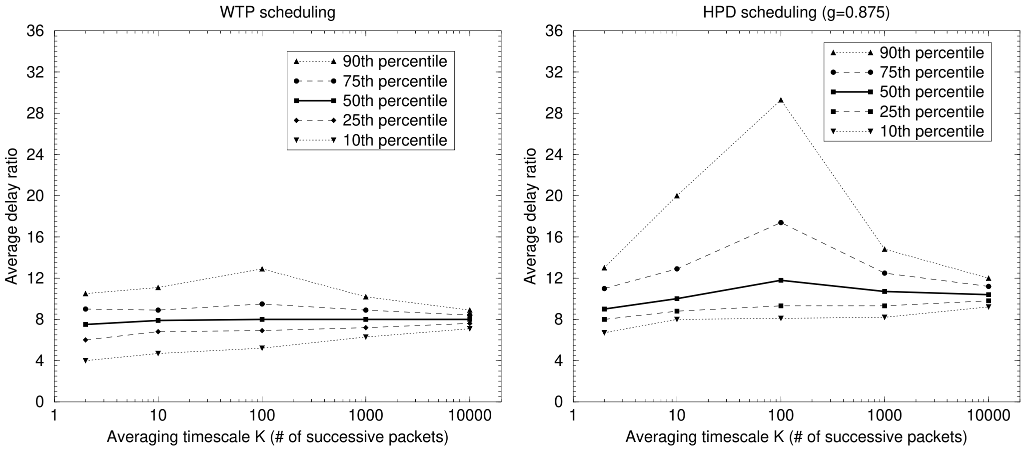
(a) WTP, $1/\delta_2$=8

(b) HPD, $1/\delta_2$=8

Figure 22: Percentiles of the average delay ratios with WTP and HPD as a function of the averaging timescale K (u=95%).

between classes, however, also met in short timescales?

Figures 22 and 23 investigate the delay differentiation that HPD and WTP produce in short timescales of different length. The graphs in these two figures show *five percentiles of the distribution of delay ratios between two classes, when the delay ratios are measured in successive time windows of K packet departures.* K determines the timescale in which we measure the class delays and compute their ratio. In these experiments, K ranges from successive packet departures (K=2) to several thousands of packet departures (K=10000). The five percentiles shown give a comprehensive view of the distribution of the resulting delay ratios, providing the median delay ratio (50th percentile), the 25th and 75th percentiles, as well as two tail delay ratios (10th and 90th percentiles). Note that when there are no packet departures from both classes in a time period of K packet departures, a delay ratio is not measured. Also, the duration of the simulation runs is adjusted so that we get approximately the same number of delay ratios for all values of K. In both figures, the specified DDP is δ_2=1/8 and the class load distribution is uniform.



(a) WTP, $1/\delta_2$=8 (b) HPD, $1/\delta_2$=8

Figure 23: Percentiles of the average delay ratios with WTP and HPD as a function of the averaging timescale K (u=80%).

Figure 22 shows the results of these experiments in heavy load conditions (u=95%). As shown, the resulting distributions of delay ratios are quite narrow, centered around the specified DDPs δ_1/δ_2=8, in both HPD and WTP. This means that in heavy load conditions, HPD inherits the excellent predictability of WTP, i.e., it provides lower delays to higher classes, and it approximates closely the PDD model. This behavior is consistent in a range of timescales, from successive departures to several thousands of packet departures. In Figure 23, on the other hand, the utilization is u=80%, and the load is moderate. In this case, the HPD delay ratio distributions are significantly more spread than the WTP distributions. Additionally, the HPD distributions are not centered exactly around the specified δ_1/δ_2=8, but they indicate slightly higher delay ratios. Even if HPD does not approximate the PDD model as closely as WTP does in moderate or low load conditions, however, it still provides lower delays to the higher class, and so it produces predictable delay differentiation.

A summary of the previous simulation study follows. In heavy load conditions, above 90% or so, both PAD and WTP meet the PDD model, and so does HPD. In lower loads conditions, HPD is closer to the PDD model than WTP, but it does not achieve the optimal behavior of PAD. HPD closely approximates the PDD model, when it is feasible, *independent of the class load distribution*. The approximation errors increase as the utilization decreases, and as the required delay differentiation between classes becomes more extreme. HPD does not have the predictability problems that PAD has. This means that HPD manages to provide lower delays to higher classes even in short timescales, ranging from successive packet departures to several thousands of packet departures. HPD, however, does not perform as well as WTP in approximating the PDD model in short timescales when the load conditions are moderate or low.

3.5 Related work on delay differentiation

In this section, we review other scheduling algorithms for relative or proportional delay differentiation model. We start with the family of *link sharing schedulers*, showing that they can provide controllable delay differentiation, but they are too sensitive to class load distribution changes. The next two schedulers, ADD and BPR, were presented in our original publications on delay differentiation ([30] and [31]). The final part of this section presents recent contributions made by other researchers, extending our work on proportional delay differentiation.

3.5.1 Link sharing schedulers

Several packet schedulers aim to *provide each class with a minimum bandwidth share of the link's capacity*. Examples of such schedulers are the packet-based approximations of GPS (§2.2) such as WFQ [29], as well as the Class Based Queueing (CBQ) [44] and Hierarchical Packet Fair Queueing (H-PFQ) [7] link sharing schedulers. These mechanisms were developed in the link sharing context, where different organizations or users are guaranteed a certain fraction of a link's capacity, sharing any available excess bandwidth. The link sharing schedulers have been also proposed for providing relative delay differentiation. For example, Heinanen suggests the use of such a scheduler to implement the *Olympic service model*, which consists of the 'Gold', 'Silver', and 'Bronze' classes [48].

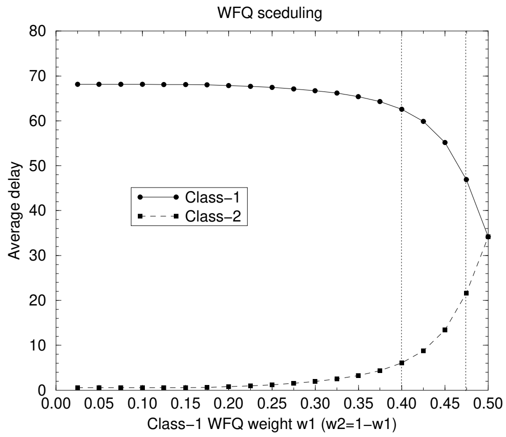
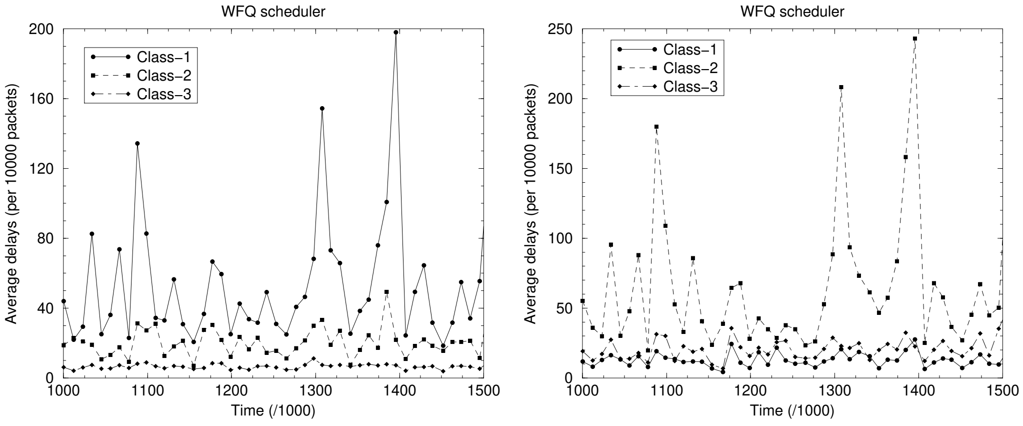


Figure 24: Average delays with a two-class WFQ scheduler as a function of the weight w_1.

To illustrate the use of link sharing weights as differentiation knobs, Figure 24 shows the resulting average delays for a two-class WFQ scheduler[8], as a function of the Class-1 weight w_1. The weight of Class-2 is fixed to w_2=1-w_1. The utilization is set to u=90% and the load distribution is uniform, i.e., (λ_1, λ_2)=(50,50). When $w_1 = w_2$, WFQ behaves as a First-Come First-Serve (FCFS) scheduler and it provides the same average delay to each class. When $w_1 = 0$ WFQ behaves as a Strict Priority (SP) scheduler that services Class-2 as the high priority class. For a more moderate differentiation, $w_1 \approx 0.475$ makes the average delay in Class-1 about twice as large as in Class-2, while $w_1 \approx 0.40$ makes the average delay in Class-1 about ten times larger than in Class-2. This example shows that WFQ is a controllable delay differentiation scheduler.

A drawback of the link sharing schedulers is that slight changes in the class load distribution affect dramatically the resulting delay differentiation. Figure 25 shows short-term average delays in a three-class WFQ scheduler, measured in every K=10000 packet departures. The WFQ weights are selected as (w_1, w_2, w_3)=(0.5,0.35.,0.3) so that, when the load distribution is $(\lambda_1, \lambda_2, \lambda_3)$=(50,30,20), the average delay in Class-1 is about 50, in Class-2 about 20, and in Class-3 about 7 time units. In Figure 25-a, notice that the selected WFQ weights create a predictable delay differentiation, providing the maximum delays to Class-1 and the lowest delays to Class-3.

[8] In the context of this discussion, the differences between GPS, WFQ, and the other implementations of GPS are not important.



(a) $(\lambda_1, \lambda_2, \lambda_3)$=(50,30,20)

(b) $(\lambda_1, \lambda_2, \lambda_3)$=(40,35,25)

Figure 25: Average delays with a three-class WFQ scheduler in two slightly different load distributions.

If, however, only 10% of the Class-1 traffic moves to Class-2 and Class-3, leading to the load distribution $(\lambda_1, \lambda_2, \lambda_3)$=(40,35,25), the class ordering is violated, causing the maximum delays for Class-2 and the minimum delays for Class-1 (see Figure 25-b). Obviously, such a 'priority inversion' between the offered classes would be unacceptable for the higher class users. This example illustrates that link sharing schedulers are quite sensitive to changes in the class load distribution. One way to deal with this problem is to dynamically adjust the link share weights based on the current class loads; this is the basic idea behind the following scheduler.

3.5.2 Backlog Proportional Rate (BPR) scheduling

BPR is a dynamic version of GPS (§2.2), in which *the class weights are dynamically adjusted based on the instantaneous class loads.* Specifically, let $r_i(t)$ be the service rate that is assigned to queue i at time t. For two backlogged queues i and j, the service rate allocation in BPR follows the proportional constraint:

$$\frac{r_i(t)}{r_j(t)} = \frac{\delta_j}{\delta_i}\,\frac{q_i(t)}{q_j(t)} \tag{3.20}$$

where $q_i(t)$ is the backlog of queue i at time t. If the queue i is empty at time t, $r_i(t) = 0$. The sum of the assigned service rates $\sum_{i=1}^{N} r_i(t)$ must be equal to the link capacity C when the scheduler is busy. We first studied BPR in [31].

The main finding of [31] is that BPR approximates the PDD model of Eq 3.2 under heavy load conditions. This asymptotic result can be explained informally as follows. In heavy load conditions, all BPR queues are almost always backlogged, due to the 'simultaneous queue clearing' property stated and proved in [31]. So, if we write (3.20) in terms of averages and use Little's law $\bar{q} = \lambda\bar{d}$, we have that:

$$\frac{r_i}{r_j} = \frac{\delta_j}{\delta_i}\frac{\bar{q}_i}{\bar{q}_j} = \frac{\delta_j}{\delta_i}\frac{\bar{d}_i\lambda_i}{\bar{d}_j\lambda_j} \tag{3.21}$$

where r_i is the average output (service) rate in class i. In steady-state and lossless conditions, however, the output rates r_i are equal to the input rates λ_i ($r_i = \lambda_i$), and so BPR tends to the PDD constraints,

$$\frac{\bar{d}_i^{BPR}}{\bar{d}_j^{BPR}} = \frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j} \tag{3.22}$$

Even though BPR tends to the PDD model, [31] showed that it does not do so as well as WTP. Specifically, the deviations from the PDD model for the same load conditions are higher, and the delay ratios in short timescales have a wider distribution. In addition, BPR suffers from the *simultaneous queue clearing* property, according to which all queues in a BPR busy period become empty at the same time. This effect causes occasional delay bursts in the high class packets.

3.5.3 Additive Delay Differentiation (ADD)

ADD is a non-preemptive priority scheduler in which the priority of the packet at the head of queue i at time t is

$$p_i(t) = w_i(t) + s_i \tag{3.23}$$

where $w_i(t)$ is the waiting time of that packet at time t, and $0 < s_1 < s_2 < \ldots < s_N$ are the ADD parameters. Note that ADD is similar to WTP, with the difference that the priority of a packet does not increase proportionally with its waiting time, but additively. We first presented ADD in the context of relative delay differentiation in [30] and [31].

The ADD scheduler was also presented in [13], together with the following expression for the average queueing delay of class i in heavy load conditions, assuming Poisson arrivals and that all classes have the same average packet size,

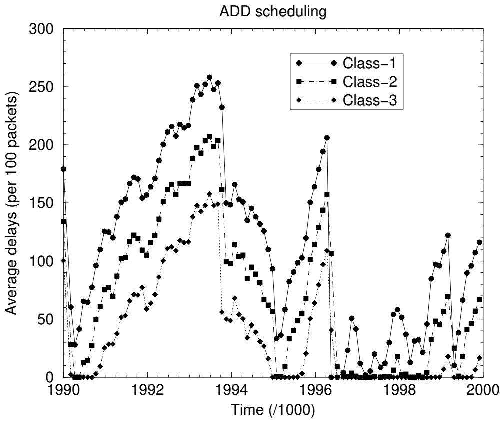
$$\bar{d}_i = \bar{d}_0/(1-\lambda) - P_q \sum_{k=1}^{N} \lambda_k (s_i - s_k) \tag{3.24}$$

P_q here is the probability that an arriving packet has to wait in the queue, and $\bar{d}_0$ is the mean remaining service time of the packet that is being transmitted when a new packet arrives. As the utilization u tends to 100% ($\lambda \to C$=1), P_q tends to one, and we can show that the ADD scheduler tends to the following *additive delay differentiation*,

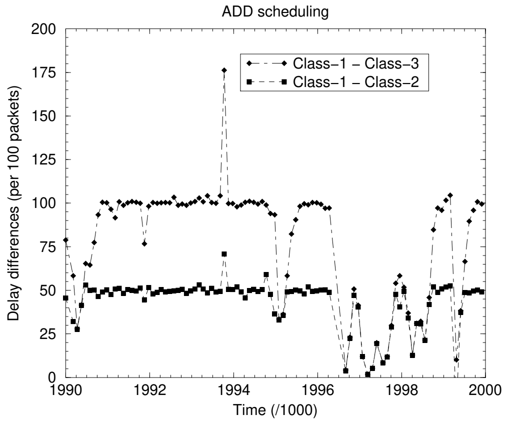
$$\bar{d}_i - \bar{d}_j \to s_j - s_i \tag{3.25}$$

The heavy load result of Equation 3.25 appears to hold even with non-Poisson arrivals. Figure 26 shows the short-term average delays and delay differences, averaged every K=100 packets, for three classes, with Pareto distributed interarrivals. The utilization in this simulation run is set to 95%, the load distribution to $(\lambda_1, \lambda_2, \lambda_3)$ = (50,30,20), while the two graphs shown cover the same time window. Note that when the queueing delays are adequately large, the delay of Class-1 packets is about 50 time units larger than the delay of Class-2 packets and about 100 time units larger than the delay of Class-3 packets. This differentiation matches the corresponding differences of the ADD parameters: s_3-s_2=50 and s_3-s_1=100, as stated in (3.25). On the other hand, when the delays are not adequately large, the delay differences are less than what is specified, implying that the ADD parameters are infeasible during those time periods.

We mention the ADD scheduler and the additive differentiation model here, as an interesting case of another relative differentiation model that deserves further investigation in the



(a) (s_1, s_2, s_3)=(0,50,100)



(b) (s_1, s_2, s_3)=(0,50,100)

Figure 26: Short-term (K=100pks) average delays and delay differences in ADD.

future.

3.5.4 Recent contributions on proportional delay differentiation

The WTP scheduler received further attention in [59]. The authors, starting from Kleinrock's result [54] on the average class delays in WTP (see Equation 7.8), derived the feasible load distribution range for a given set of DDPs. They also proposed a numerical method for calculating the scheduling weights (when N >2), so that WTP can achieve a feasible set of DDPs for any given load conditions. For N=2, they showed that WTP can achieve a DDP δ_2 only when the utilization is $u > 1 - \delta_2$. Also, given a DDP δ_2, the WTP proportional priority-increase factors should be chosen as $b_1 = 1$ for Class-1, and $b_2 = u/(u - 1 + \delta_2)$ for Class-2. The results of [59], however, are only valid in the case of Poisson arrivals, since the WTP analysis by Kleinrock was based on that assumption.

Another scheduler for proportional delay differentiation, called *Mean-Delay Proportional (MDP)*, was proposed in [72]. MDP is similar to HPD, in the sense that it also chooses for service at time t the class with the maximum normalized 'delay' metric $\tilde{D}_i(t)$. The delay metric in MDP, though, is

$$\tilde{D}_i(t) = \frac{1}{\delta_i} \frac{S_i + 1/2Q_i(1+Q_i)}{P_i + Q_i} \tag{3.26}$$

where S_i and P_i are defined as in PAD (§3.2), and Q_i is the number of backlogged packets in class i at t. For a unit packet size and transmission rate, the term $1/2\,Q_i(1+Q_i)$ is a lower bound on the cumulative delays that the backlogged packets in class i at t will experience before being serviced. In other words, the average delay metric in MDP takes into account both the previously departed packets (through S_i and P_i), and also all the currently backlogged packets. Note that as the history of departing packets develops (and P_i increases), the effect of the currently backlogged packets diminishes (because $Q_i \ll P_i$), and gradually the scheduler behaves in the same way as PAD. The authors of [72] mention a number of heuristics that can be applied to MDP, such as periodic re-initialization of the counters S_i and P_i, or computation of the average class delays in fixed-length time windows, but they do not give details on how these heuristics affect the scheduler's behavior.

[70] and [60] proposed two proportional delay schedulers (called *Proportional Queue Control Mechanism (PQCM)* and *Dynamic Weighted Fair Queueing (D-WFQ)*, respectively) which are based on the GPS rate allocation mechanism (§2.2). Both schedulers adjust the GPS weights periodically, say every Δ seconds, based on the measured class backlogs and input rates, attempting to achieve proportional average delay differentiation in the next window of Δ seconds. PQCM uses the instantaneous backlogs to adjust the weights, while D-WFQ uses exponential averaging estimators for the computation of the backlogs and input rates. Note that PQCM and D-WFQ are similar to BPR, but with an important difference. BPR adjusts the weights after each packet departure, and so it is not based on a fixed window Δ. PQCM and D-WFQ, on the other hand, perform the rate adjustments periodically, and so they introduce a feedback loop in the network, according to which the future service rates depend on the input rates over a previous time interval. This feedback loop in the network can interact with the feedback loop in the applications or users, when the latter attempt to dynamically choose an acceptable class. If the delay Δ in the PQCM or D-WFQ feedback loop is comparable to the delay of the dynamic class selection process, the two feedback loops may interact causing instability effects. This issue

is further discussed in the context of Dynamic Class Selection (DCS) in §5.5.

Other researchers proposed different relative differentiation models than the PDD model of §3.1, and studied scheduling mechanisms that can implement them. For instance, [95] proposed the *Local Optimal Proportional Differentiation (LOPD)* scheduler, which attempts to provide proportional average delay differentiation *in each busy period*, assuming that there will be no further arrivals after each packet departure. LOPD is optimal for this differentiation objective of [95]. [10] extended the PDD model in the direction of *deadline violation probabilities*, proposing a scheduler that aims for proportional delay violation probabilities among classes. The scheduler is similar to Earliest Due Date (EDD), but the deadlines are dynamically adjusted based on the fraction of packets that missed their deadline in each class. Finally, [61] proposed a *Joint Buffer management and Scheduling (JoBS)* mechanism which determines both the scheduling order and the packet drop decisions. JoBS can be used for both relative and absolute services. The basic idea is that JoBS makes a prediction after every packet arrival for the delays of the backlogged traffic, and modifies the service rates so that all QoS constraints are met. If this is not possible, some packets are dropped.

3.6 Summary and extensions

Our objective in this chapter was to apply the general model of proportional differentiation in the context of queueing delays. In the first part of the chapter, we proposed and studied the Proportional Delay Differentiation (PDD) model. The PDD model controls the ratios of the average queueing delays between classes based on the specified Delay Differentiation Parameters (DDPs). Starting from the PDD model, we derived the average queueing delay in each class given a class load distribution, showed certain dynamic properties for the class delays in PDD, and stated the conditions under which the PDD model is feasible. The feasibility model of the model can be determined from the average delays that result with the Strict Priorities (SP) scheduler.

In the second part of the chapter, we designed three schedulers for the PDD model. The Proportional Average Delays (PAD) scheduler appears to always meet the PDD model when it is feasible, and so we conjecture that PAD is optimal. PAD, however, exhibits unpredictable behavior in short timescales. The Waiting Time Priorities (WTP) scheduler, on the other hand, approximates the PDD model closely even in short timescales, but only in heavy load conditions. A third scheduler, called Hybrid Proportional Delays (HPD), combines the operation of PAD and WTP. Our simulation study showed that, in heavy load conditions, above 90% or so, both PAD and WTP meet the PDD model, and so does HPD. In lower loads conditions, HPD is closer to the PDD model than WTP, but it does not achieve the optimal behavior of PAD. HPD approximates the PDD model closely, when it is feasible, independent of the class load distribution. Also, HPD manages to provide predictable delay differentiation (i.e., lower delays to higher classes) even in short timescales. HPD, however, does not perform as well as WTP in approximating the PDD model in short timescales, when the load conditions are moderate or low.

Our focus throughout this study was on the *design* of scheduling algorithms that can achieve controllable and predictable delay differentiation, rather than on the *analysis* of these algorithms. Consequently, in several points our conclusions are based on empirical evaluation using simulations. Future work can pursue a more mathematical treatment of these schedulers, possibly making some simplifying assumptions on the traffic model or on the algorithms themselves. Specifically, the following issues deserve further investigation.

- The feasibility conditions for the PDD model can probably be analyzed in greater depth for the general case of $N > 2$. If the SP delays $\bar{d}_k^{SP}$ are known for a certain traffic model, one can derive the feasible space of DDPs given the load distribution, or determine the feasible load distribution space given a certain set of DDPs. [59] focused on this problem assuming Poisson arrivals, but for the WTP scheduler instead of the general PDD model.
- It is likely that the PAD scheduler can be proven to be optimal in meeting the PDD model within a certain family of scheduling algorithms. This family would probably include all work-conserving schedulers that do not make decisions based on future arrivals (*causality*) and that service each class in a FCFS manner. This issue deserves further investigation.
- We proved that WTP tends to the PDD model for the case of Poisson arrivals. It is likely

that in heavy load conditions this assumption is not required, as long as the aggregate average backlog tends to infinity. It is also likely that in heavy load conditions WTP not only achieves the PDD model, but it also provides proportional delay differentiation in successive packet departures, at least in an average sense (i.e., averaging over all successively departing packet pairs).

Chapter 4

Proportional Loss Differentiation

The subject of this chapter is the loss differentiation between traffic classes, and the related buffer management and packet dropping problem. We first propose the Proportional Loss Differentiation (PLD) model as a means for controllable and predictable loss rate differentiation. The PLD model is similar in several aspects to the PDD model, such as the model formulation and objectives. There are also some important differences however, that become apparent in the design of dropping algorithms for the PLD model. We propose and evaluate two Proportional Loss Rate (PLR) algorithms. The two droppers, $PLR(\infty)$ and $PLR(M)$, differ in the interval over which the loss rates are measured and proportionally adjusted. This difference causes a trade-off between the two droppers in terms of their implementation complexity, accuracy in meeting the PLD constraints, and adaptability to varying class load distributions. We also examine the coupled effect of delay and loss rate proportional differentiation on the throughput of bulk-transfer TCP connections. The chapter closes with a review of other buffer management and packet dropping algorithms in the context of relative or proportional loss differentiation.

4.1 Proportional Loss Differentiation (PLD) model

The lossless model of a Packet Forwarding Engine (PFE) was described in §3.1. In that model, a FCFS packet queue is maintained for each of the N classes of service. The N queues share the same pool of packet buffers, that in §3.1 was assumed to be unlimited. A *scheduler*, which is one of the PFE modules, determines the order in which packets are selected for transmission, providing a certain delay differentiation between classes. In this section, we extend the PFE

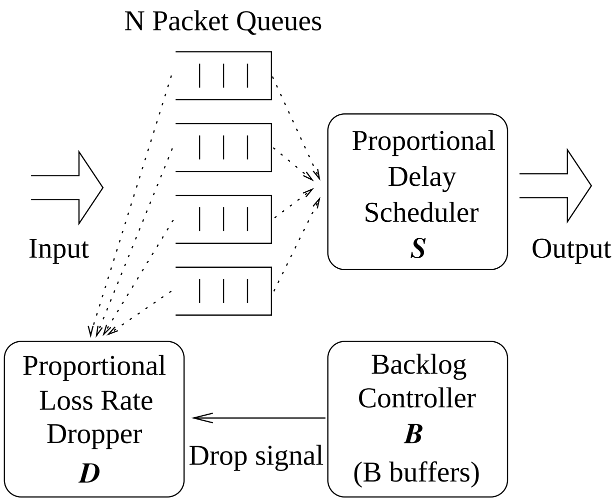


Figure 27: The lossy model of a Packet Forwarding Engine (PFE).

model to capture the possibility of packet losses, imposing a limit B on the number of available packet buffers in the PFE.

Packet losses occur in a PFE mainly for two reasons. First, occasionally packets arrive in such a bursty manner that the PFE does not have an adequate amount of buffers to store them. The buffer manager in the PFE is then forced to drop some of the arriving or backlogged packets. Such 'memory overflow' losses can be reduced by adding more buffers. Increasing the number of buffers, though, can make the queueing delays encountered by some packets excessive. For this reason, network operators often prefer to limit the number of buffers in a PFE and drop some packets, rather than transferring them too late.

The second cause of packet drops is related to the *reactive* nature of TCP traffic, which is the dominant part of the Internet workload today [112]. Roughly speaking, the TCP protocol reacts to a packet drop by reducing the transmission rate of the sender by a factor of two. Making use of this TCP behavior, routers can decrease the input rate to the PFE by dropping some packets, notifying the corresponding TCP senders in this manner that they should reduce their rates. This technique is called *active buffer management* [68] and it has several advantages, especially when the traffic workload consists of bulk-transfer TCP connections [43].

The lossy PFE model that we consider is shown in Figure 27. In addition to the N

packet queues and the scheduler $\mathcal{S}$, the model includes two more modules, the *backlog controller* $\mathcal{B}$ and the *dropper* $\mathcal{D}$. The backlog controller $\mathcal{B}$ determines the time instants that a packet should be dropped, either because there is no available buffer, or because the PFE uses an active buffer management mechanism. When a packet has to be dropped, the backlog controller signals the dropper to select the backlogged class that the packet should be dropped from. Consequently, $\mathcal{B}$ determines the aggregate amount of losses, as well as the time instants that losses occur in the PFE, while $\mathcal{D}$ determines the distribution of losses between classes, i.e., the loss differentiation. Note that the total number of buffers B is a parameter of the backlog controller.

The simplest type of backlog controller is called *Drop-Tail* and it signals a packet drop when there is no available buffer to store an arriving packet. More sophisticated algorithms, such as RED [43] and its variations [39, 78], are examples of other backlog controllers. An example of a simple dropper is the *Last-In First-Out (LIFO)* algorithm, which removes the last packet that entered the PFE, independent of its class. LIFO corresponds to the FCFS scheduler (§3.1), since both schemes do not discriminate between classes. An important dropper is the *Strict Priority (SP)* algorithm, which removes the packet from the tail of the lowest priority backlogged class. The SP dropper corresponds to the SP scheduler (§3.1), since both schemes create the strongest possible discrimination in favor of higher classes. As will be discussed later in this section, the SP dropper is directly related to the feasibility of the PLD model.

The assumptions stated in §3.1 for the lossless PFE model, namely that the scheduler is work-conserving and non-preemptive, and that the packet size distribution is the same in all classes, also hold for the lossy model of this section. In addition, we make the following assumptions. First, *a backlogged packet requires one PFE buffer independent of the packet size.* This assumption simplifies the PFE model, because an arriving packet can cause only one packet drop. It is not a requirement, though, for the correct operation of the droppers proposed in §4.2.

Second, we assume that *the dropper can perform pushout*, i.e., to drop a packet that is already backlogged, as opposed to drop arriving packets only. In the past, it was often assumed that pushout is hard to implement in high-speed links [55]. Recently, however, it has been shown with actual implementations of high-performance routers that pushout can be efficient [19, 108], when packets are removed from the head or the tail of the class queues.

Third, we consider droppers that *remove packets from the tail of the class queues.* Even though removing packets from the head of a queue leads to lower average queueing delays [122], this optimization would make the delay differentiation between classes dependent not only on the scheduler $\mathcal{S}$, but also on the dropping algorithm $\mathcal{D}$; we chose to avoid this complication.

Fourth, we limit our attention to a *Drop-Tail backlog controller.* The Drop-Tail backlog controller that we consider forces the drop of a backlogged packet when a packet arrives, if the number of backlogged packets is equal to the number of buffers B. This is a necessary assumption for the loss-related conservation law and the dynamics or feasibility of the PLD model, that follow next. It is not a requirement though, for the correct operation of the droppers in §4.2.

The loss-related performance metric that the differentiation is based on is the average loss rate, or simply the *loss rate.* The loss rate $\bar{l}_i$ in class i is defined as *the fraction of class i packets that have been dropped in the PFE.* So, if D_i packets have been dropped out of A_i arrivals in class i, the loss rate $\bar{l}_i$ is D_i/A_i. Similarly, the *aggregate loss rate* $\bar{l}_{ag}$ in the PFE is defined as the fraction of arrived packets that have been dropped in the PFE, independent of their class, i.e., $\bar{l}_{ag} = D/A = \frac{\sum_i D_i}{\sum_i A_i}$. Another loss-related performance metric is the *fraction of dropped bytes*, instead of packets, but this metric is not as commonly used in practice.

The *Proportional Loss Differentiation (PLD)* model states that the loss differentiation between classes has to follow certain proportional constraints, independent of the aggregate load or the class load distribution. These proportional constraints are controlled with parameters that the network operator specifies. Formally, the PLD model requires that the ratio of loss rates between two classes i and j is set to the ratio of the corresponding *Loss Differentiation Parameters (LDPs)*,

$$\frac{\bar{l}_i}{\bar{l}_j} = \frac{\sigma_i}{\sigma_j} \qquad 1 \leq i, j \leq N \tag{4.1}$$

where $\sigma_1 > \sigma_2 > \ldots > \sigma_N > 0$ are the LDPs. As in the PDD model, we choose Class-1 to be the 'reference class' and set σ_1=1. Then, the PLD model requires that the loss rate in each class

i is a certain fraction σ_i of the loss rate in Class-1, i.e.,

$$\bar{l}_i = \sigma_i \, \bar{l}_1 \qquad i = 2 \ldots N \tag{4.2}$$

independent of the aggregate PFE load, or the class load distribution.

Before studying the PLD model further, we need to revise the PDD model of §3.1 in the lossy PFE model of this section. The PDD model was stated in §3.1 assuming no losses. In the lossy PFE model that we consider here, we have to distinguish between the *offered rate* λ_i^o and the *accepted rate* λ_i^a in class i. If $\bar{l}_i$ is the loss rate in class i, the accepted rate is the fraction of the offered rate that corresponds to *serviced (i.e., non-dropped)* packets,

$$\lambda_i^a = (1 - \bar{l}_i) \, \lambda_i^o \qquad i = 1 \ldots N \tag{4.3}$$

Since packets are dropped only from the tail of the queues, the dropped packets do not affect the delay distribution of serviced packets [122]. So, if $\bar{d}_i$ is the average queueing delay of the serviced packets in class i, the PDD model is still stated as in (3.1), while the conservation law of (3.3) becomes

$$\sum_{i=1}^{N} \lambda_i^a \, \bar{d}_i = \bar{q}_{ag} \tag{4.4}$$

assuming the same average packet size $\bar{L}$=1 in all classes. The average delay in class i is

$$\bar{d}_i = \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \, \lambda_n^a} \qquad i = 1 \ldots N \tag{4.5}$$

The delay dynamics and the feasibility conditions for the PDD model are similarly modified in the lossy PFE model. In the following, when we refer to a class rate λ_i we mean the offered rate λ_i^o.

4.1.1 Per-class loss rates in the PLD model

Similar to the conservation law for the class average delays (4.4), a conservation law also holds for the class loss rates. The *loss rate conservation law* states that, given the assumptions in the previous paragraph, for any dropper $\mathcal{D}$ that causes a loss rate $\bar{l}_i$ in class i we must have that

$$\sum_{i=1}^{N} \lambda_i \, \bar{l}_i = \lambda \, \bar{l}_{ag} \tag{4.6}$$

where $\bar{l}_{ag}$ is the aggregate loss rate in the PFE with a LIFO dropper (or with any other dropper) [63]. The conservation law implies that even though a dropper $\mathcal{D}$ can affect the relative magnitude of the class loss rates, making the loss rate in one class lower than the loss rate in another class, this is a 'zero-sum' game because the weighted sum of (4.6) has to be equal to the loss rate $\bar{l}_{ag}$ of the aggregate traffic stream. Note that $\bar{l}_{ag}$ is independent of the class that packets belong to, and thus, of the scheduler $\mathcal{S}$ or the dropper $\mathcal{D}$. The aggregate loss rate $\bar{l}_{ag}$ depends only on the arriving traffic stream, on the aggregate backlog controller $\mathcal{B}$ (and so, on the number of buffers B), and on the link capacity C.

From the PLD model of (4.1) and the loss rate conservation law (4.6), it is easy to show that the loss rate in each class i is

$$\bar{l}_i = \frac{\sigma_i \, \bar{l}_{ag}}{\sum_{n=1}^{N} \sigma_n \lambda_n} \qquad i = 1 \ldots N \tag{4.7}$$

which is of the same form as (4.5) for the average delay in each class. Equation (4.7) shows that even though the PLD model consists of N-1 *relative* constraints on the class loss rates, the loss rate in each class is uniquely determined when the PLD model is applied to a traffic stream with a certain load distribution $\{\lambda_i\}$ and a certain aggregate loss rate $\bar{l}_{ag}$.

4.1.2 Loss rate dynamics in the PLD model

Based on Equation (4.7), we can state the following properties for the *loss rate dynamics in the PLD model*. The proofs of these properties follow similar derivations with the proofs of the delay dynamics in the PDD model (see §3.1.2 and §7.2), and they are not included here.

Property 6: Increasing the input rate of a class, increases (in the wide sense) the loss rate of all classes.

Property 7: Increasing the rate of a higher class causes a larger increase in the class loss rates than increasing the rate of a lower class.

Property 8: Decreasing the loss differentiation parameter of a class decreases (in the wide sense) the loss rate of that class, and increases (in the wide sense) the loss rate of all other classes.

Suppose that the class load distribution changes from $\{\lambda_n\}$ to $\{\lambda'_n\}$, with $\lambda'_i = \lambda_i - \epsilon$, $\lambda'_j = \lambda_j + \epsilon$, and $\lambda'_k = \lambda_k$ for all $k \neq i, j$ ($\epsilon > 0$). Let $\bar{l}'_n$ be the loss rate in class n when the class load distribution is $\{\lambda'_n\}$.

Property 9: If $i > j$ then $\bar{l}'_n \leq \bar{l}_n$ for all $n = 1 \ldots N$. Similarly, if $i < j$ then $\bar{l}'_n \geq \bar{l}_n$.

Property 10: If $i > j$ then $\bar{l}'_j \geq \bar{l}_i$. Similarly, if $i < j$ then $\bar{l}'_j \leq \bar{l}_i$.

4.1.3 Feasibility of the PLD model

As was the case in the PDD model, the PLD model may not be always feasible. Given the load distribution and the aggregate loss rate, the PLD model does not only specify N-1 loss rate ratios, but it also specifies the N loss rates of Equation (4.7). It may not be possible, though, to enforce a certain loss rate in each class. To see why, suppose that the dropper attempts to provide a very low loss rate to a class k. It can happen, though, that the only backlogged class k at the drop time instants is class k, and so the dropper will be forced to drop only class k packets. This scenario shows that the loss rate in class k cannot be arbitrarily manipulated.

Formally, given a traffic stream with class rates $\{\lambda_i\}$, a certain number of buffers B in a Drop-Tail backlog controller, an aggregate loss rate $\bar{l}_{ag}$, and a particular scheduler $\mathcal{S}$, we say that a set of LDPs $\{\sigma_i,\ i = 2 \ldots N\}$ is feasible if there exists a dropper $\mathcal{D}$ that can set the loss rate in each class as in (4.7). So, the set of LDPs $\{\sigma_i,\ i = 2 \ldots N\}$ is feasible if and only if the set of loss rates $\{\bar{l}_i = \sigma_i \bar{l}_{ag}/(\sum_{n=1}^{N} \lambda_n \sigma_n),\ i = 1 \ldots N\}$ is feasible. Note that the feasibility of the LDPs depends on the scheduler $\mathcal{S}$, since $\mathcal{S}$ affects the service that each class receives, and so the likelihood that a class is backlogged at the time instant of a packet drop.

Unfortunately, to the best of our knowledge, there are no results in the queueing theory literature for the feasibility of a set of class loss rates, similar to the feasibility conditions by Coffman and Mitrani or Regnier for the feasibility of a set of class average delays (§3.1.3). The basic intuition behind those delay-related feasibility conditions, however, seems to also hold in the case of loss rates. Specifically, the loss rate of a class (or of any set of classes) has a lower bound due to the inherent load in that class (or set of classes), given a certain scheduler $\mathcal{S}$ and a Drop-Tail backlog controller with B buffers. This lower loss rate bound would result in practice if that class (or set of classes) was given strictly higher priority (i.e., minimum losses) over the rest of the traffic.

In the following, we state, *as a conjecture*, a set of feasibility conditions for the N loss rates $\{\bar{l}_i\}$ that are similar to the Regnier feasibility conditions of (3.7). It remains an important open question to formally prove that these are the necessary and sufficient conditions for the feasibility of a set of class loss rates. Specifically, the conjecture is that the set of N class loss rates $\{\bar{l}_i\}$ is feasible if and only if the following N-1 conditions are true,

$$\sum_{i=k}^{N} \lambda_i \, \bar{l}_i \ \geq \ \bar{l}^{SP}_{k,N} \sum_{i=k}^{N} \lambda_i \qquad k = 2 \ldots N \tag{4.8}$$

where $\bar{l}^{SP}_{k,N}$ is the loss rate that would be encountered by classes $\{k, \ldots, N\}$ with an SP dropper, if these classes were given strictly higher priority over the rest of the traffic in terms of packet drops. These conditions impose a lower bound on the number of losses in the N-1 subsets of the k highest classes ($k = 1 \ldots N-1$).

The conditions of (4.8) show that the feasibility of the PLD model depends on the loss rates that would occur with the same scheduler and number of buffers, if the dropper $\mathcal{D}$ was replaced by the Strict Priority (SP) dropper. In the case of N=2 classes, a certain LDP σ_2 ($\sigma_2 < \sigma_1 = 1$), is feasible if and only if

$$\sigma_2 \geq \frac{\bar{l}_2^{SP}}{\bar{l}_1^{SP}} \quad \text{or} \quad \frac{1}{\sigma_2} \leq \frac{\bar{l}_1^{SP}}{\bar{l}_2^{SP}} \tag{4.9}$$

where $\bar{l}_k^{SP}$ is the loss rate of class k in the SP dropper that gives Class-2 a higher priority (i.e., a lower loss rate) than Class-1. So, a given loss rate ratio $\sigma_2 < 1$ between Class-2 and Class-1 is feasible if and only if the corresponding loss rate ratio in the SP dropper is not higher than σ_2. When $N > 2$, the feasibility conditions for the PLD model become the following N-1 inequalities

$$\sum_{i=k}^{N} \lambda_i \sigma_i \geq \frac{S}{\bar{l}_{ag}} \sum_{i=k}^{N} \lambda_i \bar{l}_i^{SP} \qquad k = 2 \ldots N \tag{4.10}$$

where $S = \sum_{i=1}^{N} \lambda_i \sigma_i$.

The network operator can check the feasibility of a certain set of LDPs by either measuring the loss rates $\bar{l}_k^{SP}$ on the actual link, or emulating the SP dropper with the workload of the actual link. This is essentially the same procedure as for checking the feasibility of a certain set of DDPs.

4.2 Proportional Loss Rate (PLR) droppers

In this section, we describe two dropping algorithms, called *Proportional Loss Rate (PLR)* droppers, that aim to meet the PLD model of §4.1 when the specified LDPs are feasible. Both PLR droppers follow a similar algorithm with the PAD scheduler of §3.2.

Specifically, another way to state the PLD model is that the *normalized loss rates*, defined as $\tilde{l}_i = \bar{l}_i/\sigma_i$, must be equal in all classes, i.e.,

$$\tilde{l}_i = \frac{\bar{l}_i}{\sigma_i} = \frac{\bar{l}_j}{\sigma_j} = \tilde{l}_j \qquad 1 \leq i, j \leq N \tag{4.11}$$

The PLR droppers maintain a running estimate $\tilde{l}_i(t)$ of the normalized loss rate in each class i.

When the backlog controller requires a packet to be dropped, the PLR droppers select *the backlogged class j with the minimum normalized loss ratio.* Dropping a packet from class j increases $\bar{l}_j$, and hence it reduces the difference of $\bar{l}_j/\sigma_j$ from the normalized loss rates of the other classes, tending to equalize them. The similarity with the PAD scheduler should now be obvious; PAD selects for transmission the backlogged class with the maximum normalized average delay, while the PLR droppers select for dropping the backlogged class with the minimum normalized loss rate. In both cases, the objective is to equalize the normalized performance metrics (average delay and loss rate) in the long run, and thus to meet the corresponding proportional differentiation constraints.

There is an important difference, however, between delay and loss rate differentiation. In the former, the fact that each packet encounters an individual queueing delay allows us to differentiate between the class delays even in the shortest timescales of a few packet departures. This motivated the study of WTP, as an appropriate scheduler for proportional delay differentiation between successive packet departures, and then the design of HPD, as a combination of the long-term differentiation of PAD with the short-term differentiation of WTP. In the case of loss rate differentiation, the performance metric (loss rate) is defined in terms of a number of packets and not on a per-packet basis. Consequently, we do not design a dropper that attempts to provide proportional loss rates between successive packet arrivals, corresponding to WTP. The issue of loss differentiation in short timescales is reflected, instead, on the interval over which the loss rates are defined and measured.

The first PLR dropper, called PLR(∞), measures the class loss rates based on all previous packet arrivals, aiming to achieve proportional loss rates in the long run ('long-term PLD model'). The second PLR dropper, called PLR(M), measures the class loss rates based on the last M packet arrivals, aiming to meet the PLD model in a shorter timescale, with a duration determined by M. As will be shown in §4.3, this difference between the two PLR droppers reflects on their accuracy in meeting the PLD model, on their implementation complexity, and on their adaptability to varying class load distributions.

$B(t)$: Set of backlogged classes at time t
A_i: Count of packet arrivals in class i up to time t
D_i: Count of packet drops from class i up to time t

Packet arrival in class i at time t:
$A_i = A_i + 1;$

Packet drop from class j at time t:
$j = \arg\min_{i \in B(t)} \frac{D_i}{A_i \sigma_i};$
$D_j = D_j + 1;$

Figure 28: Description of the PLR(∞) dropper.

4.2.1 PLR(∞): Proportional Loss Rate dropper with 'infinite' memory

In this dropper, the loss rate estimate $\bar{l}_i(t)$ is measured as the fraction of dropped packets in class i up to time t. $\bar{l}_i(t)$ can be measured using counters for the arrivals and drops in each class. Since the loss rate is estimated from the history of all previous arrivals and drops, we say that PLR(∞) has 'infinite' memory. The complete algorithm for this dropper is shown in Figure 28.

An important issue about the PLR(∞) dropper is the length of the counters A_i and D_i, and how to deal with counter overflows. One approach is to reset all counters, and enter a 'cold-start' phase, when any of the arrival counters overflow. With 32-bit counters, one of the A_i's will overflow after at least four billion packet arrivals. As will be shown in §4.3, however, it may be beneficial under certain traffic conditions to reset the counters over shorter intervals, or to reduce the length of the counters so that the dropper is more adaptive to varying class load distributions.

In terms of implementation complexity, the PLR(∞) dropper requires N multiplications (for the $A_i\sigma_i$ terms) and N divisions (for the $\frac{D_i}{A_i\sigma_i}$ terms) every time a packet needs to be dropped. These operations would be, in general, in floating-point arithmetic. An optimization that avoids the multiplications would be to increase A_i by σ_i every time a packet arrives in class i. If a single-precision division takes 50 cycles, the calculation of the normalized loss rates for 8 classes would require 400 cycles, and with a 500MHz processor the selection of the drop-target class would take about 0.8μs. This overhead would not be a major issue for network interfaces of up to 1Gbps, that transmit a (minimum-size) 40-byte packet in about 0.32μs, since a packet drop would need less than three packet transmission times.

4.2.2 PLR(M): Proportional Loss Rate dropper with memory M

In the PLR(M) dropper, the loss rate estimate $\bar{l}_i(t)$ is measured as the fraction of dropped packets from class i in the last M arrivals of the aggregate traffic stream. So, if there are $A_i(M)$ class i packets in the last M arrivals, and $D_i(M)$ of those packets were dropped, the loss rate estimate for class i is $D_i(M)/A_i(M)$. If $A_i(M)$=0, the loss rate in class i is undefined.

The implementation of PLR(M) is slightly more complex than the implementation of PLR(∞). PLR(M) requires a cyclic queue with M entries, called *Loss History Table (LHT)*. The LHT records the class index (*LHT[i].class* $\in \{1, \ldots, N\}$) and the drop status (*LHT[i].loss*: 1 if packet is dropped; 0 otherwise) for each packet in the last M arrivals. Based on the information stored in the LHT, the dropper knows the number of arrivals $A_i(M)$ and the number of drops $D_i(M)$ from class i in the last M arrived packets. The complete algorithm is shown in Figure 29.

The PLR(M) dropper requires a packet tag, that is only used internally in the router, recording the LHT entry of the packet. This index is necessary so that the corresponding LHT entry is updated when that packet is dropped. The LHT tags, together with the overhead of maintaining the LHT, are the main additional implementation complexity of PLR(M) compared to PLR(∞). It is noted though, that such temporary tags that are attached to a packet while the packet is stored in the router are not uncommon in modern switch and router designs.

Note that the PLR(M) dropper is different from a PLR(∞)-type of dropper in which the counters are reset to zero after every M arrivals. The PLR(M) dropper attempts to meet the proportional loss rate differentiation model in *every* time window of M arrivals, while the latter dropper attempts to meet the proportional loss rate differentiation in *successive* time windows of M arrivals.

$B(t)$: Set of backlogged classes at time t
$A_i(M)$: Count of packet arrivals in class i in last M arrivals
$D_i(M)$: Count of packet drops from class i in last M arrivals
LHT: Loss History Table (cyclical queue with M entries)
LHT[j].class: Class index of packet in the j'th LHT entry
LHT[j].loss: Drop status of packet in the j'th LHT entry
I: Tail index of LHT

Packet arrival in class i at time t:
// Replace the packet at the LHT tail with the new packet
k=LHT[I].class;
$D_k(M) = D_k(M) -$ *LHT[I].loss*;
$A_k(M) = A_k(M) - 1$;
LHT[I].class=i;
LHT[I].loss =0;
$A_i(M) = A_i(M) + 1$;
Tag g of arrived packet = I;
$I = (I + 1) \bmod M$;

Packet drop from class j at time t:
$j = \arg\min_{i \in B(t) \wedge A_i(M) > 0} \frac{D_i(M)}{A_i(M)\sigma_i}$;
g = tag of dropped packet;
LHT[g].loss=1;
$D_j(M) = D_j(M) + 1$;

Figure 29: Description of the PLR(M) dropper.

The major question in PLR(M) is how large should M be. A larger value of M leads to a closer approximation of the PLD constraints, but as will be shown in the next section, the dropper becomes less adaptive to a varying class load distribution. One constraint is that M should be large enough so that a dropped packet is always one of the last M arrived packets; otherwise, the drop will not be recorded in the LHT. This constraint is met in practice, even for small values of M, because the dropped packets are removed from the queue tails.

Another constraint on M is that it should be large enough so that it is feasible to achieve the specified LDPs in a time window of M packet arrivals. Intuitively, if M is not large enough, the PLR(M) dropper does not 'remember' enough previous arrivals and drops in order to adjust the loss rates in that time window based on the proportional constraints of (4.1). The following result gives a lower bound on the required LHT size M, under stationary conditions for the aggregate loss rate and class load distribution.

Proposition 4.1: If the aggregate loss rate is $\bar{l}_{ag}$ and the load distribution is $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$, it is necessary that the LHT size is more than M_{min} entries in order for PLR(M) to meet the LDPs $\{\sigma_1 = 1, \sigma_2, \sigma_3, \ldots, \sigma_N\}$, where

$$M_{min} = \frac{\sum_{i=1}^{N} \frac{\lambda_i}{\lambda_m} \frac{\sigma_i}{\sigma_m}}{\bar{l}_{ag}} \tag{4.12}$$

and $m = \arg\min_{1 \leq i \leq N}\{\lambda_i \sigma_i\}$.

The proof of this result is given in §7.4.

4.3 Evaluation of PLR droppers

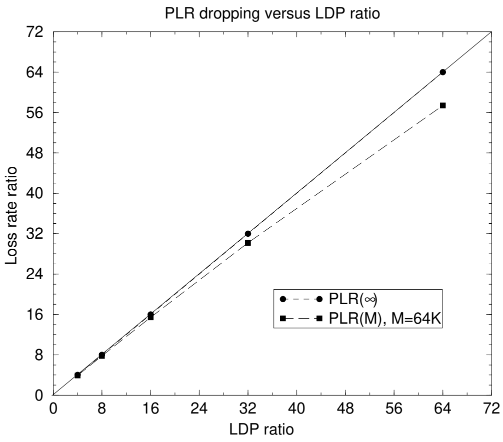
This section presents simulation results for the two proposed droppers PLR(∞) and PLR(M). The simulated model is described in detail in §7.1. We briefly mention here that the backlog controller is Drop-Tail, while the scheduler is HPD with g=0.875. The number of buffers B and the utilization u are adjusted so that the long-term aggregate loss rate does not exceed about

5%; this is a common operating regime in today's Internet. Other parameters, such as the load distribution $\{\lambda_i\}$, or the LDPs and DDPs, are given for each simulation in the corresponding graph or text. A summary of the simulation results follows.

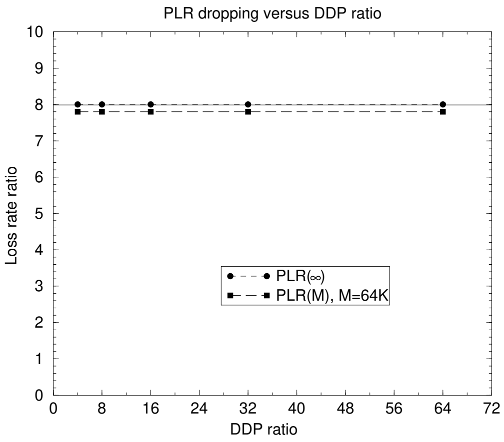
Focusing first on long-term loss differentiation, it is shown that PLR(∞) is able to meet the specified PLD constraints over a wide range of operating conditions and differentiation parameters. When this is not the case, the simulations show that the Strict Priorities (SP) dropper cannot meet the specified PLD constraints either, meaning that the PLD model is infeasible in those operating conditions. PLR(M) is less accurate in meeting the PLD constraints than PLR(∞). The effect of the LHT size M is shown; the deviations of PLR(M) from the PLD model are less than 10% when M is more than 50-100 thousand packets.

We then focus on loss differentiation in shorter timescales, namely in every K packet arrivals of the aggregate traffic stream. When the class load distribution is stationary and K is in the order of 100,000 packets or more, PLR(∞) approximates quite closely the specified LDPs in almost every interval of K packet arrivals. PLR(M) performs similar with PLR(∞), when K and M are in the same range (100,000 packets or more). When the class load distribution is nonstationary (varying), however, PLR(∞) deviates significantly from the specified LDPs, since it cannot adapt quickly to the changing arrival rates. PLR(M), on the other hand, approximates closely the specified LDPs even with a nonstationary load distribution, as long as the timescales in which the class loads vary are larger than M. Finally, we briefly examine a 'two-dimensional' differentiation framework in which some classes encounter lower delays but higher loss rates than other classes, and illustrate some feasibility problems that appear in that framework.

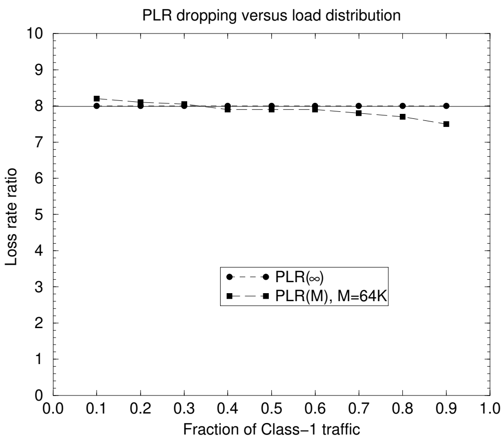
Feasible proportional loss rate differentiation: Figure 30 shows simulation results for the loss rate ratio between two classes with PLR(∞) and PLR(M), in diverse operating conditions. The loss rate ratio shown is $\bar{l}_1/\bar{l}_2$. The aggregate loss rate is about 1.3% in the first three graphs, and it varies between 0.6%-2.7% in the fourth graph. The parameters that we vary in these simulations are the specified LDP ratio σ_1/σ_2, the DDP ratio δ_1/δ_2, the load distribution (determined by λ_1/λ), and the number of buffers B. The LDP ratio is the desired loss rate



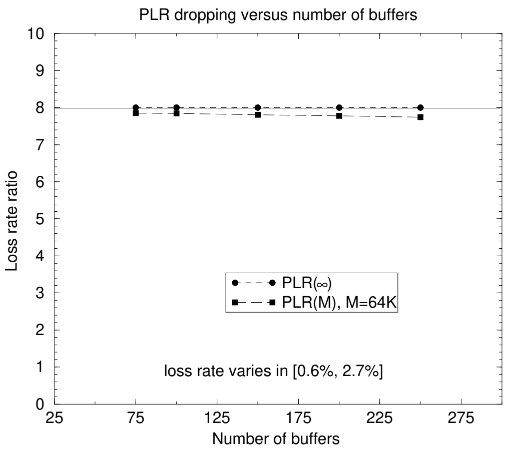
(a) B=150, (λ_1, λ_2)=(70,30), δ_1/δ_2=8



(b) B=150, (λ_1, λ_2)=(70,30), σ_1/σ_2=8



(c) B=150, σ_1/σ_2=8, δ_1/δ_2=8



(d) (λ_1, λ_2)=(70,30), σ_1/σ_2=8, δ_1/δ_2=8

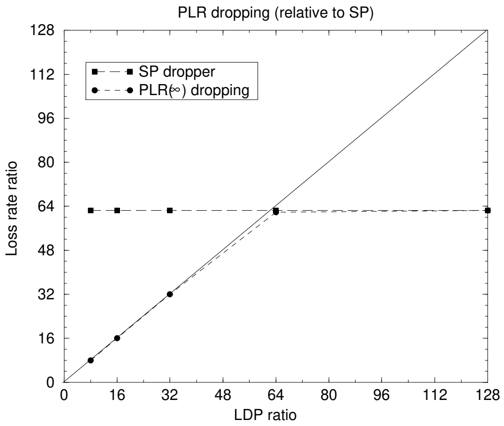
Figure 30: Loss rate differentiation with PLR(∞) and PLR(M) (feasible LDPs).

between the two classes. The DDP ratio affects the service that each class receives, and thus the likelihood that a class is idle at the drop time instants. A higher DDP ratio makes Class-2 more likely to be empty. Similarly, the load distribution and the number of buffers affect the likelihood that a certain class is empty at the time of a packet drop, and thus they can affect the feasibility of the PLR model.

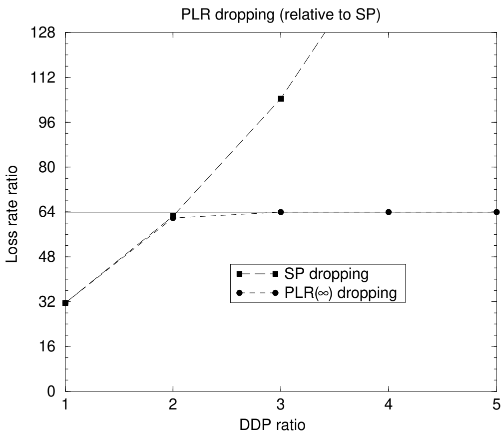
The major observation in these graphs is that PLR(∞) achieves almost exactly the specified LDPs in all the diverse operating conditions shown. In the next paragraph, we show some simulation results in which PLR(∞) deviates from the PLD model, but as it turns out, the model is infeasible in those operating conditions. PLR(M), on the other hand, with an LHT size of M=64K entries, approximates the PLD model with some deviations; these deviations are quantified later in this section.

Infeasible proportional loss rate differentiation: Recall from §4.1 that the LDP ratio σ_1/σ_2 is feasible if and only if the corresponding loss rate ratio $\bar{l}_1/\bar{l}_2$ with the SP dropper is higher than σ_1/σ_2. Figure 31 shows simulation results for the loss rate ratio $\bar{l}_1/\bar{l}_2$ between two classes with PLR(∞) and SP, in operating conditions that reside at the onset of infeasibility.

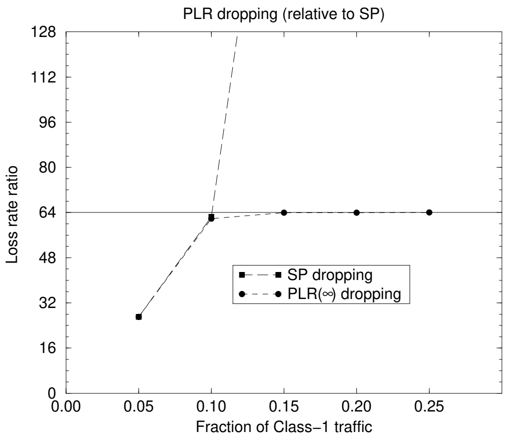
In Figure 31-a, both droppers create a loss rate ratio that is lower than σ_1/σ_2=128. The infeasibility cause is that with this high LDP ratio and weak Class-1 load (λ_1/λ=0.10), there are not enough Class-1 packets to drop for the specified loss rate ratio. In Figure 31-b, both droppers create a loss rate ratio that is lower than σ_1/σ_2=64 when there is no delay differentiation between the two classes (δ_1/δ_2=1). The cause of infeasibility is that when the DDP ratio is too low, Class-2 is not discriminated in favor of Class-1, and so Class-1 does not accumulate enough backlog for the specified loss rate ratio. In Figure 31-c, both droppers create a loss rate ratio that is lower than σ_1/σ_2=64 when the load distribution is λ_1/λ=0.05. The infeasibility cause is that Class-1 is too weakly loaded, and so there are not enough Class-1 packets to drop for the specified loss rate ratio. In Figure 31-d, both droppers create a loss rate ratio that is lower than σ_1/σ_2=64 when B=150. The cause of infeasibility is that the low number of buffers leads to a low average backlog in Class-1, and so Class-1 is often idle when it should encounter a packet



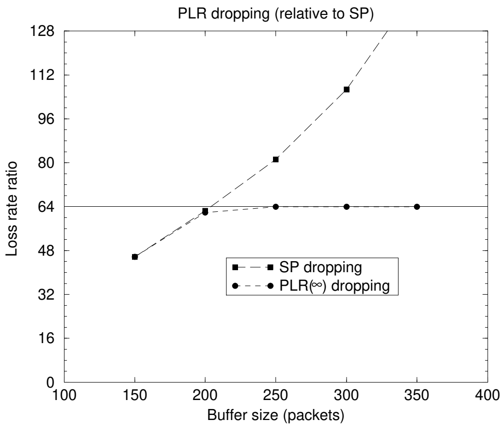
(a) B=200, (λ_1, λ_2)=(10,90), δ_1/δ_2=2



(b) B=200, (λ_1, λ_2)=(10,90), σ_1/σ_2=64

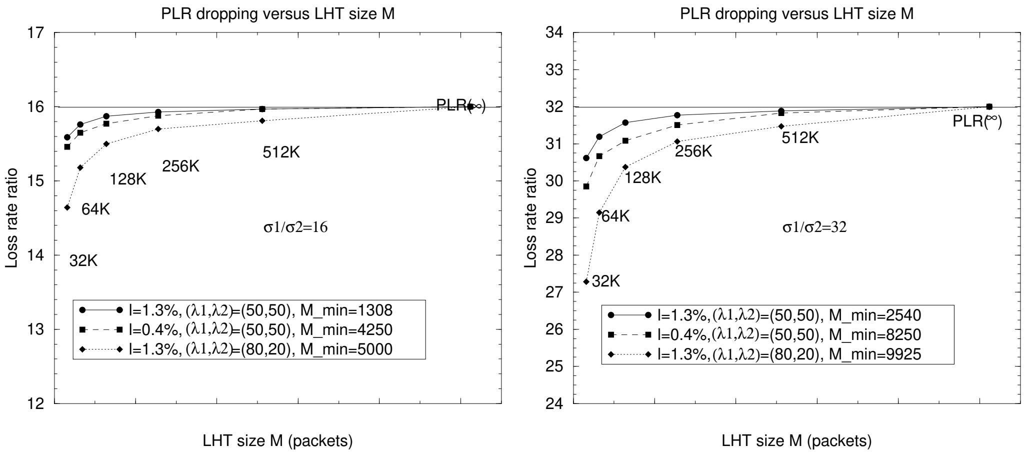


(c) B=200, σ_1/σ_2=64, δ_1/δ_2=2



(d) (λ_1, λ_2)=(10,90), σ_1/σ_2=64, δ_1/δ_2=2

Figure 31: Proportional loss rate differentiation at the onset of infeasibility.



(a) σ_1/σ_2=16, δ_1/δ_2=8 (b) σ_1/σ_2=32, δ_1/δ_2=8

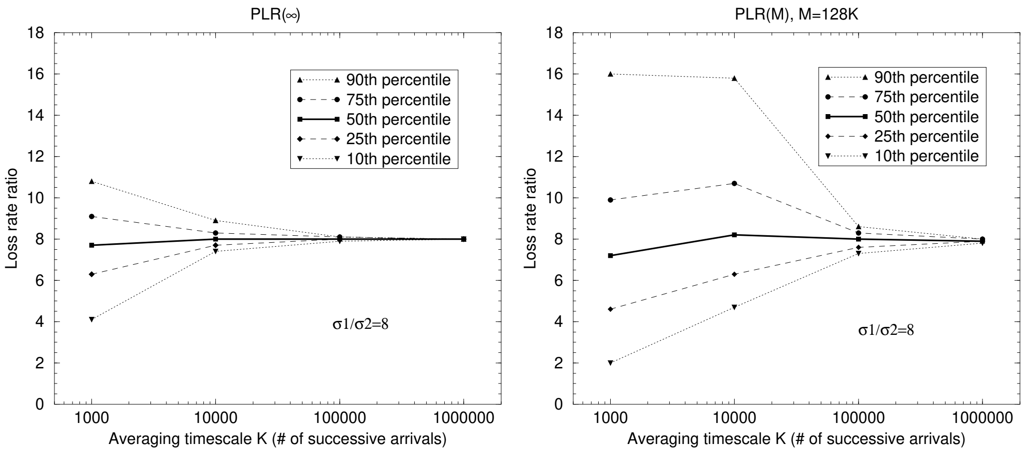
Figure 32: The effect of the LHT size in PLR(M).

drop.

In all cases when the PLD model is infeasible, the PLR(∞) and SP droppers lead to the same loss rate ratio. This leads us to the conjecture that *PLR(∞) is the optimal dropper in the context of the PLD model, since it can meet the specified LDPs when the PLR model is feasible.* This conjecture is similar to the optimality of the PAD scheduler in the context of the PDD model (§3.2).

The effect of the LHT size M in PLR(M): Figure 32 shows the loss rate ratio between two classes for different values of the LHT size M. The resulting loss rate ratio with PLR(∞) is also shown, as the asymptotic case $M \to \infty$, in order to compare the two droppers.

The first observation is that as we increase M, the deviations of PLR(M) from the PLD model decrease, because PLR(M) maintains more accurate loss rate estimates. For the operating conditions in these graphs, if the relative error from the PLD constraints is to be less than 10%, the LHT should have at least 64K entries. The second observation is that the required LHT size for meeting the PLD model with a certain accuracy increases as the LDP ratio σ_1/σ_2



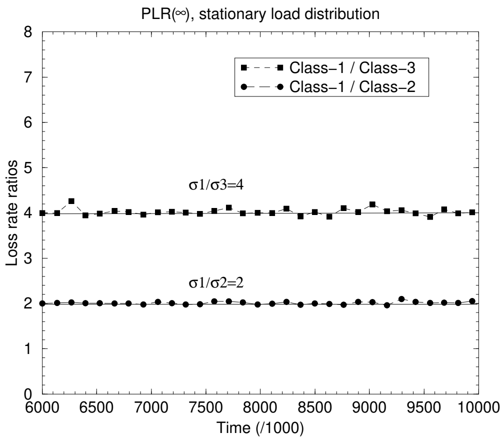
(a) PLR(∞)

(b) PLR(M), M=128000

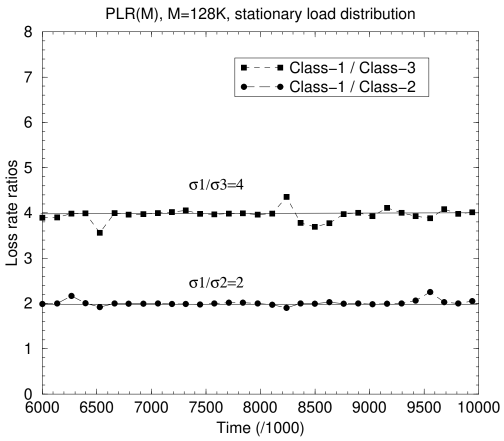
Figure 33: Percentiles of the loss rate ratios with PLR(∞) and PLR(M) as a function of the averaging timescale K.

increases, as the aggregate loss rate $\bar{l}_{ag}$ decreases, or as the load ratio λ_1/λ_2 increases. These dependencies of the required LHT size M are also predicted from (4.12). Third, the lower bound on M given in (4.12) is quite smaller than what is needed in practice for a close approximation of the PLD model. The reason is that the proof of Proposition 4.1 assumes that the class load distribution and the aggregate loss rate *in any time window* of M packet arrivals follow their long-term average values. In practice, though, the class load distribution and the aggregate loss rate vary significantly around their average values, due to the burstiness in the arrival and packet loss random processes.

Proportional loss rate differentiation in short timescales: Figure 33 examines the behavior of PLR(∞) and PLR(M) in short timescales. Specifically, the graphs in Figure 33 show *five percentiles of the distribution of loss rate ratios between two classes, when the loss rates are measured in time windows of K successive packet arrivals.* For more details about this type of simulation experiments, see also Figure 22 and the corresponding explanations given in §3.4. The number of buffers in these simulations is B=80, and the aggregate loss rate is $\bar{l}$=3.7%. The DDP



(a) PLR(∞),



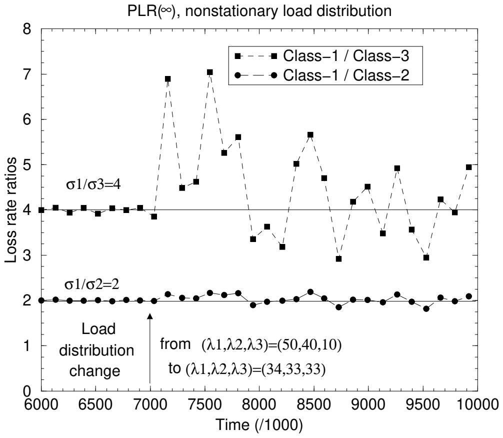
(b) PLR(M), M=128K

Figure 34: Loss rate differentiation with PLR(∞) and PLR(M) in a stationary class load distribution.

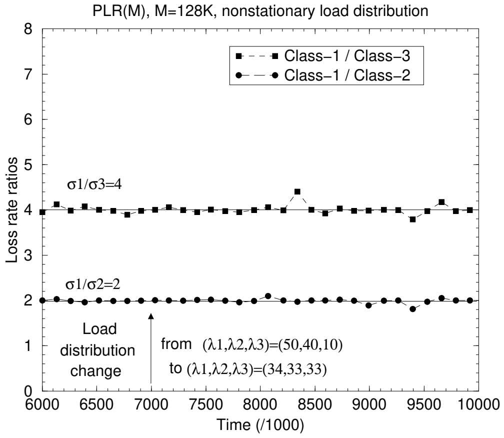
ratio is δ_1/δ_2=8, the LDP ratio is σ_1/σ_2=8, and the load distribution is (λ_1, λ_2)=(70,30).

As shown, the resulting distributions of loss rate ratios with PLR(∞) are quite narrow and centered around the specified LDP, when K is 100,000 packets or more. In these timescales, the desired loss rate ratio is achieved in almost all intervals of K packet arrivals. In short timescales, say every K=1,000 or 10,000 packets, the resulting distribution of loss rate ratios becomes significantly more spread, but the loss differentiation is still predictable (i.e., the higher class has a lower loss rate). The corresponding distributions in PLR(M) have a wider range than in PLR(∞), especially when $K < M$. For $K > M$, PLR(M) also provides narrow loss rate ratio distributions, centered around the specified LDPs. The PLR(∞) dropper, though, provides more accurate and narrow distributions than PLR(M) in all timescales. As will be shown in the next paragraph, the major advantage of PLR(M) over PLR(∞) is when the load distribution between classes is nonstationary.

Stationary and nonstationary class load distributions: Figure 34 shows a sample path of the loss rate ratios between three classes with PLR(∞) and PLR(M). The loss rates are measured



(a) PLR(∞)



(b) PLR(M), M=128K

Figure 35: Loss rate differentiation with PLR(∞) and PLR(M) in a nonstationary class load distribution.

in every K=100,000 packet arrivals of the aggregate traffic stream. In these graphs, the average arrival rate in each class remains constant, i.e., *the class load distribution is stationary.* The major observation is that both droppers approximate quite closely the specified LDPs (σ_i/σ_{i+1}=2) in almost all time intervals of K packets. A more thorough inspection of these, as well as other similar sample paths, shows that PLR(∞) meets the PLD constraints more accurately than PLR(M); this observation also agrees with the results shown in Figure 33.

Figure 35 also shows a sample path of the loss rate ratios between three classes, but this time the class load distribution is nonstationary. Specifically, at the time instant 7000, the average class load distribution is modified from $(\lambda_1, \lambda_2, \lambda_3)$=(50,40,10) to $(\lambda_1, \lambda_2, \lambda_3)$=(34,33,33). Such a change can occur in practice if some users switch between classes.

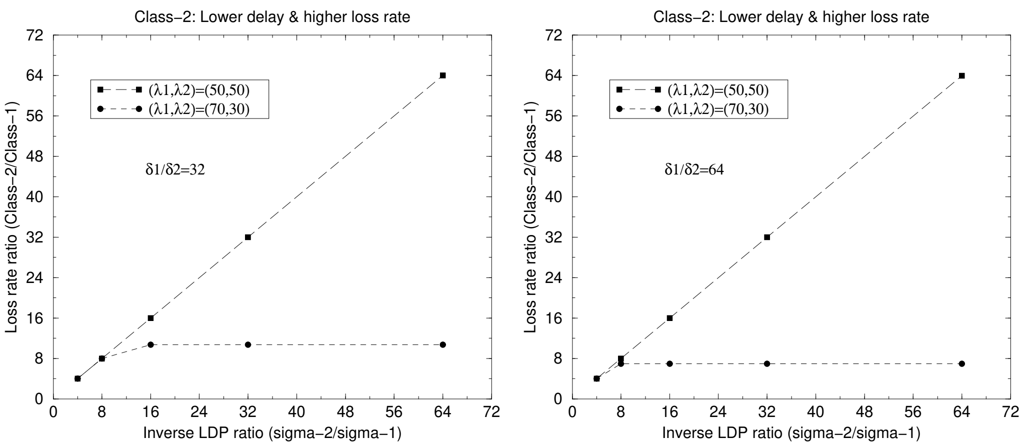
The PLR(M) dropper is unaffected by the load distribution change, since it only 'remembers' the arrivals from each class in the last M=128,000 packets. So, quickly after the load distribution change, it adapts to the new class arrival rates and delivers the desired loss rate ratios. PLR(M) can deal effectively with continuous load distribution changes, as long as the timescales in which they occur are larger than M.

The PLR(∞) dropper, on the other hand, is affected negatively by the load distribution change, and the loss rate ratio, especially between classes 1 and 3, diverges significantly from the specified LDPs. The reason is that PLR(∞) attempts to provide a certain loss rate ratio over the entire previous history of packet arrivals, which does not reflect the new arrival rates in each class. So, even though it manages to achieve the specified LDPs over the long run, it fails to provide the desired loss rate ratio in short timescales after the load distribution change. The duration of the erratic behavior depends on the size of the arrival counters and the time until their next counter overflow.

This experiment shows that when the class load distribution is strongly nonstationary, perhaps due to users that switch between classes in order to dynamically select an acceptable class, the preferred dropper should be PLR(M). Even though PLR(M) is not optimal in meeting the PLD constraints when the LDPs are feasible, it is adaptive to varying class load distributions while PLR(∞) is not. If the PLR(M) dropper is considered too hard to implement, the PLR(∞) can be used instead, as long as the arrival and drop counters are reset to zero in relatively short timescales (e.g., every few millions of packets).

Two-dimensional delay and loss rate differentiation: As discussed in §2.1, the context of this dissertation is a 'one-dimensional' differentiation framework (such as the IETF Class Selectors) in which *higher classes offer better performance in terms of both queueing delays and packet losses*. The proposed PDD and PLD models, as well as the corresponding schedulers and droppers, however, can also be applied in a 'two-dimensional' differentiation framework in which some classes offer lower delays but higher loss rates than other classes. Even though such a two-dimensional framework is out of the scope of this thesis, we show here an example of this approach, illustrating also a feasibility issue that emerges.

In the case of two classes, suppose that Class-2 offers a lower average delay but a higher loss rate than Class-1. The PDD model requires that the average delay ratio is $\bar{d}_2/\bar{d}_1 = \delta_2 < 1$, while the PLD model requires that the loss rate ratio is $\bar{l}_2/\bar{l}_1 = \sigma_2 > 1$. Figure 36 shows the resulting loss rate ratio for two load distributions and two target delay ratios. The scheduler



(a) δ_1/δ_2=32 (b) δ_1/δ_2=64

Figure 36: Example of two-dimensional differentiation.

is HPD with g=0.875, and the dropper is PLR(∞). When the load distribution is uniform, (λ_1, λ_2)=(50,50), the specified loss rate ratio is achieved even when we require that Class-2 has a 64 times lower average delay and a 64 times higher loss rate than Class-1. When the load in Class-2 is slightly less though, namely (λ_1, λ_2)=(70,30), the loss rate ratio is $\bar{l}_2/\bar{l}_1 < 11$ when δ_1/δ_2=32, and $\bar{l}_2/\bar{l}_1 < 7$ when δ_1/δ_2=64. The reason that the loss differentiation is not as specified is that the delay differentiation in favor of Class-2 increases the likelihood that Class-2 is idle. Consequently, it becomes hard to introduce a high loss rate in Class-2, relative to the loss rate in Class-1. This effect becomes stronger as the Class-2 load decreases.

In the 'one-dimensional' differentiation framework, on the other hand, the fact that classes with higher delays also have a higher loss rate reduces the space of infeasible operating conditions. The reason is that classes with higher delays are more likely to be backlogged, and so, it is simpler to also provide them with a higher loss rate. In summary, even though a two-dimensional differentiation framework appears to be more flexible than the one-dimensional framework that we consider, it faces some harder feasibility problems that require further investigation.

4.4 TCP throughput with proportional delay and loss differentiation

Most of the Internet traffic today uses TCP as the underlying transport protocol [112]. Consequently, it is interesting to examine the impact of per-hop proportional delay and loss rate differentiation on the TCP throughput. In this section, we attempt a first-order investigation of this issue for the case of *bulk-transfer TCP connections*, which spend most of their lifetime in the congestion avoidance phase of the TCP algorithm [65]. During that phase, TCP increases the sending window by a fixed amount of bytes every time it receives an acknowledgement [50]. When a packet loss is detected, the sending window is halved in order to reduce the congestion in the network.

The throughput of a bulk-transfer TCP connection in the congestion avoidance phase can be approximated by

$$R = \frac{0.93\,M}{T\,\sqrt{l}} \tag{4.13}$$

where M is the connection's Maximum Segment Size, l is the average loss rate that the connection encounters, and T is the round-trip delay in the connection's path, *including* queueing delays [65]. We assume that the connection encounters the average delay and loss rate of the aggregate traffic in each link along its path. The round-trip delay when the connection is in class i consists of a fixed propagation delay τ and an average queueing delay $\bar{d}_i$, i.e., $T_i = \tau + \bar{d}_i$. Also, let $\bar{l}_i$ be the end-to-end loss rate in class i. If $\bar{d}_i^k$ and $\bar{l}_i^k$ are the average queueing delay and the loss rate, respectively, in class i at the k'th hop of the connection's path, we have that $\bar{d}_i = \sum_k \bar{d}_i^k$, and $\bar{l}_i = 1 - \prod_k (1 - \bar{l}_i^k)$. If $\sum_k \bar{l}_i^k$ is less than 1% or so, then $\bar{l}_i \approx \sum_k \bar{l}_i^k$.

Suppose now that every hop along the connection's path offers proportional delay and loss rate differentiation with the same DDPs and LDPs. Then, $\bar{d}_j^k = \delta \bar{d}_i^k$ and $\bar{l}_j^k = \sigma \bar{l}_i^k$, where $\delta = \delta_j / \delta_i$ and $\sigma = \sigma_j / \sigma_i$; if $i > j$, $\delta > 1$ and $\sigma > 1$. So, the ratio of the throughput that the

TCP connection would get in class i over class j is

$$\frac{R_i}{R_j} = \sqrt{\sigma}\,\frac{\delta + \tau/\bar{d}_i}{1 + \tau/\bar{d}_i} \tag{4.14}$$

Note that the loss rate differentiation affects the TCP throughput ratio between classes in a square-root law. The effect of the delay differentiation, on the other hand, depends on the relative magnitude of the queueing delays compared to the propagation delay of the path. If the propagation delay is small compared to the queueing delay ($\frac{\tau}{\bar{d}_i} \ll 1 < \delta$), the delay differentiation affects the TCP throughput ratio linearly. When the connection's propagation delay is large compared to the queueing delays, the impact of the delay differentiation can be minor.

As an example, suppose that δ=σ=2, and that the the propagation delay is equal to the class i average queueing delay ($\tau \approx \bar{d}_i$). The TCP throughput in class i will be approximately 2.1 times higher than in class j, and the contribution of the loss rate differentiation on the throughput ratio will be about the same with the contribution of the delay differentiation. If the propagation delay is three times higher than the class i queueing delay though (say in an cross-continental connection), the TCP throughput in class i will be about 1.8 times higher than in class j, and the contribution of the loss rate differentiation on the TCP throughput ratio will be about 80%.

4.5 Related work on loss differentiation

Prioritized buffer management and dropping algorithms have received significant attention in the literature, especially in the ATM context. The reason is that the ATM cell header includes a 'Cell Loss Priority' bit which creates two levels of loss differentiation [2]. Detailed reviews of the early works in this field can be found in [63, 55]. Several of those efforts assumed that the pushout procedure (removing a backlogged packet) is hard to implement, and so they developed buffer management schemes that can offer loss rate differentiation dropping only arriving packets. Such schemes include the Complete Buffer Partitioning (CBP) [63], Partial Buffer Sharing (PBS) [55], and Dynamic Threshold [23] algorithms. As mentioned in §4.1, it has been shown recently with actual implementations of switches and routers that pushout can be supported in high-speed links when packets are removed only from the head or the tail of the class queues [19, 108].

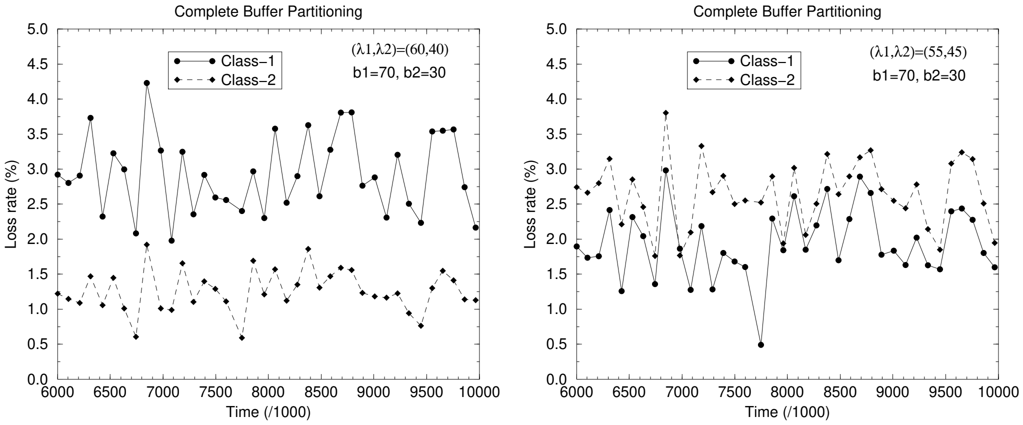
In the following, we first review some of the early buffer management and dropping schemes, illustrating their limitations in the context of relative loss differentiation. Then, we examine RIO and multiclass-RED, which have received significant attention recently, especially because of the active buffer management features of RED [43]. Finally, we discuss a few other droppers in the context of proportional loss differentiation, which are similar to the PLR droppers that we propose here.

4.5.1 Early buffer management schemes

Pushout or Strict Priority (SP): In this simple dropper, the packet removed is always from the lowest backlogged class [63, 55]. We call it *Strict Priority (SP) dropper* to show that it corresponds to the SP scheduler in the delay differentiation context. As shown in §4.1, the SP dropper is directly related to the feasibility of the PLD model. Although SP provides predictable relative differentiation, it does not provide any means of controlling the loss rates between classes, i.e., it is not a controllable differentiation mechanism. Additionally, SP can cause starvation in the lower classes in the form of excessive packet drops.

Complete Buffer Partitioning (CBP): Another simple buffer management scheme is to statically partition the buffer space between classes, i.e., to allocate b_i packet buffers to each class i (with $\sum_i b_i = B$). From Little's law, the ratio of aggregate backlogs between two classes i and j is $\bar{q}_i/\bar{q}_j = \bar{d}_i/\bar{d}_j \, \lambda_i^a/\lambda_j^a$, where λ_i^a is the input (accepted) rate in class i and $\bar{d}_i$ is the average queueing delay of the serviced packets in class i. This relationship shows that the buffer partitions must be selected based on the class load distribution and the delay differentiation provided by the scheduler. CBP increases the aggregate loss rate, compared to LIFO or SP for instance, because packets drops occur even when there are available buffers.

Another problem with CBP is that the selection of the buffer partitions, in order to



(a) Well provisioned buffer partitions

(b) Poorly provisioned buffer partitions

Figure 37: Loss rates with a two-class CBP dropper.

achieve a certain loss rate differentiation, is quite sensitive to variations in the load distribution. This is shown with two sample paths in Figure 37. The loss rates are measured in every K=100,000 packet arrivals. The HPD scheduler with g=0.875 and δ_1/δ_2=2 provides proportional delay differentiation between the two classes. The buffer partitions in Figure 37-a are selected so that, when the load distribution is (λ_1, λ_2)=(60,40), the loss rate is about 3% in Class-1 and about 1% in Class-2. A slight change in the load distribution, however, from (λ_1, λ_2)=(60,40) to (λ_1, λ_2)=(55,45), makes the selected buffer partitions totally inappropriate, since they cause a larger loss rate in Class-2 than in Class-1 (Figure 37-b). This is an instance of *unpredictable* loss differentiation, which is common for static resource allocation mechanisms such as CBP.

Partial Buffer Sharing (PBS): In the PBS scheme [55], a packet from class i is accepted in the corresponding queue if the *aggregate backlog* is less than a threshold b_i (with $b_i < b_j$ for $i < j$). So, higher classes are given more buffer space than lower classes. The PBS scheme does not require pushout, and offers the operator with a set of adjusting parameters. A problem that we observed experimenting with PBS, is that the buffer thresholds cannot adjust the loss rate differentiation between classes with a fine granularity. In other words, a slight change in the thresholds b_i can lead to a dramatic change in the obtained loss rates.

A general problem with both CBP and PBS is that they are *static resource allocation mechanisms*, and so they do not dynamically adjust the partitioning of buffer resources among classes based on the actual class loads and on the desired loss rate differentiation. On the contrary, the PLR droppers of §4.2 dynamically readjust the number of buffers that each class occupies based on the deviation of the measured loss rate differentiation from the target loss rate differentiation.

4.5.2 Multiclass Random Early Detection (RED) schemes

The mRED [93] and RIO [24] mechanisms are priority extensions of the RED active buffer manager [43]. RIO was originally proposed for two classes, while mRED (or *Weighted RED*) is proposed for N classes. Both RIO and mRED are similar to PBS, since each class is given a buffer threshold b_i, and arriving packets in that class are dropped when the aggregate backlog is more than b_i. The two algorithms differ from PBS in certain buffer management features that they borrow from RED, such as probabilistic dropping, gradual increase of the loss probability as the backlog increases, and backlog estimation through run-time averages. These features are not related to the loss differentiation aspect though, that is the same as in PBS. Consequently, both RIO and mRED have the same drawbacks as PBS in achieving a controllable and predictable relative loss differentiation. These drawbacks, together with some other problems of mRED, are studied in [12].

4.5.3 Other proportional loss rate droppers

A dropping algorithm that is essentially the same with PLR(∞) has been also discussed in [119, 19], but from a different perspective than ours. Specifically, [19] designed a *self-calibrating pushout* ATM dropper in which 'the number of dropped cells is proportional to the loss weight of different loss priorities', which is similar to the PLD model. [19] proposed a VLSI implementation that does not require floating-point operations for this dropper, *when the arrival rate in each class is policed and known.* We cannot use that optimization, because the class loads are not known a priori in the context of relative differentiation.

[119] studied a dropper that is essentially the same as PLR(∞), in a more abstract manner. Specifically, [119] showed that the PLR(∞) dropper, jointly with a FCFS scheduler, is *optimal* because it requires *the minimum capacity for a certain loss rate in each class.* This result is related to the *class provisioning* problem, in which the capacity of a link is adjusted based on the expected class loads, so that each class provides a given *absolute* performance level. We study this problem in §5.4 in the context of average delay provisioning. The result of [119] is interesting, since it implies that the proportional differentiation model is not only attractive for relative differentiation, but it may also be optimal when it is considered in the context of class provisioning.

After our original work on the PLR droppers [34], other researchers have proposed variations and improvements. The *Joint Buffer management and Scheduling (JoBS)* algorithm [61], which is also discussed in §3.5, can offer proportional loss rate differentiation between classes. An interesting feature of JoBS is that it unifies the scheduling and dropping decisions in a single step. The loss rate metric that [61] uses is the fraction of lost traffic since the beginning of the current busy period. JoBS solves an optimization problem, taking into account both absolute and relative differentiation constraints, in order to decide which packet to transmit or drop next. Because the solution of this optimization problem would make the algorithm prohibitively slow, the authors propose a more efficient heuristic implementation.

Recently, [11] proposed a variation of PLR(M) called *Average Drop Distance (ADD)* dropper. The main advantage of ADD over PLR(M) is that it avoids the division that is required for calculating the fraction of dropped packets (i.e., the D_i/A_i terms). ADD is based on a run-time estimator for the average drop distance, which is the average number of arrived packets between two successive drops in a class. The estimation of this measure does not require a division, and it can be efficiently implemented in software or hardware. The authors of [11] implemented ADD in the FreeBSD kernel of a Pentium II system, and measured that selecting the drop-target class takes 113 cycles and updating of the loss rate estimates takes 240 cycles for 8 classes.

4.6 Summary and extensions

Our objective in this chapter was to apply the general model of proportional differentiation in the context of packet losses. Packet losses have a significant impact on many applications, and especially in the throughput of TCP traffic.

In the first part of the chapter, we proposed and studied the Proportional Loss Differentiation (PLD) model. According to PLD, the loss rate ratios between classes are fixed, based on certain parameters (LDPs) that the network operator specifies. The PLD model provides a *differentiation invariant* that does not depend on the aggregate load or on the class load distribution. The PLD model is similar in several aspects with the PDD model of Chapter 3. It turns out that the PLD model cannot be feasible in all operating conditions either. The feasibility conditions can be checked with the Strict Priorities dropper, using a similar procedure as in the case of the PDD model.

In the second part of the chapter, we focused on the design of two Proportional Loss Rate (PLR) droppers for the PLD model. PLR(∞) appears to be the optimal dropper for the PLD model, because it can achieve the PLD constraints when the model is feasible. In addition, it approximates the PLD model quite well in short timescales when the loss rates are measured in every 100,000 packets or more. PLR(M), on the other hand, is not as accurate as PLR(∞) in meeting the PLD model. PLR(M) also approximates the PLD model closely in short timescales when the loss rates are measured in every 100,000 packets or more, given that M is also in the same range. The PLR(∞) dropper is simpler to implement than PLR(M), because the latter requires a cyclical queue and a tag for each packet. PLR(M) should be preferred over PLR(∞), however, when the class load distribution varies with time. In such conditions, PLR(∞) can deviate for large time intervals from the PLD constraints. PLR(M), instead, adapts quickly to the new load conditions.

Our focus throughout this study was more on proposing the model of proportional loss differentiation and on the design of dropping algorithms, rather than on the analysis of these algorithms. Consequently, as it was also the case in Chapter 3, our conclusions in several points are based on empirical results and simulations. A number of interesting issues that deserve further investigation follow.

- The feasibility conditions for the PLD model (4.8) were presented here as a conjecture, based on the corresponding Regnier conditions (3.7) for the feasibility of the PDD model. Even though our simulation results provide evidence that the conditions of (4.8) are correct, a mathematical treatment of the feasibility problem in the context of losses is still needed.

- In most of the chapter, including the evaluation of the two PLR droppers (§4.3), we considered only the Drop-Tail backlog controller. It is interesting to examine what changes when more sophisticated backlog controllers are used, both for the PLD model (conservation law, feasibility, etc), and for the behavior of the two PLR droppers. Certain active buffer management schemes, including RED, drop packets randomly, even when there are available packet buffers. For such backlog controllers the conservation law does not hold, because they are not 'work-conserving' in terms of buffer utilization.

- Our claim that PLR(∞) is optimal in stationary load conditions, in the sense that it can meet the PLD model when the LDPs are feasible, is supported by simulations, but it lacks a mathematical proof.

- Our study of TCP throughput differentiation under both delay and loss differentiation is limited to bulk-transfer connections in the congestion avoidance phase. An interesting problem is to examine the impact of proportional differentiation on shorter TCP connections, which spend most of their lifetime in the 'slow-start' TCP phase.

Chapter 5

Dynamic Class Selection and Class Provisioning

A central premise in the relative differentiation architecture is that *users with an absolute QoS requirement can dynamically search for a class which provides the desired QoS level.* The first part of this chapter investigates this *Dynamic Class Selection (DCS)* framework, in the context of proportional delay differentiation. For a single link model, we give an algorithm that checks whether it is feasible to satisfy all users, and if this is the case, computes the minimum acceptable class selection for each user. Users converge in a distributed manner to this minimum acceptable class, if the DCS equilibrium is unique. However, other suboptimal DCS equilibria may also exist. Simulations of an end-to-end DCS algorithm provide further insight in the dynamic behavior of DCS, show the relation between DCS and the network DDPs, and demonstrate how to control the trade-off between a flow's performance and cost using DCS. In the second part of this chapter, we study the problem of *class provisioning*. In this case, the network provider has some knowledge about the traffic types that a link carries (average rates and delay requirements). The objective is to *compute the minimum required link capacity and the appropriate Delay Differentiation Parameters (DDPs)* for these traffic types. We give a methodology that first determines the optimal mapping from traffic types to service classes, and then computes the minimum capacity and the required DDPs. The class provisioning methodology is demonstrated with examples. The chapter closes with a review of related works on class selection and provisioning.

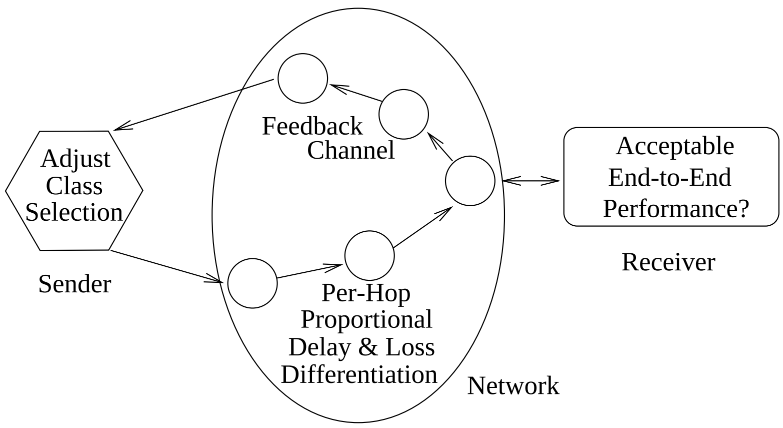


Figure 38: Dynamic Class Selection (DCS) model.

5.1 Dynamic Class Selection (DCS) and proportional differentiation

The mapping of traffic to service classes (or *traffic classification*) can be performed at the ingress routers or at the network end-points. By 'end-points' we mean individual users, the operating system at the end-hosts, or the applications[1]. A router-based classification is preferable when the users cannot participate in the process, or when the network provider wants to map specific traffic types to different classes. A user-based classification, on the other hand, is preferable when users have a specific QoS requirement from the network, such as a maximum Round-Trip Delay (RTD) or loss rate. In that case, the users can dynamically search for a class that provides them with the required QoS level. We refer to this user-based classification approach as *Dynamic Class Selection (DCS)*.

The DCS model is illustrated in Figure 38. The network offers per-hop relative delay and loss differentiation. A user that does not have absolute QoS requirements, can choose a class based on the maximum tariff that she is willing to pay. That class will provide the best possible QoS, given her cost constraints. A user that has absolute QoS requirements, on the other hand, has to dynamically choose a class in which the observed QoS is acceptable. If the user is also

[1] In the following, these end-points are referred to as *users*.

interested in minimizing the cost of the session, she would like to choose the least expensive, or minimum, class that is acceptable.

In the model of Figure 38, it is the sender that makes the class selections, and the receiver that monitors the end-to-end QoS. The receiver notifies the sender about the observed QoS through a *feedback channel*, such as the one provided by the Real-Time Control Protocol (RTCP) [96]. Based on this feedback, the sender decides whether to stay in the same class, or switch to the higher or lower class. There are several variations of this basic DCS model. For instance, a user can perform DCS aiming for a minimum TCP throughput, a maximum loss rate, or a maximum latency of Web transfers. Also, the magnitude of the class transitions can depend on the deviation between the desired and the measured performance. A complete DCS algorithm is described in §5.3.

In the DCS model, users may experience transient performance degradations as they search for an acceptable class. Such transient performance degradations would be unacceptable for *unelastic* applications that require strict performance guarantees. We consider such applications outside the scope of relative differentiation. If the application is *adaptive*, however, the transient performance degradations can often be masked using a variety of techniques. For instance, playback-adaptation techniques can mask excessive delays in voice applications [87], while limited losses can be dealt with using retransmissions [91], loss concealment [36], or Forward-Error-Correction (FEC) codes [83]. It is noted that most Internet applications today are adaptive.

The proportional differentiation model of §2.4 provides an appropriate per-hop behavior for DCS due to the following reasons. First, the proportional differentiation model is *predictable* (§2.3), meaning that *higher classes provide better performance than lower classes, independent of the aggregate load or the class load distribution.* This is particularly important in the context of DCS, because the class load distribution can be strongly nonstationary as users move from one class to another. Relative differentiation mechanisms that are based on static resource partitioning between classes are not predictable when the class load distribution varies. This was shown in §2.3 for the case of Weighted-Fair-Queueing (WFQ) scheduling, and in §4.5.1 for the case of Complete-Buffer-Partitioning (CBQ) dropping. Periodic adjustments of the resource partitioning, on the other hand, may cause instability effects (discussed in §5.5).

The second reason is that the proportional differentiation model is *controllable* (§2.2), meaning that *the network provider can adjust the performance spacing between classes based on certain class differentiation parameters.* These differentiation parameters allow the network provider to provision the performance spacing between classes based on the corresponding performance spacing in the requirements of different traffic types or user groups. As shown in §5.3, this type of 'differentiation provisioning' can increase the number of DCS users that can find an acceptable class, leading to a more efficient network operation. Other relative differentiation mechanisms, such as the *strict prioritization* model (§2.2), cannot adjust the performance spacing between classes, and so the network provider has no means to provision the class differentiation.

Finally, the proportional differentiation model can be applied in both queueing delays (§3.1) and packet losses (§4.1), and so it can support a number of DCS performance requirements, such as a maximum end-to-end or round-trip delay, a maximum loss rate, or combinations of these metrics, such as a minimum TCP throughput (§4.4).

5.2 A single link DCS model

In this section, we study an analytical model of DCS in the context of a single link. The link provides proportional delay differentiation, while the users search for the minimum class that provides them with an average queueing delay that is less than a certain threshold.

5.2.1 Link and user models

Consider a network link $\mathcal{L}$. The offered rate in $\mathcal{L}$ is λ, the capacity is C, and the utilization is $u = \lambda/C < 1$. We assume that the link has adequate buffers to avoid any packet losses. $\mathcal{L}$ offers N classes of service, which are relatively differentiated based on the *Proportional Delay Differentiation (PDD) model* of §3.1. Specifically, if $\bar{d}_i$ is the average queueing delay in class i, the PDD model requires that

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j} \qquad 1 \leq i, j \leq N \tag{5.1}$$

where $\delta_1 = 1 > \delta_2 > \ldots > \delta_N > 0$ are the *Delay Differentiation Parameters (DDPs)*. Note that according to the PDD model, higher classes have lower average delays, *independent of the class loads.*

As shown in §3.1.1, when the class load distribution $\{\lambda_n\}$ is given, the average queueing delay in class i under the PDD constraints is

$$\bar{d}_i = \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda_n} \tag{5.2}$$

where $\bar{q}_{ag} = \sum_{n=1}^{N} \lambda_n \bar{d}_n$ is the *average aggregate backlog* in $\mathcal{L}$. From the conservation law (3.3), the aggregate backlog $\bar{q}_{ag}$ is independent of the class load distribution or the scheduling algorithm, when the latter is work-conserving and indifferent to packet sizes. $\bar{q}_{ag}$ only depends on the link utilization and on the statistical properties (burstiness) of the traffic.

Suppose now that a population of users $\mathcal{U} = \{1, \ldots, U\}$ create the traffic of link $\mathcal{L}$. Each user j generates a stationary flow with an average rate r_j, and can tolerate a maximum average queueing delay ϕ_j in $\mathcal{L}$. The total offered rate to $\mathcal{L}$ is $\lambda = \sum_j^U r_j$. Without loss of generality, $\mathcal{U}$ is ordered so that $\phi_1 \geq \phi_2 \geq \ldots \geq \phi_U > 0$.

Suppose that each user j in $\mathcal{U}$ selects a class $c_j \in \{1, \ldots N\}$. The corresponding *Class Selection Vector (CSV)* $\mathbf{c}$ is defined as $\mathbf{c} = (c_1, c_2, \ldots, c_U)$. Given a CSV $\mathbf{c}$, the offered rate in each class i is

$$\lambda_i(\mathbf{c}) = \sum_{j: c_j = i}^{U} r_j \tag{5.3}$$

From (5.2) and (5.3), we see that the CSV $\mathbf{c}$ determines the average delay $\bar{d}_i(\mathbf{c})$ in each class i, when the DDPs and the average aggregate backlog are given. Note that the average aggregate backlog does not depend on the CSV $\mathbf{c}$.

The objective of each user is to select the minimum class that meets the user's delay requirement. A user j is said to be *satisfied with* $\mathbf{c}$ if $\bar{d}_{c_j}(\mathbf{c}) \leq \phi_j$; otherwise, the user is said to be *unsatisfied with* $\mathbf{c}$. The CSV $\mathbf{c}$ is called *acceptable*, if all users in $\mathcal{U}$ are satisfied with $\mathbf{c}$. When $\mathbf{c}$ is acceptable, we also say that class c_j is acceptable for user j. CSVs can be compared in the following sense: $\mathbf{c}' \geq \mathbf{c}$ when $c'_j \geq c_j$ for all $j = 1 \ldots U$.

Since U and N are finite, we can examine all possible CSVs to see whether they are acceptable. The set of acceptable CSVs is denoted by C_A. Note that C_A may be the null set; this occurs when the user requirements cannot be met with the given link capacity and DDPs. If $C_A \neq \emptyset$ and $\mathbf{c} \in C_A$, we say that the user population $\mathcal{U}$ is *satisfied with* $\mathbf{c}$, or simply *satisfiable*. Equivalently, the link $\mathcal{L}$ is said to be *well-provisioned* for $\mathcal{U}$. If $C_A = \emptyset$, the user population is said to be *unsatisfiable*, and the link is said to be *under-provisioned* for $\mathcal{U}$.

The following result expresses an important property of the PDD model. When one ore more users move to a higher class, the delay of all classes increases. The class delays decrease, on the other hand, when one or more users move to a lower class. The proofs of all lemmas in this section can be found in §7.5.

Lemma 1: If $\mathbf{c}$ and $\mathbf{c}'$ are two different CSVs such that $\mathbf{c} \geq \mathbf{c}'$, the average delay in each class i is $\bar{d}_i(\mathbf{c}') \leq \bar{d}_i(\mathbf{c})$.

A second important property of the PDD model is that when a user moves from one class to another, while the rest of the users stay in the same class, the user observes a consistent class ordering, i.e., the higher class provides a lower delay. In the following, the notation $\mathbf{c}' = \mathbf{c}\rangle_j^i$ means that the CSV $\mathbf{c}'$ is identical to $\mathbf{c}$, except that the j'th entry of $\mathbf{c}$ is replaced with class i ($j \in \{1, \ldots, U\}$ and $i \in \{1, \ldots, N\}$).

Lemma 2: Suppose that $\mathbf{c}' = \mathbf{c}\rangle_j^k$ with $k \in \{1, \ldots, N\}$. If $k > c_j$ then $\bar{d}_k(\mathbf{c}') \leq \bar{d}_{c_j}(\mathbf{c})$. Similarly, if $k < c_j$ then $\bar{d}_k(\mathbf{c}') \geq \bar{d}_{c_j}(\mathbf{c})$.

5.2.2 The well-provisioned case

First consider the case when $\mathcal{U}$ is *satisfiable*. We start with some important properties for the set of acceptable CSVs C_A.

A CSV is said to be ordered when users with more stringent delay requirements select higher classes. The following property shows that an acceptable CSV $\mathbf{c}$ can be always replaced with a lower acceptable CSV $\mathbf{c}' \leq \mathbf{c}$ that is *ordered*. For example, if $\mathbf{c} = (1,3,2,4,3)$ is an acceptable CSV, we can construct the CSV $\mathbf{c}' = (1,2,2,3,3)$ that is lower than $\mathbf{c}$ and ordered. As shown in the proof of this property, the CSV $\mathbf{c}'$ is such that $c'_j = \min_{k=j,\ldots,U}\{c_k\}$ for $j = 1,\ldots,U$.

Lemma 3: Given an acceptable CSV $\mathbf{c}$ we can always construct another acceptable CSV $\mathbf{c}' \leq \mathbf{c}$, such that $c'_i \leq c'_j$ for any $i < j$ $(i,j \in \{1,\ldots U\})$.

The following property shows that given two acceptable CSVs, we can construct another acceptable CSV in which each user selects the minimum between the two acceptable classes for that user. For example, if (1,2,2,3,4) and (1,1,3,4,4) are two acceptable CSVs, then the CSV (1,1,2,3,4) is also acceptable. To express this 'per-user minimum class' operation, we write $\mathbf{c}^m = \min\{\mathbf{c}^1, \mathbf{c}^2\}$, or more generally, $\mathbf{c}^m = \min\{\mathbf{c}^1,\ldots,\mathbf{c}^k\}$.

Lemma 4: Suppose that $\mathbf{c}^1$ and $\mathbf{c}^2$ are two acceptable CSVs. The CSV $\mathbf{c}^m$, with $c^m_j = \min(c^1_j, c^2_j)$ $(j = 1,\ldots,U)$, is also acceptable.

Feasibility test and minimum acceptable CSV

We examine whether there exists a CSV in which all users are satisfied, i.e., whether the link $\mathcal{L}$ is *well-provisioned* for $\mathcal{U}$ $(C_A \neq \emptyset)$. If the link is well-provisioned, what is the CSV with the minimum acceptable class for each user? Such a CSV would be optimal for the user population, because all users would be satisfied, and with the minimum cost for each user. Let $\hat{\mathbf{c}}$ be such an optimal CSV. Formally, $\hat{\mathbf{c}}$ is defined as

$$\hat{\mathbf{c}} = \min_{\mathbf{c} \in C_A}\{\mathbf{c}\} \tag{5.4}$$

Based on Lemma 4, $\hat{\mathbf{c}}$ is also acceptable, and by definition, unique. We refer to $\hat{\mathbf{c}}$ as the *minimum acceptable CSV*.

```
min_accept_CSV (c_1, c_2, ..., c_U)
{
// c = (c_1, c_2, ..., c_U).
// Initially, call min_accept_CSV (1, 1, ..., 1).

    compute_class_rates λ(c); // From (5.3).
    compute_class_delays d̄(c); // From (5.2).

    if (c ∈ C_A) // (i.e., d̄_{c_j}(c) ≤ φ_j for all j ∈ 𝒰)
        return (ĉ = c);
    else {
        k = max_{j=1...U} j such that c_j < N;
        if (k == 1 and c_1==N-1)
            return (𝒰: Unsatisfiable);
        else if (k == 1)
            min_accept_CSV (c_1 + 1, c_1 + 1, ..., c_1 + 1, c_1 + 2);
        else
            min_accept_CSV (c_1, c_2, ..., c_{k-1}, c_k + 1, ..., c_k + 1);
    }
}
```

Figure 39: Algorithm to compute the minimum acceptable CSV $\hat{\mathbf{c}}$.

A bruteforce approach to compute $\hat{\mathbf{c}}$ is to search through all CSVs. A more efficient algorithm is shown in Figure 39. The algorithm determines the minimum acceptable CSV $\hat{\mathbf{c}}$, given the rates and delay requirements of the users in $\mathcal{U}$, the average aggregate backlog $\bar{q}_{ag}$ (which is an invariant given $\mathcal{U}$ and $\mathcal{L}$), and the DDPs. The algorithm generates *the sequence of ordered CSVs in increasing order*[2]. For each CSV, it is examined whether it is acceptable. If this is the case, we can show (based on Lemma 3) that this is the minimum acceptable CSV $\hat{\mathbf{c}}$, and the algorithm terminates. If there is no acceptable CSV in the sequence of ordered CSVs, then the user population $\mathcal{U}$ is unsatisfiable, and the link $\mathcal{L}$ is *under-provisioned* for $\mathcal{U}$ ($C_A = \emptyset$). Note that CSVs of the form $(c_i, c_i, \ldots, c_i)$ with $c_i \neq 1$ are not examined, because if they are acceptable, then the lower CSV $(1, 1, \ldots, 1)$ is also acceptable.

This 'well-provisioning' of the link $\mathcal{L}$ for the user population $\mathcal{U}$ depends on the user rates, the user delay requirements, the specified DDPs, and the average aggregate backlog. The latter depends on the traffic burstiness, the aggregate user rate λ, and the link capacity C. Note that the 'inverse' network-design problem is to determine the optimal DDPs and the minimum link capacity that can satisfy a user population $\mathcal{U}$; we study this problem in §5.4.

Distributed DCS model

The algorithm of Figure 39 computes the minimum acceptable CSV in a centralized manner. In practice, users would act independently to select the minimum class that satisfies their delay requirement, without knowing the class selections and delay requirements of other users. What is the resulting CSV in this *distributed DCS model*?

Users perform the following *class transitions* in the distributed DCS model. Note that a user j is supposed to only know the queueing delay $\bar{d}_{c_j} = \bar{d}_{c_j}(\mathbf{c})$ in the class c_j that she uses. If $\bar{d}_{c_j} > \phi_j$ and $c_j < N$ the user moves to the higher class $c_j + 1$, expecting to get a lower average delay (Lemma 2). If the user is already in the highest class N and $\bar{d}_{c_j} > \phi_j$, the user stays unsatisfied in class N. Also, if $\bar{d}_{c_j} \leq \phi_j$ and $c_j > 1$, the user moves to the lower class $c_j - 1$, in

[2] By increasing order, we mean that if $\mathbf{c}^k$ is generated before $\mathbf{c}^l$, $\mathbf{c}^k \leq \mathbf{c}^l$.

order to examine whether the higher delay of that class is also acceptable (Lemma 2). If class $c_j - 1$ is not acceptable, the user returns to class c_j. Note that the occasional transitions to a lower class are necessary in order for users to search for the *minimum* acceptable class.

It is important to note that in this distributed DCS model *a user does not stay in an acceptable class indefinitely.* So, strictly speaking, *the user population does not converge to a certain CSV*, even when $\mathcal{U}$ is satisfiable. We can define, though, a *distributed DCS equilibrium* $\tilde{\mathbf{c}}$ as a CSV such that

a) all unsatisfied users are in the highest class, and

b) when a satisfied user unilaterally moves to the lower class, while all other users remain in the same class, that user becomes unsatisfied.

Formally, let $\mathcal{U}_s(\mathbf{c})$ and $\mathcal{U}_u(\mathbf{c})$, respectively, be the set of satisfied and unsatisfied users for a CSV $\mathbf{c}$. We say that a CSV $\tilde{\mathbf{c}}$ is a distributed DCS equilibrium if it meets the following two conditions:

$$\tilde{c}_j = N \quad \text{for all } j \in \mathcal{U}_u(\tilde{\mathbf{c}}) \tag{5.5}$$

and

$$\text{for any } j \in \mathcal{U}_s(\tilde{\mathbf{c}}) \quad (\text{with } \tilde{c}_j > 1), \quad j \notin \mathcal{U}_s(\mathbf{c}') \text{ where } \mathbf{c}' = \tilde{\mathbf{c}}\rangle_j^{\tilde{c}_j - 1} \tag{5.6}$$

If $\tilde{\mathbf{c}} \in C_A$, we say that it is an *acceptable DCS equilibrium*; otherwise, it is an *unacceptable DCS equilibrium.*

Ren and Park considered a game theoretic model of DCS in [89]. Specifically, [89] showed that *the users converge to a distributed DCS equilibrium (Nash equilibrium), under either sequential or concurrent class transitions*, when three 'per-hop control' properties are met. The PDD model satisfies these properties because of Equation 5.1, Lemma 1, and Lemma 2. So, the results of [89] regarding the existence and stability of the DCS equilibria are applicable to our PDD-based link model. The major result of [89] is that there can be no *persistent cycles* in a sequence of CSVs that results from DCS class transitions. Cycles can occur with concurrent class transitions, but they are only transient, in the sense that from any CSV on the cycle there exist class transitions (sequential or concurrent) that lead to a distributed DCS equilibrium.

It is easy to see that the minimum acceptable CSV $\hat{\mathbf{c}}$ is an acceptable DCS equilibrium. So, if there is only one DCS equilibrium, it has to be $\hat{\mathbf{c}}$. Additionally, it can be shown that if all users start from the lowest class, i.e., if the initial CSV is $\mathbf{c} = (1, 1, \ldots, 1)$, then the resulting DCS equilibrium is $\hat{\mathbf{c}}$.

Other DCS equilibria can also exist however. For example, consider the case of two users and two classes in a well-provisioned link. Suppose that the minimum acceptable CSV is $\hat{\mathbf{c}} = (1, 1)$, and that the CSVs $(1, 2)$ and $(2, 1)$ are unacceptable. Since $(1, 1)$ is acceptable, the CSV $(2, 2)$ is also acceptable (the users encounter the same average delays in both CSVs). Consequently, the CSV $(2, 2)$ is also an acceptable DCS equilibrium. Such acceptable equilibria are suboptimal for the users, because the users need to pay for a higher class than the minimum acceptable class that exists for them. On the other hand, such equilibria may be more preferable for the network provider, since they can lead to higher network revenues.

The resulting DCS equilibria can also be unacceptable, even when the link is well-provisioned. For example, consider a well-provisioned link with three users and three classes, and let $\hat{\mathbf{c}} = (1, 2, 2)$. It is possible that the CSVs $(1, 2, 3)$, $(1, 3, 2)$, and $(1, 3, 3)$ are unacceptable. In that case, the CSV $(1, 3, 3)$ is an unacceptable DCS equilibrium, even though the link is well-provisioned. Unacceptable DCS equilibria are of course suboptimal both for the users and the network provider.

5.2.3 The under-provisioned case

Suppose now that $\mathcal{U}$ is unsatisfiable ($C_A = \emptyset$). By definition, there are some users that cannot meet their delay requirement in any class. Based on the distributed DCS model of the previous paragraph, these users remain unsatisfied in the highest class. As in the well-provisioned case, it can be shown that the population of users converges to a distributed DCS equilibrium, that is always unacceptable in this case.

The following result shows that, in the under-provisioned case, a distributed DCS equilibrium $\tilde{\mathbf{c}}$ is such that the unsatisfied users have the most stringent (i.e., smallest) delay requirements.

Lemma 5: If $\mathcal{U}$ is unsatisfiable, a distributed DCS equilibrium $\tilde{\mathbf{c}}$ is always of the form

$$\tilde{\mathbf{c}} = (c_1, \ldots, c_S, N, \ldots, N) \tag{5.7}$$

where S is the number of satisfied users $(0 \leq S < U)$ in $\tilde{\mathbf{c}}$.

So, when $\mathcal{U}$ is unsatisfiable, S users with the largest delay requirements are satisfied, while the rest $U - S$ users with the smallest delay requirements are unsatisfied in the highest class.

The *maximum value of* S and the minimum acceptable class for those S satisfied users can be determined using the following centralized algorithm. The algorithm starts with $S = U-1$. Given that the U'th user selects class N and is unsatisfied, the algorithm of Figure 39 examines whether there exists an acceptable CSV for the lower S users (including the rate r_U in λ_N). If this the case, the algorithm terminates, returning the minimum class selection for the S lower users. Otherwise, S is decreased to $U - 2$, and the process repeats.

In practice, the unsatisfied users may leave the network, or change their rates and/or delays requirement. Such user behavior leads to a network load relaxation, and to a new user population $\mathcal{U}'$ that may be satisfiable. Unsatisfied users, in practice, also introduce some cost for the network provider (in the form of loosing customers for example).

5.3 Simulation study of an end-to-end DCS algorithm

The model of Section 5.2 considers only a single link, and assumes that users know the exact average queueing delay in the class that they use. Also, the model does not specify the timescales in which users measure the average delays and adjust their class selections. In this section, we focus on an end-to-end DCS algorithm that takes into account all these issues. The behavior of this DCS algorithm in a multi-hop network that is loaded with heavy tailed cross-traffic is studied with simulations. The results of this section provide further insight into the factors that determine the well-provisioning of a network, and in the dynamic behavior of the DCS algorithm.

A complete DCS algorithm: As noted in §5.1, there can be several variations of DCS algorithms, depending on the performance metrics that users are interested in and measure. In the following DCS algorithm, the user specifies a requirement D_{max} on the maximum Round-Trip Delay (RTD) in the flow's path. Users measure the RTD as follows. The sender timestamps each packet k before transmitting it, while the receiver returns the timestamps back to the sender in the same class that they are received in. The sender measures the RTD D_k of the k'th packet, subtracting the packet's timestamp from the current time. In a bi-directional flow, such as a telephony session, the flow of timestamps back to the sender can be piggybacked with the reverse data flow. The minimum RTD D_{min} is also measured, in order to estimate the total propagation and transmission delays in the path; obviously, the user cannot ask for $D_{max} < D_{min}$.

After the k'th timestamp is received, the DCS sender estimates the *average RTD* $\tilde{D}$ using an exponential running-average of the form

$$\tilde{D}_{k+1} = (1 - w)\tilde{D}_k + wD_k \tag{5.8}$$

where $0 < w \leq 1$ is the averaging weight. $\tilde{D}$ expresses the performance timescales that the user is interested in. For instance, if $w = 1$ the user cares about *per-packet RTDs*, which may be the case in a highly interactive application. If $w = 0.1$, the last 50 RTDs contribute 99% to $\tilde{D}_k$, while the last 10 RTDs contribute 65% to $\tilde{D}_k$. If $w = 0.01$, the last 500 RTDs contribute 99% to $\tilde{D}_k$, while the last 100 RTDs contribute 63% to $\tilde{D}_k$. Unless stated otherwise, w is set to 0.01 (one of the following experiments, though, also considers per-packet RTD constraints).

The Dynamic Class Selection (DCS) algorithm is shown in detail in Figure 40. The class that the user selects is denoted by c, with $c \in \{1, \ldots, N\}$. When the measured RTD D_k is larger than D_{max}, the class selection is increased. This decision is based on the last RTD D_k, instead of

DCS algorithm:
{
D_k= k'th RTD measurement;
if ($D_k > D_{max}$) {
$c = \min\{c + \lceil \log_2 \frac{D_k - D_{min}}{D_{max} - D_{min}} \rceil, N\}$;
Do not increase class in next $f_I D_k$ time units;
}
if ($D_k < \kappa D_{max}$) {
$c = \max\{c - 1, 0\}$;
Do not decrease class in next $f_D D_k$ time units;
}
}

Figure 40: DCS algorithm.

the average $\tilde{D}_k$, in order to react faster to excessive delays in the current class. The class increase is based on the deviation of the measured queueing delay $D_k - D_{min}$ from the maximum allowed queueing delay $D_{max} - D_{min}$. In this particular case, the DCS algorithm assumes that the DDPs in each hop are ratioed as $\delta_1/\delta_i = 2^{i-1}$, and so the appropriate class increase is given by the logarithmic formula shown. The DDPs can be estimated measuring the queueing delay ratios between successive classes. A DDP-based class increase can move the user to the appropriate class faster. We emphasize, though, that *it is not required that the sender knows the actual DDPs in the network*, and that a simpler class increase formula can be used instead. After the class increase, the sender waits for some time $f_I D_k$ before a further class increase. This *adaptation delay* is needed in order to measure the resulting RTD in the new class selection. A typical value for the adaptation delay factor would be f_I=1 (i.e., wait for one RTD), but a more delay-sensitive user can set f_I to less than one assuming that the class increase will cause a lower RTD. Unless stated otherwise, f_I=1.

The class is decreased, on the other hand, when the RTD is lower than κD_{max} ($\kappa < 1$). κ is a *tolerance factor* for the maximum RTD that triggers a class decrease. A user that is not so interested to minimize the cost of her flow can reduce κ. The class is only decremented by one, since the reaction latency is not as critical in the class decrease as in the class increase case.

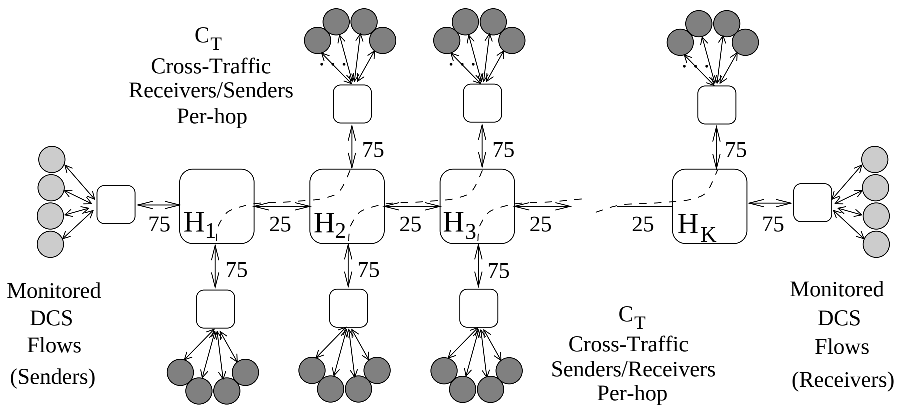


Figure 41: Simulation topology.

A class decrease is also followed by an adaptation delay $f_D D_k$. Cost-sensitive users can set f_D close to one, while delay-sensitive users can set f_D to a higher value. Unless stated otherwise, $f_D = 4$ and κ=0.9. Note that when $\kappa < 1$, it is possible that a user settles in a class that is not the minimum acceptable class. This is a trade-off for the user, since a lower κ reduces the 'annoyance' of lower class transitions, but it can also lead to a suboptimal class.

Simulation topology and parameters: Figure 41 shows the multi-hop simulation topology. The *monitored* DCS flows go through K=5 backbone hops in the horizontal direction. The Cross-Traffic (CT) in the path is generated from bi-directional flows that go through the topology in the vertical direction. C_T=50 such flows are created in each hop. The K backbone links have a 25Mbps capacity, while the rest of the links (access links) have a 75Mbps capacity. The propagation delay in each link is τ=5msec. The monitored DCS flows have a minimum RTD $D_{min} = 2(K+2)\tau$=70 msec due to propagation delays. The CT flows go through 3 links, and so their minimum RTD is 6τ=30msec.

The class adjustments are performed at the DCS nodes labeled as 'Senders'. The CT sources are either DCS flows themselves, or they generate packets with a given, average class load distribution (non-DCS flows). Notice that both directions in the backbone path are equally loaded from the CT sources. Both the monitored and CT sources generate packets based on a Pareto distribution with infinite variance (α=1.5). The packet size is fixed to 500 bytes for all sources. The monitored DCS flows have an average rate of 400kbps, or 100 packets per second. The rate of the CT sources is adjusted to cause a certain utilization u in the backbone links; u is set to 90% in the following experiments.

The backbone links use the WTP scheduler (§3.3) to provide proportional delay differentiation[3]. The network offers N=8 classes with DDPs $\delta_1/\delta_i = \{1, 2, 4, 8, 12, 16, 24, 32\}$ $(i = 1 \ldots 8)$, unless stated otherwise. Notice that these DDPs are not entirely consistent with the 'power-of-two' DDPs that the algorithm of Figure 40 assumes. When the CT is created from non-DCS flows, the average class load distribution is (10,20,20,15,15,10,5,5)[4]. When the CT is created from DCS flows, the class load distribution is determined from the delay requirements of those flows. The number of buffers in the links is adequately large to avoid losses.

Performance metrics: The performance of a DCS flow is measured with the fraction $\mathcal{P}$ of RTD values $\tilde{D}_k$ that are lower than the specified maximum RTD D_{max}. For K RTD values,

$$\mathcal{P} = \frac{\sum_k^K I(D_{max} - \tilde{D}_k)}{K} \tag{5.9}$$

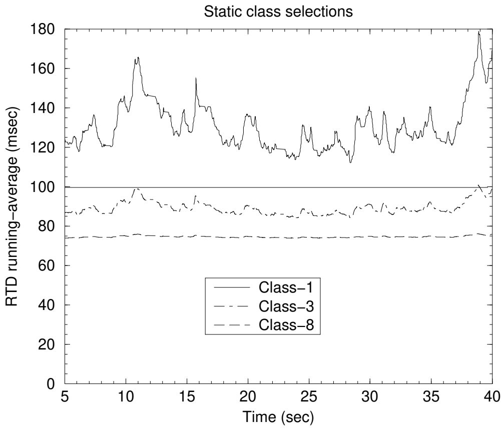
where $I(x)$ is zero if $x < 0$ and one otherwise. We refer to $\mathcal{P}$ as the *acceptable delay ratio*. Note that the acceptable delay ratio is based on the running average $\tilde{D}_k$, and not on the individual RTD measurements, because $\tilde{D}_k$ expresses the performance timescales that the user cares about.

The cost of a DCS flow is measured with the *average class* metric $\mathcal{C}$. If the k'th packet was sent in class c_k, the average class of a DCS flow that transferred K packets is

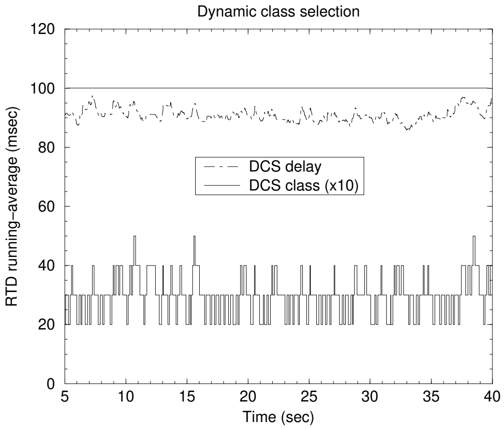
$$\mathcal{C} = \frac{\sum_k c_k}{K} \tag{5.10}$$

A lower value of $\mathcal{C}$ causes a lower cost for the user, since higher classes are assumed to be more expensive (in a monetary or other sense).

[3] In the high utilization range that we simulate here, the differences between WTP and HPD (§3.4) are minor.
[4] Each CT source generates traffic with this class load distribution.



(a) Static class selection



(b) Dynamic class selection

Figure 42: Static versus dynamic class selection for a flow with D_{max}=100msec.

The DCS framework introduces a trade-off between the performance and cost of a flow. Higher classes lead to lower delays, but they also cost more. The objective of the DCS algorithm is to achieve a high acceptable delay ratio $\mathcal{P}$ and a low average class $\mathcal{C}$. The exact point in this trade-off can be chosen from the user, based on her performance/cost sensitivity. The algorithm of Figure 40 allows the control of this trade-off, through the parameters f_I, f_D, and κ.

Static versus dynamic class selection: We first compare the DCS algorithm with a simple static class selection scheme. A monitored flow has a maximum RTD requirement D_{max}=100msec. Figure 42-a shows a sample path of the RTD average $\tilde{D}$ in the case of three static class selections. Class-1 provides excessive delays, and the acceptable delay ratio is only $\mathcal{P}$=0.12. Class-8, on the other extreme, leads to much lower delays than needed, and $\mathcal{P}$=1.00. Class-3 turns out to be the minimum class that leads to an acceptable RTD for almost all packets ($\mathcal{P}$=0.99). So, Class-3 is the minimum acceptable class for this flow.

Figure 42-b shows what happens when the monitored flow uses DCS instead of a static class selection. The class selection variations are also shown (scaled by a factor of ten). Note that

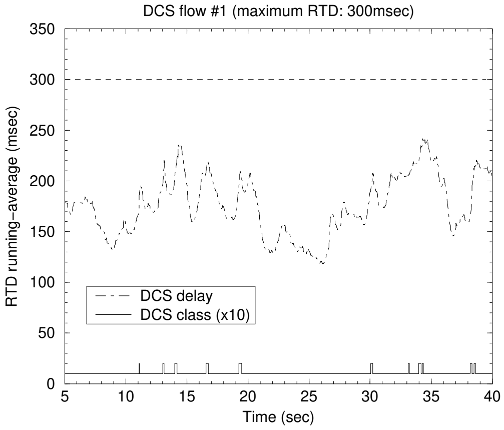
DCS meets the specified RTD constraint, and the acceptable delay ratio is $\mathcal{P}$=1.00. The average class metric is $\mathcal{C}$=2.92, and so the flow uses mainly Class-3, at least in an average sense. This is also shown from the class selection variations in the graph. In other words, *in the resulting DCS equilibrium of this experiment, the DCS flow selects (in an average sense) the minimum acceptable class for that flow*, namely Class-3. The oscillations around Class-3 are caused by attempts to find a lower acceptable class, and by some excessive RTDs, due to traffic bursts, that cause temporary class increases.

Satisfied and unsatisfied DCS flows: In this experiment, we focus on four monitored DCS flows with diverse maximum RTD requirements. The value of D_{max} for these flows is 300, 150, 100, and 75msec. The first three flows find a class in which the acceptable delay ratio is $\mathcal{P}$=1.00, and so they are *satisfied*. The average class for these flows is $\mathcal{C}$=1.06, 2.13, and 4.17, respectively. The fourth flow, on the other hand, requires that D_{max}=75msec, which is only 5msec more than the propagation delays. The flow moves to Class-8, but still cannot meet its RTD requirement. The acceptable delay ratio for this flow is only $\mathcal{C}$=0.13, which implies that it is *unsatisfied*.

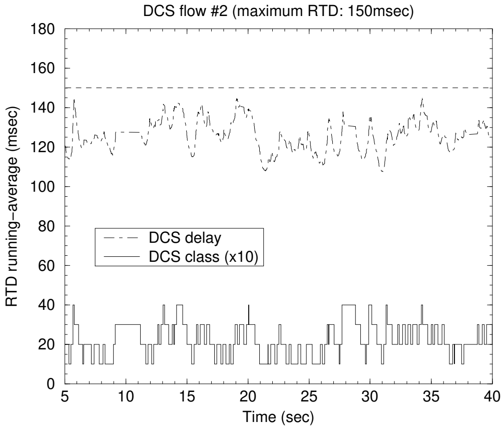
In other words, *the unsatisfied users, if they exist, are flows with the most stringent requirements, they move to the highest class, and they remain unsatisfied there. The satisfied users, on the other hand, converge to an acceptable class, oscillating around that DCS equilibrium.*

The performance versus cost tradeoff in DCS: As noted earlier, there is a trade-off between the performance and cost of a flow. The DCS algorithm allows the user to control the operating point in this trade-off through the parameters f_I and f_D (the adaptation-delay parameters), and κ (the RTD tolerance factor). These parameters control how often the DCS algorithm checks for a required class increase (f_I), for a possible class decrease (f_D), and when is a class decrease allowed (κ).

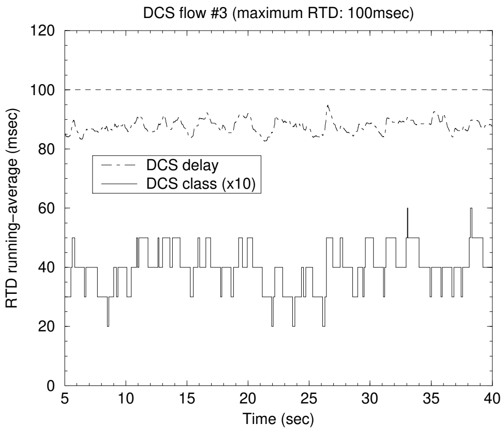
To illustrate the performance versus cost trade-off, consider a flow that has a stringent requirement on the *individual RTDs of each packet*. Such a constraint means that w=1 in



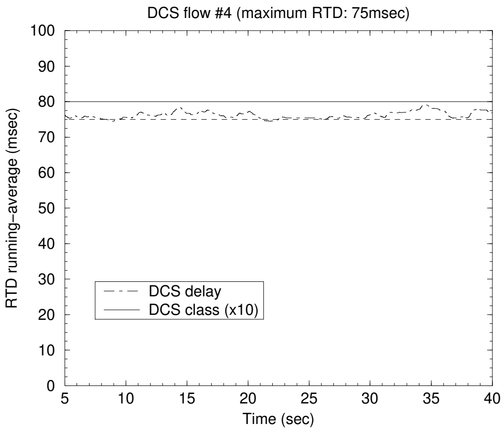
(a) Flow-1: D_{max}=300msec



(b) Flow-2: D_{max}=150msec

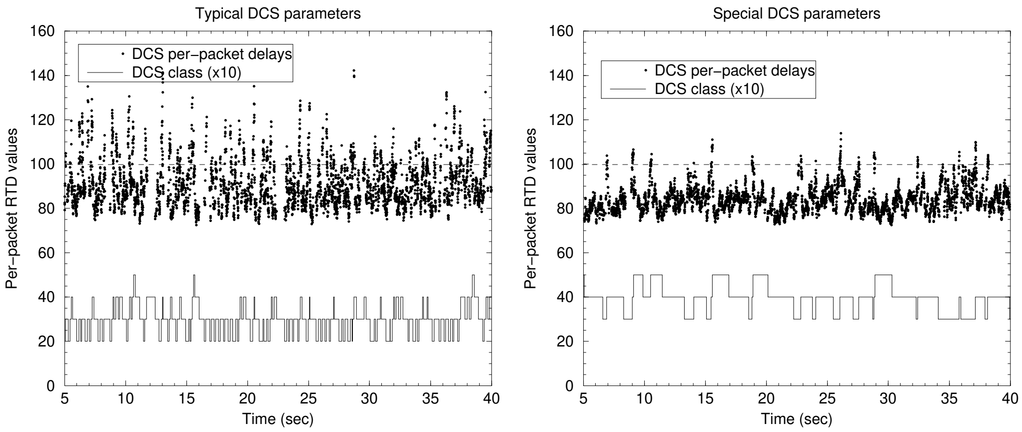


(c) Flow-3: D_{max}=100msec



(d) Flow-4: D_{max}=75msec

Figure 43: Three satisfied and one unsatisfied DCS flows.



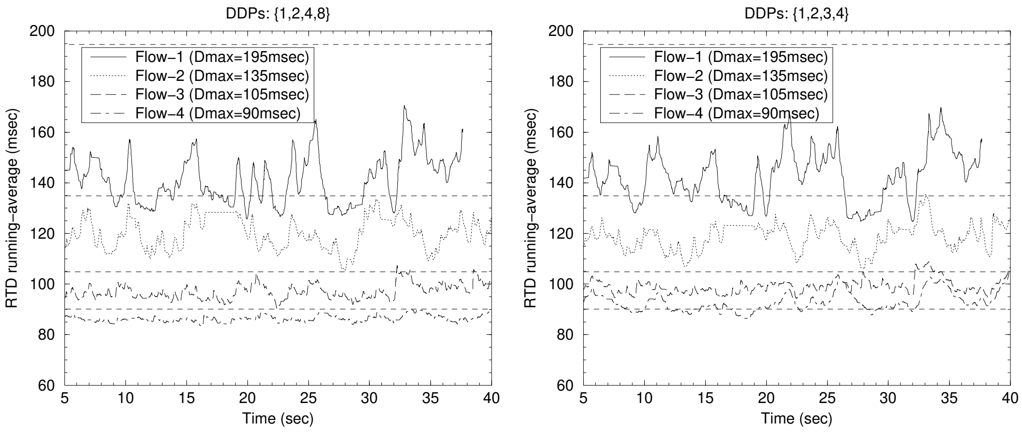
(a) $f_I = 1, f_D = 4, \kappa = 0.9$

(b) $f_I = 0.9, f_D = 20, \kappa = 0.8$

Figure 44: Controlling the DCS parameters to meet a per-packet RTD requirement.

(5.8). The maximum RTD requirement is D_{max}=100msec. Figure 44-a shows a sample path of the per-packet RTDs with the 'typical' DCS parameters that we mentioned earlier (f_I=1, f_D=4, κ=0.9). The acceptable delay ratio is only $\mathcal{P}$=0.81, and the average class is $\mathcal{C}$=2.92. Violating the RTD bound in about 20% of the packets would probably be unacceptable for several interactive applications.

Figure 44-b shows the resulting sample path of per-packet RTDs, when the DCS parameters are selected as f_I=0.9, f_D=20, and κ=0.8. The acceptable delay ratio now is significantly improved to $\mathcal{P}$=0.97, and only 3% of the packets miss their deadlines. The average class $\mathcal{C}$ is increased from 2.92 to 4.21, though, which shows the price that has to be paid for the more stringent QoS. This experiment illustrates two important points. First, *the DCS framework can be used to meet absolute RTD requirements not only in terms of averages, but also for individual packets*. Second, *meeting a more strict performance requirement requires normally the use of higher classes, and thus a larger cost.* A practical DCS algorithm has to provide the flexibility to control this trade-off.



(a) Well provisioned DDPs

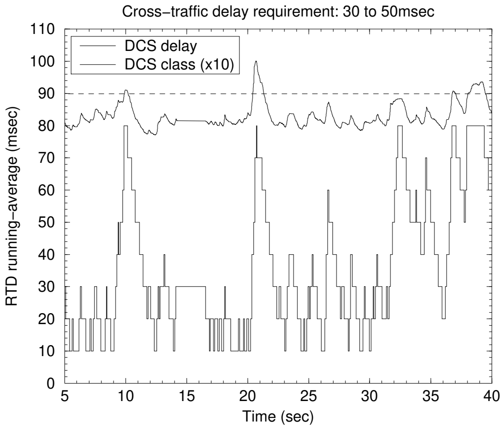
(b) Poorly provisioned DDPs

Figure 45: The effect of the DDPs on DCS flows.

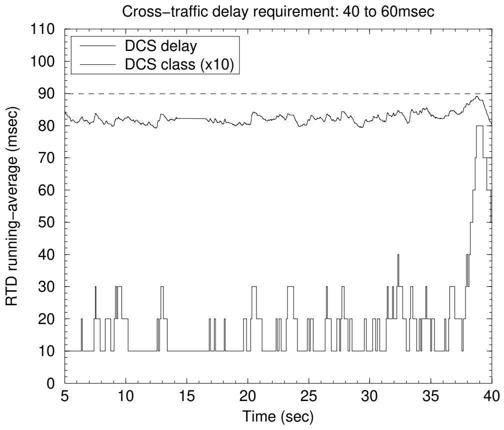
Class provisioning and the selection of DDPs: This experiment illustrates the importance of selecting appropriate DDPs for a given DCS traffic mix. Four DCS monitored flows have a D_{max} requirement of 195, 135, 105, and 90msec, respectively. The network offers four classes. How is the selection of DDPs important in satisfying these DCS flows?

Suppose first that $\delta_1/\delta_i = \{1, 2, 4, 8\}$ $(i = 1 \ldots 4)$. Figure 45-a shows that with these DDPs each DCS flow gets an almost perfect acceptable delay ratio $\mathcal{P} \approx 1.00$. The average class $\mathcal{C}$ for the four flows is 1.28, 1.97, 2,87, and 3.59, respectively. The given DDPs were not selected randomly in this experiment. On the contrary, *they were chosen based on the queueing delay requirements of the DCS flows.* To see how, note that the minimum RTD in the path (due to propagation and transmission delays) is about D_{min}=75msec. So, the four flows can tolerate up to 120, 60, 30, and 15msec of queueing delays in the round-trip path. These maximum queueing delays are ratioed as: 120/15=8, 60/15=4, and 30/15=2. So, the DDPs in this case are ratioed based on the corresponding ratios of the queueing delay requirements of the DCS flows.

Figure 45-b, on the other hand, shows what happens if the network operator chooses the DDPs $\delta_1/\delta_i = \{1, 2, 3, 4\}$ $(i = 1 \ldots 4)$. These DDPs do not match well the queueing delay



(a) More demanding CT flows



(b) Less demanding CT flows

Figure 46: The effect of the CT delay requirements on a DCS flow.

requirements of the DCS flows. Specifically, these DDPs only provide a maximum delay ratio of four between the highest and the lowest class, while the DCS flows require a maximum ratio of eight. The result of this mismatch is that the flow with the tightest delay requirement (Flow-4) remains unsatisfied in the highest class with an acceptable delay ratio $\mathcal{P}$=0.29. It is important to note that the network utilization is the same in both cases; the only difference is the specified DDPs.

This experiment illustrates the importance of selecting DDPs that are appropriate for the delay requirements of DCS flows. The problem of computing the optimal DDPs and the minimum link capacity that are required for a certain user population is studied in §5.4.

The effect of the Cross-Traffic (CT) performance requirements: In the experiment of this paragraph, *all sources, both monitored and CT, generate DCS flows.* In this scenario, an important parameter is the delay requirements of the CT flows. Since they also perform DCS, the more stringent delay requirements they have, the higher classes they use, and so the harder it becomes for any DCS flow to find an acceptable class. Figure 46 shows what happens to a monitored DCS flow with a maximum RTD requirement D_{max}=90msec, in two different cases of CT delay requirements.

In Figure 46-a, the CT flows ask for a maximum RTD that is uniformly distributed in the range 30-50msec. Note that the minimum RTD in the CT path is about 30msec. So, some of the CT flows in this case ask for practically zero queueing delays. Such a demanding CT load causes a relatively low acceptable delay ratio to the monitored flow ($\mathcal{P}$=0.86), while its average class is $\mathcal{C}$=3.69. In Figure 46-b, on the other hand, the CT flows are less demanding, asking for a maximum RTD in the range 40-70msec. Even though the aggregate load and the DDPs remain the same, the monitored DCS flow is able now to get a perfect acceptable delay ratio ($\mathcal{P}$=1.00) with a lower cost ($\mathcal{C}$=1.94).

To summarize the experiments of this section, the satisfied DCS flows converge to an acceptable class, and oscillate around that DCS equilibrium. Unsatisfied DCS flows move to the highest class and remain unsatisfied there. The trade-off between a flow's performance and cost can be controlled using certain parameters of the DCS algorithm. Finally, the 'well-provisioning' of a network for a given traffic workload depends on the link utilization, on the delay requirements (or static class selection) of the cross-traffic, and on the provisioned delay differentiation between classes.

5.4 Class provisioning

A central point in the relative differentiation architecture (§2.1) is that the network provider does not control the aggregate load or the class load distribution in a network link. This is because the architecture does not use admission control, or any other similar mechanisms. Additionally, the class loads can vary dramatically in an IP network due to routing changes, traffic burstiness, or as shown in the previous sections, due to dynamic class selections at the end-points. Obviously, if the class loads are not known a priori, the QoS level of each class cannot be controlled, and so the provider can only offer some kind of relative differentiation.

In this section, we examine this issue from a different perspective. We consider the case that the network provider has at least a rough estimate of the traffic workload in a link. By 'workload estimate', we mean that *the provider knows the major 'traffic types' that the link carries, as well as an average rate for each traffic type in different times of day and/or days of week*. Such information is often available, based on operational data and statistics, in stable networks that are well monitored [17]. In this section, we show that *a network provider can use this workload estimate, when it is available, to provide an absolute QoS level to each traffic type.* We refer to this methodology as *class provisioning*.

The exact form of class provisioning we consider here *provides an upper bound on the average queueing delay of each traffic type*. For example, a network provider can provision a maximum average delay of 50msec for the E-mail, Network-News (NNTP), and other 'bulk' traffic, 20msec for the WWW traffic, and 10 msec for the IP telephony and video conferencing traffic. The objective in the proposed provisioning methodology is to compute *the minimum required link capacity* that meets the delay requirements of the supported traffic types. This is important, especially when the network bandwidth is a scarce resource.

The proposed class provisioning methodology is based on the PDD model (§3.1). The PDD model provides a simple way to adjust the queueing delay spacing (i.e., ratios) between the offered classes, based on the delay requirements of the given traffic types. Before expanding more on the class provisioning methodology, let us first note how the provisioning problem relates to the Dynamic Class Selection (DCS) framework of §5.1. In DCS, the users aim for an absolute end-to-end performance in a network that only offers relative differentiation between classes, and so they have to dynamically search for an acceptable class. The capacity and the DDPs of each link, in that framework, are given and fixed. In class provisioning, on the other hand, the network provider determines the link capacity and the DDPs, so that the link provides a desired absolute QoS level to each traffic type.

5.4.1 Problem description and formulation

Consider a network link $\mathcal{L}$. Suppose that $\mathcal{L}$ carries M *traffic types*. A traffic type is an aggregation of flows that have the same performance requirements. In our model, all flows in a traffic type have the same average delay requirement in $\mathcal{L}$. Specifically, a traffic type j is characterized by a maximum average queueing delay requirement χ_j, and an average rate ζ_j. Without loss of generality, the traffic types are ordered based on their delay requirement, as $\chi_1 > \chi_2 > \ldots > \chi_M > 0$.

Our model for $\mathcal{L}$ is that of *a lossless PFE* with N classes of service, as presented in §3.1. The link capacity is denoted by C, the offered rate in class i is λ_i, and the aggregate offered rate is $\lambda = \sum_i \lambda_i$. The PFE uses a work-conserving, non-preemptive, proportional delay scheduler $\mathcal{S}$, such as HPD, that can achieve (or approximate) the PDD model when the specified DDPs are feasible (§3.1). We assume that the utilization is $u = \lambda/C < 1$, and that the PFE has an adequate number of buffers for lossless operation.

The class provisioning methodology consists of two parts. First, we determine the *optimal average delay* $\hat{v}_i$ and the corresponding *optimal offered rate* $\hat{h}_i$ for each class $i = 1 \ldots N$. The objective in the selection of the N pairs $\{(\hat{v}_i, \hat{h}_i), i = 1 \ldots N\}$ is that $\mathcal{L}$ *meets the average delay requirements* $\{\chi_j, j = 1, \ldots, M\}$ *of the M traffic types with the minimum capacity requirement.* Second, we determine this minimum link capacity $\hat{C}$, as well as the DDPs $\{\hat{\delta}_i, i = 2 \ldots N\}$ required to meet the previous objective.

In summary, the class provisioning methodology has the following outcomes:

1. The optimal average delay $\hat{v}_i$ and the optimal offered rate $\hat{h}_i$ in each class i.
2. The *nominal mapping* from each traffic type to the corresponding service class.
3. The minimum capacity $\hat{C}$ in $\mathcal{L}$.
4. The required Delay Differentiation Parameters (DDPs) $\{\hat{\delta}_i\}$.

5.4.2 Optimal Class Operating Point (COP) selection

We define a *Class Operating Point (COP)* as a vector $\mathbf{v} = \{v_1, \ldots, v_N\}$, such that $v_1 \geq v_2 \geq \ldots \geq v_N > 0$, where v_i is the desired (target) average delay in class i. A COP $\mathbf{v}$ is *acceptable* when for each traffic type j there exists at least one class i such that $v_i \leq \chi_j$. Let V be the set of acceptable COPs. If $\mathbf{v} \in V$, then for each traffic type j there exists a class $n(j) \in \{1 \ldots N\}$ such that $v_{n(j)} \leq \chi_j < v_{n(j)-1}$ ($v_0 = \infty$).

Given an acceptable COP $\mathbf{v}$, each traffic type j is supposed to use class $n(j)$, since that is the *minimum class* that satisfies the delay requirement of j. We say that traffic type j *maps* to class $n(j)$, or that $n(j)$ is the *nominal class* for traffic type j. Note that when $M > N$, which would probably be the case in practice, there will be more than one traffic types mapping to certain classes. Some classes, that are referred to as *void*, may not be nominal for any traffic type. To denote the inverse mapping, from classes to traffic types, $t(i)$ is the *maximum* traffic type that maps to class i; if class i is void, then $t(i)$=0.

The *expected rate* h_i in class i is the aggregate rate of all traffic types that map to class i. Since an acceptable COP $\mathbf{v}$ uniquely determines the nominal class for each traffic type, the expected rates are a function of $\mathbf{v}$,

$$\mathbf{h}(\mathbf{v}) = \{h_1(\mathbf{v}), h_2(\mathbf{v}), \ldots, h_N(\mathbf{v})\} \quad \text{with} \quad h_i(\mathbf{v}) = \sum_{j:n(j)=i}^{M} \zeta_j \geq 0 \tag{5.11}$$

When the particular $\mathbf{v}$ that we consider is obvious, we write $\mathbf{h}$ or h_i, instead of $\mathbf{h}(\mathbf{v})$ or $h_i(\mathbf{v})$, respectively. The *total expected rate* in the link is

$$h = \sum_{i=1}^{N} h_i = \sum_{j=1}^{M} \zeta_j \tag{5.12}$$

that is independent of $\mathbf{v}$.

We say that an acceptable COP is *realized* if the average delay in each class i becomes $\bar{d}_i = v_i$ when the class average rates are $\lambda_k = h_k$ for $k = 1 \ldots N$. The link capacity that is required for realizing $\mathbf{v}$ is called the *capacity requirement* of $\mathbf{v}$ and is denoted by $C(\mathbf{v})$. When $\mathbf{v}$ is realized, the aggregate backlog in the PFE becomes

$$\bar{q}_{ag}(\mathbf{v}) = \sum_{i=1}^{N} \bar{d}_i \lambda_i = \sum_{i=1}^{N} v_i h_i \tag{5.13}$$

Recall from §3.1 that the average backlog $\bar{q}_{ag}$ depends on the link utilization and the statistical characteristics of the traffic, and not on the scheduler or the class load distribution.

In order to compute the capacity requirement of a COP $\mathbf{v}$, we need to know the average backlog $\bar{q}_{ag}$ as a function of the link utilization u. In the following, we assume that the average backlog is given by a *backlog function* $\bar{q}_{ag} = \beta(u)$ of the link utilization u. $\beta(u)$ is a strictly increasing and convex function for $u \in (0, 1)$, that becomes unbounded as u tends to one. $\beta(u)$ is invertible, meaning that the link utilization u can be computed from the average backlog through the *inverse backlog function* $\beta^{-1}(\bar{q}_{ag})$. The problem of estimating the average backlog function in practice is discussed in §5.4.4. Given the inverse backlog function, we can determine the capacity requirement of $\mathbf{v}$ from

$$C(\mathbf{v}) = \frac{h}{u(\mathbf{v})} = \frac{h}{\beta^{-1}(\bar{q}_{ag}(\mathbf{v}))} \tag{5.14}$$

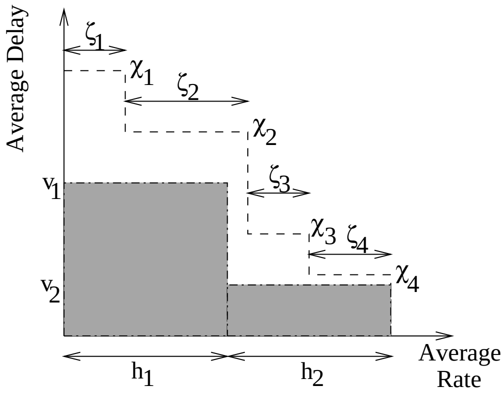
where $u(\mathbf{v})$ is the link utilization that creates an average backlog of $\bar{q}_{ag}(\mathbf{v})$.

An important part of the class provisioning methodology is to select the *optimal COP* $\hat{\mathbf{v}}$ among all acceptable COPs. The optimality constraint in the selection of $\hat{\mathbf{v}}$ is that *it has to be the acceptable COP with the minimum capacity requirement*,

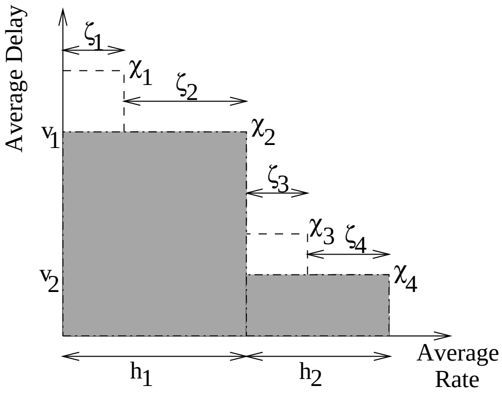
$$\hat{\mathbf{v}} = \arg\min_{\mathbf{v} \in V} C(\mathbf{v}) \tag{5.15}$$

Since the average backlog function $\bar{q}_{ag} = \beta(u) = \beta(\lambda/C)$, though, is strictly increasing, it is easy to see that *the COP with the minimum capacity requirement is the COP with the maximum average backlog.* So, the optimal COP is the acceptable COP with the maximum average aggregate backlog,

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v} \in V} \bar{q}_{ag}(\mathbf{v}) \tag{5.16}$$



(a) An acceptable COP



(b) The optimal COP

Figure 47: An acceptable COP and the optimal COP for a link with N=2 classes and M=4 traffic types.

To determine $\hat{\mathbf{v}}$ in practice, we only need to consider a finite set of acceptable COPs. To see why, consider the example of Figure 47. The example refers to a link with N=2 classes and M=4 traffic types, and it shows two acceptable COPs. In the COP of Figure 47-a, the first traffic type and a large part of the second traffic type are mapped to Class-1; the rest of the traffic is mapped to Class-2. Notice that the four traffic types meet their delay requirements with this class mapping, but there is some 'waste' of resources since the two classes provide lower delays than what the traffic types need.

In the COP of Figure 47-b, on the other hand, the average delay in each class is equal to the delay requirement of one of the traffic types. Specifically, the first two traffic types map to Class-1, which offers the delay requirement χ_2 of the second traffic type, while the two higher traffic types map to Class-4, which offers the delay requirement χ_4 of the fourth traffic type. Note that the shaded area in each COP represents the average backlog $\bar{q}_{ag}(\mathbf{v}) = \sum_{i=1}^{N} v_i h_i$. The optimal COP has to maximize the average backlog, and thus, to maximize the shaded area in Figure 47. In the previous example, the COP of Figure 47-b can be shown to be optimal.

Based on the graphical insight from the previous example, we can see that the optimal COP $\hat{\mathbf{v}}$ satisfies the following properties. First, *each optimal class delay* $\hat{v}_i$ *should be equal to the*

```
optimal_cop (t_1, t_2, ..., t_{N-1},  i, max_q, best_cop)
{
// t_i: maximum traffic type (∈ {1...M}) that maps to class i.
// max_q and best_cop are call-by-reference arguments.
// Initially, call optimal_cop (0, 0, ..., 0,  1,  0,  ∅).
// The optimal COP v̂ is returned in the best_cop argument.
// avg_backlog() computes the backlog of a COP as in (5.13).
// Note: t_N=M and t_0 = 0.

    if (i ≤ N − 1) {
        for t_i = (t_{i-1} + 1)  to  (M − N + i)
            optimal_cop (t_1, t_2, ..., t_{N-1},  i + 1, max_q, best_cop);
    }
    else {
        q = avg_backlog (χ_{t_1}, χ_{t_2}, ..., χ_{t_{N-1}}, χ_{t_N});
        if (q > max_q) {
            max_q = q;
            best_cop = (χ_{t_1}, χ_{t_2}, ..., χ_{t_{N-1}}, χ_{t_N});
        }
    }
}
```

Figure 48: Algorithm to determine the optimal COP $\hat{\mathbf{v}}$.

delay requirement of a traffic type, i.e., for each $i = 1 \ldots N$ there is a $j \in \{1 \ldots M\}$ such that $\hat{v}_i = \chi_j$. Second, *the optimal COP should not have void classes* , because void classes always lead to an average backlog that is less than maximum. So, if $\hat{v}_i = \chi_j$, then there should be no other class k with $\hat{v}_k = \chi_j$. Third, following from the previous two properties, the target delay for the highest class should be the most stringent traffic type delay requirement, i.e., $\hat{v}_N = \chi_M$.

Putting the previous three properties together, we see that the finite set of acceptable COPs that should be examined in order to determine the optimal COP $\hat{\mathbf{v}}$ is

$$\{\mathbf{v} \in V : v_1 > v_2 > \ldots v_N, \ \forall i = 1 \ldots N, \exists j \in \{1 \ldots M\} \text{ such that } v_i = \chi_j \quad (v_N = \chi_M)\} \quad (5.17)$$

Note that the strict inequalities between the v_i's prevent the existence of void classes.

A recursive algorithm for selecting the optimal COP is shown in Figure 48. The run-time complexity of the algorithm is $\mathrm{O}\Big((M-N)^{N-1}\Big)$. For instance, in the case of N=3 classes and $M \geq 3$ traffic types, the algorithm examines $(M-N+1)(M-N+2)/2$ COPs. Since the provisioning methodology is performed off-line, and the number of classes and traffic types is expected to be relatively small (e.g., N=8, M=16), the run-time complexity of the algorithm is not prohibitive.

Example of optimal COP selection:

Suppose that a certain link supports N=2 classes and M=3 traffic types. We need to consider two COPs, depending on whether the maximum traffic type that maps to Class-1 is traffic type 1 or 2. Specifically, the two COPs are:

$$\mathbf{v_1} = (\chi_1, \chi_3) \text{ with } \mathbf{h}_1 = (\zeta_1, \zeta_2 + \zeta_3) \quad \text{and} \quad \mathbf{v_2} = (\chi_2, \chi_3) \text{ with } \mathbf{h}_2 = (\zeta_1 + \zeta_2, \zeta_3)$$

The average backlog in the two COPs is:

$$\bar{q}_{ag}(\mathbf{v_1}) = \chi_1\zeta_1 + \chi_3(\zeta_2 + \zeta_3) \quad \text{and} \quad \bar{q}_{ag}(\mathbf{v_2}) = \chi_2(\zeta_1 + \zeta_2) + \chi_3\zeta_3$$

Which COP has the maximum average backlog depends on the relation between the traffic type rates and average delay requirements. If $\zeta_1(\chi_1 - \chi_2) > \zeta_2(\chi_2 - \chi_3)$, then $\bar{q}_{ag}(\mathbf{v_1}) \geq \bar{q}_{ag}(\mathbf{v_2})$ and the optimal COP is $\mathbf{v_1}$; otherwise, the optimal COP is $\mathbf{v_2}$.

5.4.3 Minimum link capacity and optimal Delay Differentiation Parameters (DDPs)

In the first part of the class provisioning methodology, the goal was to determine the acceptable mapping from traffic types to classes that leads to the minimum capacity requirement. Given the given COP $\hat{\mathbf{v}}$ and the corresponding expected rate vector $\mathbf{h}(\hat{\mathbf{v}})$, the second part of the provisioning methodology determines the minimum capacity requirement and the required DDPs.

The minimum capacity requirement $\hat{C}$ can be computed using the inverse backlog function, as

$$\hat{C} = C(\hat{\mathbf{v}}) = \frac{h}{\beta^{-1}\left(\bar{q}_{ag}(\hat{\mathbf{v}})\right)} = \frac{h}{\beta^{-1}\left(\sum_{i=1}^{N} \hat{v}_i \hat{h}_i\right)} \tag{5.18}$$

where h is the total expected rate given in (5.12). The required DDPs, on the other hand, are simply the ratios of the corresponding optimal average class delays, i.e.,

$$\frac{\hat{\delta}_i}{\hat{\delta}_1} = \frac{\hat{v}_i}{\hat{v}_1} \qquad i = 2 \ldots N \tag{5.19}$$

with $\hat{\delta}_1$=1.

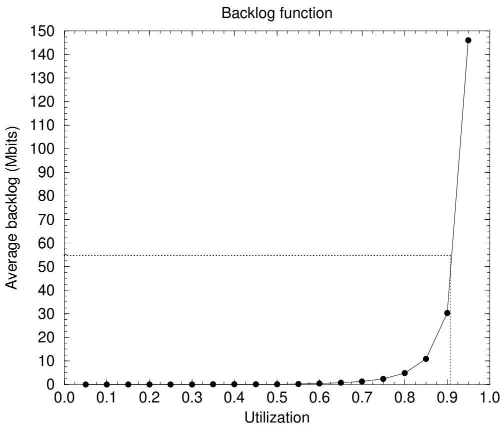
Note that the proportional delay differentiation adjusts the ratio between the class average delays, based on the corresponding delay ratios in the optimal COP. Without an appropriate capacity though, even though the delay ratios will be as in the optimal COP, the actual average class delays will not. If the capacity is $C > \hat{C}$, it is easy to see that *all* class delays will be lower than the optimal COP delays, i.e., $\bar{d}_i < \hat{v}_i$ for all i. We refer to such an operating region as *over-provisioning*. On the other hand, if the capacity is $C < \hat{C}$, *all* class delays will be higher than the optimal COP delays, i.e., $\bar{d}_i > \hat{v}_i$ for all i. We refer to such an operating region as *under-provisioning*. In practice, of course, there will also be a *well-provisioning* operating region in which the capacity is $C \in (\hat{C}_-, \hat{C}_+)$ where $\hat{C}_+ = \hat{C}$ and $\hat{C}_- = f\hat{C}$, with f being a tolerance factor ($f < 1$).

Example of DDP and capacity selection:

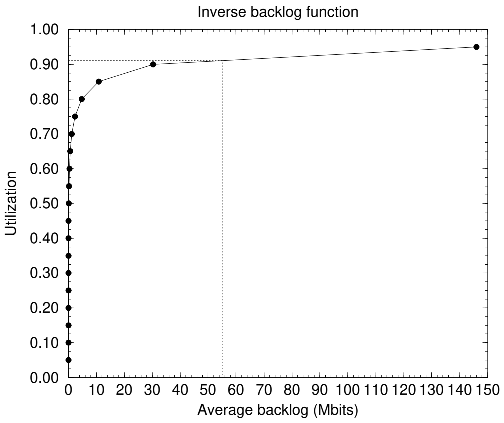
Suppose that a certain link offers N=4 classes, and that we are given an acceptable COP that the link has to meet:

$$\mathbf{v} = (40, 20, 10, 5)\text{msec} \quad \text{and} \quad \mathbf{h} = (0.5, 0.5, 2.0, 1.0)\text{kpps}$$

where *kpps* stands for 'kilo-packets-per-second'. If the average packet size is 1000 bytes, the total expected rate is $h = \sum_i h_i$=4kpps, or about 32Mbps. The problem is to determine the DDPs



(a) $q_{ag} = \beta(u)$



(b) $u = \beta^{-1}(q_{ag})$

Figure 49: The backlog and the inverse backlog functions for Pareto traffic with α=1.5.

and the minimum link capacity required to realize this COP.

From (5.19), the required DDPs are

$$\delta_2 = \frac{20}{40} = 0.5 \quad \delta_3 = \frac{10}{40} = 0.25 \quad \delta_4 = \frac{5}{40} = 0.125$$

With this optimal COP, the average backlog is

$$\bar{q}_{ag} = \sum_i v_i h_i = 40 \times 0.5 + 20 \times 0.5 + 10 \times 2.0 + 5 \times 1.0 = 55 \text{ packets}$$

We can now use the inverse backlog function $u = \beta^{-1}(\bar{q}_{ag})$ to compute the required utilization u. In this example, suppose that the average backlog function (and its inverse) are as in Figure 49 (these curves are generated from simulating Pareto interarrivals with α=1.5). For $\bar{q}_{ag}$=55 packets we find that the utilization is $u = \beta^{-1}(55)$ ≈92.0%, and so the capacity requirement is C=h/u=4444pps, or about 35.6Mbps.

Simulating the link with the previous DDPs and with u=92.0%, we get that the class average delays are $(\bar{d}_1, \bar{d}_2, \bar{d}_3, \bar{d}_4) = (35.1, 17.5, 8.8, 4.4)$msec, that are slighly less than the given maximum average delays specified in the given COP. Because the backlog curve is quite steep in the heavy load range, however, slight variations in the utilization or in the expected class rates can violate the average delay requirements. For example, if the utilization is increased to u=94.0%, the average class delays become $(\bar{d}_1, \bar{d}_2, \bar{d}_3, \bar{d}_4) = (68.2, 34.1, 17.1, 8.6)$msec, which violate the maximum average delay requirements. The large sensitivity of the capacity computation in the heavy load range implies that the network operator should use some tolerance in the computation of C. Even if the network provider provisions the link with a higher capacity than $\hat{C}$, in order to provide this tolerance and deal with uncertainties in the load estimates, it is still useful to know $\hat{C}$ as a *lower bound* on the required link capacity.

5.4.4 Other issues in class provisioning

An important requirement for computing the capacity requirement of a COP $\mathbf{v}$ is to know the average backlog $\bar{q}_{ag}$ as a function of the link utilization u. For simple queueing models, the function $\beta(u)$ is analytically known. For instance, in the $M|M|1$ system $\beta(u) = \frac{u^2}{1-u}$ packets, while in the $M|G|1$ system $\beta(u) = \frac{u^2}{1-u}\frac{1+c_L^2}{2}$, where c_L is the coefficient of variation of the packet size distribution [13]. In the more general $G|G|1$ system, the average backlog can be approximated by the Allen-Cunneen formula $\beta(u) \approx \frac{u^2}{1-u}\frac{c_A^2+c_L^2}{2}$ [1], where c_A is the coefficient of variation of the distribution of interarrivals. If the traffic interarrivals exhibit Long Range Dependency (LRD), c_A may not be well-defined (i.e., theoretically infinite) [58]. Recent measurement studies, however, have shown that in links with high flow aggregation, despite the fact that the traffic is still non-Poisson, the variance increases with the square root (i.e., sub-linearly) of the average bandwidth [71]. Such experimental results imply that even with the complex behavior of Internet traffic, tractable queueing models may still be applicable for an accurate estimation of statistical measures such as the average backlog in a PFE. A practical alternative, instead of relying on queueing models, is to *measure* the function $\beta(u)$ directly on the router, by monitoring the actual backlog in the PFE in different utilizations.

The backlog function may not only depend on the utilization u, but also on the capacity C. This can occur when the statistical properties of the traffic (burstiness) depend on C. For instance, with the same utilization, an OC-3 link (155Mbps) may have a larger backlog than a T-1 link (1.5Mbps), because higher-capacity links attract in general more bursty traffic. In a relatively narrow range of C though, we can assume that the traffic burstiness remains invariant, and that the backlog function depends on u but not on C.

The class provisioning methodology can be performed over relatively long timescales (say weeks or months), depending on how simple it is to adjust the link capacity. It is noted though that it gradually becomes simpler to adjust the capacity of a link even in a few minutes or seconds, through the use of Wavelength-Division-Multiplexing (WDM) and optical switches [85]. Using such technologies, an ISP can lease the capacity of an additional 'wavelength' from the backbone provider that owns the network fibers, when a larger traffic demand is anticipated or encountered. There are several backbone providers today that lease capacity in the range 1Mbps-1Gbps in increments of 1Mbps. Also, if the characterization of traffic types on a certain link follows different patterns through the day (e.g., many IP-telephony sessions through the day and mostly WWW sessions in the evening), the network operator can perform class provisioning for the different traffic patterns, and operate the link with a schedule of different capacities and DDPs during the day.

The proposed class provisioning methodology remains effective as long as the workload estimate is valid. If the traffic types have different average rates than what was assumed during provisioning, or if the average backlog function is not accurately known, the optimal average class delays will not be met. Similar problems can arise due to dynamic routing changes, link or router failures, or unexpected increases in the traffic demand. In such cases, only relative differentiation can be provided.

5.5 Related work on class selection and provisioning

A brief investigation of DCS in the context of proportional delay differentiation appeared in [72]. That work considered a class selection algorithm that routers can perform in order to provide a maximum delay to each flow. With a continuous-time model, and assuming a continuous range of class choices, [72] showed that when the set of user requirements is feasible, each flow converges to a class that provides the requested delay.

Orda and Shimkin considered multi-class networks with users that dynamically choose a class based on performance and pricing constraints [77]. The users select, in a distributed and selfish manner, the class that provides them with the maximum difference between utility and cost. The problem then is to compute the optimal class prices that will cause users to select the *nominal service class* that the network has provisioned for them. Even though there are some similarities between such an 'incentive pricing' framework and DCS, the problem formulation, the proposed mechanisms, and the final results are quire different. It is also likely that in practice the class prices would depend on marketing and competition, rather than on network provisioning mechanisms.

Chen and Park considered individual flows with absolute performance requirements in a stateless network [22]. The class selection is formulated as an optimization problem in which the overall resource usage cost is to be minimized, subject to the constraint that the performance requirement of each flow has to be met. Interestingly, there is a distributed algorithm that solves this problem.

Ren and Park considered the DiffServ model of per-hop priority mechanisms and traffic aggregation from individual flows to service classes [89]. [89] derived an optimal classifier (minimizing the resource usage in a mean-square sense) for mapping a flow to a class, subject to each flow's performance requirements. The model and results of [89] are based on game theory. Even though the DCS problem can be formulated as a non-cooperative game, with users that act independently and selfishly to meet their requirements given a finite resource, we chose in this work to take a different approach that only uses basic queueing concepts. The underlying concepts, though, are common in both approaches. For instance, the DCS equilibria are called *Nash equilibria* in game theoretic terms, while the minimum acceptable CSV $\hat{\mathbf{c}}$ can be shown to be Pareto and system optimal for the user population.

[89] assumed that the underlying per-hop mechanism is GPS scheduling with *periodic*

GPS weight adjustments. Such adjustments create a feedback loop that modifies the GPS weights based on the measured class loads. Changing the GPS weights, however, affects the per-class delays and losses, and in the DCS context, this leads to new class transitions and modified class loads. So, there is another feedback loop in the system that modifies the class loads based on the GPS weights. The previous two feedback loops can interact, causing races or instabilities, if they operate in about the same timescales. For this reason, we argue that the proportional differentiation model is a more appropriate per-hop behavior for DCS, as compared to GPS-like schedulers (or other static resource partitioning schemes) that perform periodic weight adjustments.

The rest of this section reviews some related works on the problem of class and QoS provisioning. Our class provisioning model follows the framework of [77]. In that framework, a number of traffic types with diverse QoS requirements are mapped to a normally smaller number of classes, while the network provider provisions a nominal service class for each traffic type. As mentioned earlier though, [77] uses this framework in the problem of computing of optimal class prices, while we focus on computing the optimal class differentiation parameters and the minimum link capacity.

[94] considers the problem of sharing the bandwidth and buffers of a link between a number of classes of service. The objective is to find the optimal resource partitioning between classes so that the *utility* of each class is maximized, given the 'wealth' of that class, and the average offered rate in that class. One of the utility functions considered is an upper bound on the loss rate in each class. A difference between that work and our class provisioning methodology is that [94] is based on static resource partitioning mechanisms (Weighted-Round-Robin scheduling and Complete-Buffer-Partitioning dropping), while we focus on proportional differentiation mechanisms.

As also discussed in §4.5, [119] studied the provisioning problem in the context of providing a certain loss rate to each class. [119] showed that the PLR(∞) dropper, jointly with a FCFS scheduler, is *optimal* because it requires *the minimum capacity for the given loss rates*, compared to any other work-conserving dropping scheme. It is interesting to examine if a proportional delay scheduler has the same property, i.e., whether it can provide a certain average delay in each class with the minimum capacity among all work-conserving schedulers.

5.6 Summary and extensions

A central premise in the framework of relative differentiation is that users and applications with an absolute QoS requirement can dynamically search for a class which provides the desired QoS level. In the first part of this chapter we investigated this Dynamic Class Selection (DCS) premise. The proportional differentiation model provides an appropriate per-hop behavior for DCS for two reasons. First, it maintains a predictable class ordering even when the class load distribution is constantly changing due to DCS traffic. Second, it allows the network provider to adjust the class spacing based on the corresponding spacing in the QoS requirements of the DCS workload.

For a single link model, we gave an algorithm that computes the minimum acceptable class selection for each user, when it is feasible to satisfy all users (well-provisioned case). Users converge to this minimum acceptable class when the distributed DCS equilibrium is unique. Other DCS equilibria can also exist, however, that are suboptimal for either only the users, or for both the users and the network provider. In the under-provisioned case, some users (with the most stringent requirements) converge to the highest class and remain unsatisfied there. The simulation study of an end-to-end DCS algorithm provided further insight into the dynamic behavior of DCS. We showed that a user can control the trade-off between the performance and cost of a flow using certain parameters of the DCS algorithm. Finally, it was demonstrated that the 'well-provisioning' of a network for a given traffic workload depends on the link utilization, on the delay requirements (or static class selection) of the cross-traffic, and on the provisioned delay differentiation between classes.

In the second part of this chapter, we studied the related problem of class provisioning. In class provisioning, the network provider has some knowledge about the traffic types that use a link (rates and average delay requirements). The objective is to compute the minimum required link capacity and the appropriate Delay Differentiation Parameters (DDPs) that can meet the given average delay for each traffic type. We gave a methodology that first determines the optimal mapping from traffic types to service classes, and then computes the minimum required capacity and the corresponding DDPs. The class provisioning methodology was demonstrated with examples.

There are several interesting open issues in the context of DCS and class provisioning.

- DCS should be evaluated in the context of other performance requirements, such as a maximum loss rate or a minimum TCP throughput. The dynamic properties of the proportional differentiation model also hold for these performance measures (see §4.1.2 for the loss rate dynamics in the PLD model), and so we expect similar DCS equilibrium results.

- Related to the class provisioning problem, it is important to examine whether static partitioning mechanisms (such as GPS) with periodic adjustments of the class partitions can actually cause race conditions and instability effects. We made this claim in §5.5, but we did not demonstrate that such effects actually occur in practice.

- A discouraging result about DCS is that, even when the network is well-provisioned, there may be DCS equilibria in which some users do not converge to the minimum acceptable class that exists for them. Even worse, the resulting DCS equilibria may be unacceptable. How can we improve the DCS algorithm to avoid such equilibria? We examined the following solution: when a DCS user is unsatisfied, she 'loops down' to the lowest class, instead of staying in the highest class. Such a class transition eliminates the unacceptable DCS equilibria (in the well-provisioned case), but it can lead to persistent CSV cycles.

  An important point, from a more practical perspective, is that the burstiness of the traffic and the variations in the queueing delays forces the DCS users to adapt their class selections in short timescales. Consequently, the concept of 'DCS equilibrium' is not well defined in short timescales. So, the stability and the efficiency of the resulting DCS equilibria may be largely of theoretical importance only.

Chapter 6

Summary and Future Work

6.1 Summary and retrospective

Our original goal in this dissertation was to develop a service differentiation architecture for the Internet that is scalable and simple to implement, deploy, and manage. The only way to prove, however, that an architecture meets these goals is to actually implement and use it in a 'real-world' wide-area internetwork, which is a task beyond our capabilities and constraints. So, we had to follow the conventional wisdom and the published literature about which components of a network architecture are not scalable, or hard to implement, deploy, and manage.

We chose to not follow the framework of per-flow end-to-end resource reservations that the IntServ architecture is based on, because of its non-scalability and complexity issues. Additionally, we chose to not follow the framework of absolute differentiation, that schemes such as the Virtual-Leased-Line or Assured service offer, because they also require some type of admission control and inter-domain resource reservations or careful provisioning.

The architecture that met our requirements for scalability and simplicity is the relative differentiation model. The only guarantee that the network provides in this case is that higher classes provide better service than lower classes. If users and applications have absolute QoS requirements, then they have to dynamically search for a class that provides that QoS. This model follows the end-to-end principle that the Internet architecture is based on: *keep the network as simple as possible and move the complexity to the end-points.*

The first question raised was which requirements should a relative differentiation scheme meet? Simple mechanisms, such as strict prioritization or static resource partitioning between classes, are not satisfactory either because they do not provide any 'knobs' for controlling the QoS spacing between classes, or because they do not maintain a predictable class ordering under varying load conditions. These two requirements, controllability and predictability, led us to the proportional differentiation model. The main strength of this model is that it can adjust the QoS ratio between two classes independent of the load conditions.

Given that our objective is to achieve proportional differentiation, the next goal was to design appropriate router mechanisms for the differentiation of the queueing delays and packet losses. We designed three packet schedulers for proportional delay differentiation. They differ in how closely they can approximate the proportional constraints, and in their behavior in short timescales. One of these schedulers, the Hybrid Proportional Delay (HPD) scheduler, performs sufficiently well in both these aspects, and it is the scheduling algorithm that we propose. Our study of the proportional delay differentiation model revealed that the model may not be always feasible. We derived the conditions that determine whether a set of Delay Differentiation Parameters (DDPs) is feasible in given load conditions.

We then focused on proportional loss differentiation and in the related packet dropping algorithms. We designed two droppers, that differ in the loss rate estimation horizon. This difference causes trade-offs between the two droppers in terms of their implementation complexity, accuracy, and adaptability to varying class load distributions. This last factor is quite important, leading us to propose the PLR(M) dropper as the appropriate algorithm for proportional loss differentiation. Our analytical study of the proportional loss differentiation model showed that this model may not be always feasible either, and we conjectured on the corresponding feasibility conditions.

A central premise in the relative differentiation architecture is that users with an absolute QoS requirement can dynamically search for a class which provides the desired QoS level. We investigated this Dynamic Class Selection (DCS) framework in the context of proportional delay differentiation. We examined whether it is feasible to satisfy all users, and if this is the case, computed the minimum acceptable class selection for each user. Users converge in a distributed manner to this minimum acceptable class, when the DCS equilibrium is unique. Other DCS equilibria, that are suboptimal for the users, may also exist however.

Finally, we considered the case that the network provider has some additional information about the traffic workload. Specifically, we assumed that the network provider knows the traffic types that use a link (rate and average delay requirement). When this is the case, the network provider can compute the minimum required link capacity and the appropriate DDPs for meeting the given average delay of each traffic type, based on a class provisioning methodology that we proposed.

In retrospective, the proportional differentiation architecture cannot offer the strict per-flow service guarantees that IntServ achieves. Also, the proposed architecture lacks mechanisms, such as admission control, that can prevent traffic from overloading the network. Additionally, the fact that the proportional differentiation model is not always feasible complicates the selection of the appropriate class differentiation parameters (DDPs and LDPs). Another discouraging result is that there can be suboptimal DCS equilibria, even when the network is well-provisioned.

On the other hand, this dissertation has shown that a relative differentiation architecture can provide controllable and predictable differentiation, when it follows the proportional differentiation model. Additionally, the proposed architecture can provide absolute QoS requirements to the users and applications that perform Dynamic Class Selection, and furthermore, it can be used to provision an absolute QoS level in each class.

Whether or not the proportional differentiation architecture is a satisfactory solution for providing service differentiation in the Internet depends on many factors, some of which are non-technical. We hope, however, that this dissertation has made a useful contribution to the range of possible solutions for this problem.

6.2 Suggestions for future work

Throughout the thesis, we pointed out specific issues that deserve further research. Here, we give two more suggestions for future work.

- A valuable experimental study would be to test an interactive application with absolute QoS requirements, such as IP-telephony, over a network that offers proportional delay and loss differentiation. Can the application meet its QoS requirements when it performs Dynamic Class Selection? How does the observed QoS compare to the QoS that results with a guaranteed bandwidth ATM virtual-circuit, or with some other 'IntServ'-like scheme?

- DCS can also be performed at the edge routers of a network. The end-points in that service model do not have to be DCS-capable, which is important for legacy applications. The edge routers can monitor the QoS level of each class in the various edge-to-edge network paths through active measurements. For instance, the packet dispersion methodologies studied in [35] can be used for bandwidth measurements. The users, in this model, negotiate with the network a certain edge-to-edge QoS. It is then up to the edge routers to dynamically select the class for the traffic of each user, in order to deliver the contracted QoS.

Chapter 7

Appendix

7.1 Simulation setup and parameters

In the evaluation of packet scheduling and dropping mechanisms, we simulated the following model (see Figure 50). A scheduler services N *logical* queues, one for each class. The capacity (service rate) of the scheduler is normalized to C=1 average-size packet per time unit. The utilization of the scheduler is $u = \lambda/C$, where λ is the average rate of packet arrivals. The class load distribution is specified in terms of packets. For example, when we say that Class-1 is 70% of the aggregate traffic, we mean that 70% of the packets belong to Class-1.

All queues share the same pool of buffers. In the lossless simulations, there is no constraint on the number of available buffers. In the lossy simulations, the number of buffers is limited to B. A packet requires one packet buffer, independent of the packet size. When the number of buffers is exceeded, *a packet is dropped from the tail of one of the backlogged queues*;

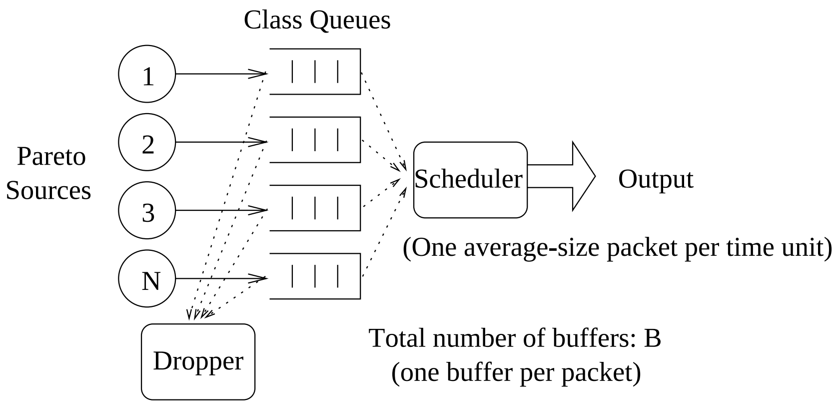


Figure 50: The simulation model for a Packet Forwarding Engine.

that queue is determined by the packet dropper.

The traffic in each class is generated from a random source. The interarrivals between packets of the same class follow a Pareto distribution with shape parameter $\alpha = 1.5$. The cumulative distribution function of this distribution is $F(x) = 1 - x^{-\alpha}$, the mean is $\alpha/(\alpha-1)$ (for $\alpha > 1$), and the variance is $\alpha/[(\alpha-1)^2(\alpha-2)]$ (for $\alpha > 2$). The distribution has infinite variance when $\alpha < 2$, leading to very bursty interarrivals. The aggregation of many Pareto sources with $\alpha < 2$ has been shown to produce Long Range Dependent (LRD) traffic [109].

We have experimented with several distributions for the packet sizes, including the exponential distribution, multimodal distributions around certain common values (e.g., 40, 550, and 1500 bytes), and also with fixed-size packets. In the case of variable-size packets, all classes have the same packet size distribution. In this dissertation, we only include simulation results for fixed-size packets; we did not observe any qualitative differences in the results when simulating variable-size packets.

The time unit in all graphs is the time it takes to transmit an average-size packet. The reported delays in all graphs are in these time units. In the case of fixed-size packets, when we say that the queueing delay of a packet is 10 units, we mean that the packet waited in the queue for the transmission of ten other packets.

7.2 Proofs of Properties (1)-(5) in §3.1.2

In the following proofs, $S = \sum_{n=1}^{N} \lambda_n \delta_n$.

Proof of Property (1).

Since the DDPs are positive, an increase in the average delay of one class causes an increase in the average delays of all classes. Since an increase in the aggregate input rate cannot cause a decrease in all class delays, we have that

$$\frac{\partial \bar{d}_i}{\partial \lambda_j} \geq 0 \qquad 1 \leq i, j \leq N \tag{7.1}$$

◇

Proof of Property (2).

From Equation (3.5) we have that

$$\frac{\partial \bar{d}_i}{\partial \lambda_j} - \frac{\partial \bar{d}_i}{\partial \lambda_k} = \frac{\delta_i}{S^2}\left[S\left(\frac{\partial \bar{q}_{ag}}{\partial \lambda_j} - \frac{\partial \bar{q}_{ag}}{\partial \lambda_k}\right) + \bar{q}_{ag}(\delta_k - \delta_j)\right] \tag{7.2}$$

The average backlog $\bar{q}_{ag}$, however, does not depend on the class that the traffic belongs to, and so $\partial \bar{q}_{ag}/\partial \lambda_j = \partial \bar{q}_{ag}/\partial \lambda_k$. If $k < j$, $\delta_k > \delta_j$, and so

$$\frac{\partial \bar{d}_i}{\partial \lambda_j} > \frac{\partial \bar{d}_i}{\partial \lambda_k} \qquad \text{when} \quad k < j \tag{7.3}$$

◇

Proof of Property (3).

From Equation (3.5) we have that

$$\frac{\partial \bar{d}_i}{\partial \delta_j} = \frac{-\bar{q}_{ag}\delta_i\lambda_j}{S^2} \leq 0 \qquad i \neq j \tag{7.4}$$

and

$$\frac{\partial \bar{d}_i}{\partial \delta_i} = \frac{\bar{q}_{ag}S - \bar{q}_{ag}\delta_i\lambda_i}{S^2} \geq 0 \tag{7.5}$$

The equalities are necessary for the case $\lambda_j = 0$ and $\lambda_i = \lambda$, respectively.

◇

Proof of Property (4).

Let λ'_k be the rate and $\bar{d}'_k$ be the average delay of a class k after the load transition from class i to class j. We have that $\lambda'_i = \lambda_i - \epsilon$ $(0 < \epsilon \leq \lambda_i)$, $\lambda'_j = \lambda_j + \epsilon$, and $\lambda'_k = \lambda_k$ for all $k \neq i, j$. So,

$$S' = \sum_{n=1}^{N} \delta_n \lambda'_n = S + \epsilon(\delta_j - \delta_i) \tag{7.6}$$

When $i > j$, $\delta_j > \delta_i$, and so $S' \geq S$. Note that $S' = S$ when $\lambda_i = \lambda'_j = \lambda$. Thus, from Equation (3.5), $\bar{d}'_k \leq \bar{d}_k$ for all $k = 1 \ldots N$. Similarly, when $i < j$, it follows that $\bar{d}'_k \geq \bar{d}_k$. Note that both load distributions have the same average aggregate backlog $\bar{q}_{ag}$.

◇

Proof of Property (5).

As in the proof of the previous property, $S' = S + \epsilon(\delta_j - \delta_i)$. Suppose that $i > j$ and thus $\delta_j > \delta_i$. It is easy to see that $\delta_j S \geq \delta_i S'$, because $S \geq \epsilon\delta_i$. Consequently,

$$\bar{d}'_j = \delta_j \bar{q}_{ag}/S' \geq \delta_i \bar{q}_{ag}/S = \bar{d}_i \tag{7.7}$$

Similarly, when $i < j$, it follows that $\bar{d}'_j \leq \bar{d}_i$.

7.3 Proof of Proposition 3.1 in §3.3

Kleinrock derived the average queueing delays with WTP [54] in the special case of Poisson arrivals. Assuming that all classes have the same average packet size, and normalizing the service rate to C=1, Kleinrock's result states that:

$$\bar{d}_i^{WTP} = \bar{d}_i = \frac{\bar{d}_0/(1-\lambda) - \sum_{k=1}^{i-1} \lambda_k \delta_k \, (1-\delta_i/\delta_k)}{1-\sum_{k=i+1}^{N}(1-\delta_k/\delta_i)} \qquad i = 1 \ldots N \tag{7.8}$$

where $\bar{d}_0$ is the average remaining service time for the packet that is being transmitted upon the arrival of a new packet. Recall that $\delta_1 = 1 > \delta_2 > \ldots > \delta_N > 0$. We prove Equation (3.17) using induction in the following manner: after showing that $\bar{d}_2/\bar{d}_1 = \delta_2$, we will assume that $\bar{d}_k/\bar{d}_1 = \delta_k$ for all $k = 2 \ldots m < N$ with $m \in \{2, \ldots, N-1\}$, and then show that $\bar{d}_{m+1}/\bar{d}_1 = \delta_{m+1}$.

For the initial induction step, it is easy to show from (7.8) that

$$\frac{\bar{d}_2}{\bar{d}_1} = \frac{1-\sum_{k=2}^{N} \lambda_k(1-\delta_k) - \lambda_1(1-\delta_2)}{1-\sum_{k=3}^{N} \lambda_k(1-\delta_k/\delta_2)} \tag{7.9}$$

and so,

$$\frac{\bar{d}_2}{\bar{d}_1} = \frac{(1-\lambda)+\sum_{k=1}^{N} \lambda_k\delta_k + \lambda_1(\delta_2-1)}{(1-\lambda)+\sum_{k=1}^{N} \lambda_k\delta_k/\delta_2 + \lambda_1(1-1/\delta_2)} \tag{7.10}$$

As $\lambda \to 1$, and thus, $u \to 100\%$,

$$\frac{\bar{d}_2}{\bar{d}_1} = \frac{\tilde{S}+\lambda_1(\delta_2-1)}{\tilde{S}/\delta_2+\lambda_1(1-1/\delta_2)} = \delta_2 \tag{7.11}$$

where $\tilde{S} = \lim_{\lambda\to 1} \sum_{k=1}^{N} \lambda_k \delta_k$. This completes the proof for N=2.

For N >2, the inductive assumption is that, as $\lambda \to 1$,

$$\frac{\bar{d}_k}{\bar{d}_1} \to \delta_k \qquad \text{for all } k = 2 \ldots m < N \tag{7.12}$$

with $m \in \{2, \ldots, N-1\}$. Then, the average delay of the $(m+1)$'th class is

$$\bar{d}_{m+1} = \frac{\bar{d}_0/(1-\lambda) - \sum_{k=1}^{m} \lambda_k \delta_k \bar{d}_1 (1 - \delta_{m+1}/\delta_k)}{1 - \sum_{k=m+2}^{N} \lambda_k (1 - \delta_k/\delta_{m+1})} \tag{7.13}$$

and the ratio of $\bar{d}_{m+1}$ and $\bar{d}_1$ becomes

$$\frac{\bar{d}_{m+1}}{\bar{d}_1} = \frac{1 - \sum_{k=1}^{N} \lambda_k (1-\delta_k) - \sum_{k=1}^{m} \lambda_k \delta_k (1 - \delta_{m+1}/\delta_k)}{1 - \sum_{k=1}^{N} \lambda_k (1 - \delta_k/\delta_{m+1}) + \sum_{k=1}^{m} \lambda_k (1 - \delta_k/\delta_{m+1})} \tag{7.14}$$

As $\lambda \to 1$, let $\tilde{S} = \lim_{\lambda \to 1} \sum_{k=1}^{N} \lambda_k \delta_k$, $\tilde{S}_m = \lim_{\lambda \to 1} \sum_{k=1}^{m} \lambda_k \delta_k$, and $\tilde{\lambda}_m = \lim_{\lambda \to 1} \sum_{k=1}^{m} \lambda_k$. Using this notation, we have that as $\lambda \to 1$,

$$\frac{\bar{d}_{m+1}}{\bar{d}_1} \to \frac{\tilde{S} - \tilde{S}_m + \delta_{m+1} \tilde{\lambda}_m}{\tilde{S}/\delta_{m+1} - \tilde{S}_m/\delta_{m+1} + \tilde{\lambda}_m} = \delta_{m+1} \tag{7.15}$$

which completes the proof.

7.4 Proof of Proposition 4.1 in §4.2

Suppose that the PLD constraints of (4.1) are met. If A_i and D_i are the number of arrivals and drops, respectively, in class i, the ratio of packet drops between any pair of classes i and j is

$$\frac{D_i}{D_j} = \frac{\sigma_i}{\sigma_j}\frac{A_i}{A_j} \tag{7.16}$$

Under stationary conditions, the ratio of arrivals is equal to the ratio of the corresponding arrival rates, and so,

$$\frac{D_i}{D_j} = \frac{\sigma_i}{\sigma_j}\frac{\lambda_i}{\lambda_j} \tag{7.17}$$

Consequently, the class with the minimum number of drops is class m, where

$$m = \arg\min_{1\leq i\leq N}\{\lambda_i\sigma_i\} \tag{7.18}$$

Let us now derive the minimum required number of arrivals, given a class load distribution $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ and an aggregate average loss rate $\bar{l}_{ag}$, so that the $N-1$ PLD constraints of (4.1) are met. The minimum number of arrivals causes the minimum number of drops. If class m, which has the minimum number of drops, gets only one packet drop ($D_m = 1$), class i will get

$$D_i = D_m\frac{\sigma_i}{\sigma_m}\frac{A_i}{A_m} = \frac{\sigma_i}{\sigma_m}\frac{\lambda_i}{\lambda_m} \tag{7.19}$$

drops, and so the total number of drops from all classes will be

$$D = \sum_{i=1}^{N} D_i = \sum_{i=1}^{N}\frac{\sigma_i}{\sigma_m}\frac{\lambda_i}{\lambda_m} \tag{7.20}$$

Since the aggregate loss rate is $\bar{l}_{ag}$, the required number of arrivals is

$$A = \frac{\sum_{i=1}^{N}\frac{\sigma_i}{\sigma_m}\frac{\lambda_i}{\lambda_m}}{\bar{l}_{ag}} \tag{7.21}$$

This is the minimum number of arrivals that the PLR(M) should remember, in order for the $N-1$

proportional loss rate constraints of (4.1) to hold. So, this is also the minimum requirement on the LHT size M_{min},

$$M \geq M_{min} = \frac{\sum_{i=1}^{N} \frac{\sigma_i}{\sigma_m} \frac{\lambda_i}{\lambda_m}}{\bar{l}_{ag}} \tag{7.22}$$

We emphasize that M_{min} is a lower bound on the required LHT size M. The reason is that the previous proof assumes that the class load distribution and the aggregate loss rate are given by their average values, namely $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ and $\bar{l}_{ag}$, *in any time window of M packet arrivals.* In practice though, the class load distribution and the aggregate loss rate vary significantly around their average values, even when M is many thousands of packets, due to the burstiness of the traffic arrivals and of the packet loss process.

7.5 Proofs of Lemmas in §5.2

Proof of Lemma 1.

Since $c_j \geq c'_j$ for all j (and $\mathbf{c} \neq \mathbf{c}'$), there exists a class $k \in \{1, \ldots N-1\}$ such that

$$\sum_{i=1}^{k} \lambda_i < \sum_{i=1}^{k} \lambda'_i = \sum_{i=1}^{k} \lambda_i + \epsilon \tag{7.23}$$

with $\epsilon > 0$, and

$$\sum_{i=k+1}^{N} \lambda_i > \sum_{i=k+1}^{N} \lambda'_i = \sum_{i=k+1}^{N} \lambda_i - \epsilon \tag{7.24}$$

Since $\delta_1 = 1 > \delta_2 > \ldots > \delta_N > 0$, we have that

$$\sum_{i=1}^{k} \delta_i \lambda'_i + \sum_{i=k+1}^{N} \delta_i \lambda'_i > \sum_{i=1}^{k} \delta_i \lambda_i + \epsilon\delta_k + \sum_{i=k+1}^{N} \delta_i \lambda_i - \epsilon\delta_{k+1} \tag{7.25}$$

and thus

$$\sum_{i=1}^{N} \delta_i \lambda'_i > \sum_{i=1}^{N} \delta_i \lambda_i + \epsilon(\delta_k - \delta_{k+1}) > \sum_{i=1}^{N} \delta_i \lambda_i \tag{7.26}$$

It follows that the average delay in each class with $\mathbf{c}'$ is lower (in the wide sense) than with $\mathbf{c}$, because

$$\bar{d}_i(\mathbf{c}') = \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda'_n} \leq \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda_n} = \bar{d}_i(\mathbf{c}) \tag{7.27}$$

$\diamond$

Proof of Lemma 2.

The proof of this result follows directly from Property 5 in §3.1.2.

$\diamond$

Proof of Lemma 3.

Suppose that $\mathbf{c}$ is such that $c_i > c_j$ for two users i and $j > i$ (so, $\phi_j \leq \phi_i$). Let us construct the

CSV $\mathbf{c}' = \mathbf{c}\rangle_i^k$, which results if user i moves to the same class $k = c_j$ as user j. Since $\mathbf{c} \geq \mathbf{c}'$, from Lemma 1 we have that $\bar{d}_k(\mathbf{c}') = \bar{d}_k(\mathbf{c}') \leq \bar{d}_k(\mathbf{c}) \leq \phi_j \leq \phi_i$, which means that user i is satisfied with $\mathbf{c}'$. The rest of the users are also satisfied with $\mathbf{c}'$ because the average delay in each class is lower with $\mathbf{c}'$ than with $\mathbf{c}$. So, $\mathbf{c}'$ is also acceptable.

Applying the above procedure iteratively, we can construct an acceptable CSV $\mathbf{c}'$ that is ordered, in the sense that for any $i, j \in \{1, \ldots U\}$, $c'_i \leq c'_j$ when $i < j$. The order of applying the above iteration does not affect the ordered CSV that results at the end. From the way $\mathbf{c}'$ is constructed, we have that

$$c'_j = \min_{k=j,\ldots,U} \{c_k\}, \qquad j = 1, \ldots, U \tag{7.28}$$

$\diamond$

Proof of Lemma 4.

From the definition of $\mathbf{c^m}$, we have that $\mathbf{c}^1 \geq \mathbf{c}^m$. If these two CSVs are equal, the proof is complete. Otherwise, from Lemma 1, we have that $\bar{d}_i(\mathbf{c}^m) \leq \bar{d}_i(\mathbf{c}^1)$ for each class i. Similarly, $\bar{d}_i(\mathbf{c}^m) \leq \bar{d}_i(\mathbf{c}^2)$. For each user $j \in \mathcal{U}$, however, $\bar{d}_{c_j^1}(\mathbf{c}^1) \leq \phi_j$ and $\bar{d}_{c_j^2}(\mathbf{c}^2) \leq \phi_j$. So, $\bar{d}_{c_j^m}(\mathbf{c}^m) \leq \phi_j$. Since this is true for all users, $\mathbf{c}^m \in C_A$.

$\diamond$

Proof of Lemma 5.

Based on the DCS algorithm, all unsatisfied users move to the N'th class, while the rest of the users are satisfied in a class, which may also be class N. Let $\tilde{\mathbf{c}}$ be the corresponding distributed equilibrium CSV. We prove that *any satisfied user must have a larger average delay requirement than any unsatisfied user.*

For any unsatisfied user, say j, we have that $\tilde{c}_j = N$ and that $\bar{d}_{\tilde{c}_j}(\tilde{\mathbf{c}}) > \phi_j$. If there are no satisfied users, there is nothing to prove. Let i be a satisfied user in a class $\tilde{c}_i \leq N$. Suppose that user i has a lower or equal average delay requirement than user j, i.e., $\phi_i \leq \phi_j$. User i is satisfied, and so $\bar{d}_{\tilde{c}_i}(\tilde{\mathbf{c}}) \leq \phi_i \leq \phi_j$. So,

$$\bar{d}_{\tilde{c}_i}(\tilde{\mathbf{c}}) < \bar{d}_{\tilde{c}_j}(\tilde{\mathbf{c}}) \tag{7.29}$$

Since $\tilde{c}_i \leq \tilde{c}_j$, however, the PDD model requires that $\bar{d}_{\tilde{c}_i}(\tilde{\mathbf{c}}) \geq \bar{d}_{\tilde{c}_j}(\tilde{\mathbf{c}})$ which contradicts (7.29). So, it must be that $\phi_i > \phi_j$ for any satisfied user i and unsatisfied user j. Since the satisfied users have larger delay requirements than the unsatisfied users, the resulting DCS equilibrium must be of the form

$$\tilde{\mathbf{c}} = (c_1, \ldots, c_S, N, \ldots, N) \tag{7.30}$$

where S is the number of satisfied users $(0 \leq S < U)$.

Bibliography

[1] O. Allen, *Probability, Statistics, and Queueing Theory with Computer Science Applications*, Academic Press, 2nd edition, 1990.

[2] ATM Forum. *ATM Traffic Management Specification, Version 4.0*, April 1996. Available at ftp://ftp.atmforum.com/pub.

[3] F. Baker, R. Guerin, and D. Kandlur. *Specification of Committed Rate Quality of Service*, June 1996. draft-ietf-intserv-commit-rate-svc-00.txt.

[4] A. Banchs and R. Denda, "A Scalable Share Differentiation Architecture for Elastic and Real-Time Traffic," In *IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, June 2000.

[5] A. Banerjea, D. Ferrari, B. A. Mah, M. Moran, D. C. Verma, and H. Zhang, "The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences," *IEEE/ACM Transactions on Networking*, vol. 4, no. 1, pp. 1–10, February 1996.

[6] A. Begel, S.McCanne, and S.L.Graham, "BPF+: Exploiting Global Data-flow Optimization in a Generalized Packet Filter Architecture," In *Proceedings ACM SIGCOMM*, September 1999.

[7] J. Bennett and H.Zhang, "Hierarchical Packet Fair Queueing Algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, October 1997.

[8] N. Bhatti and R.Friedrich, "Web Server Support for Tiered Services," *IEEE Network*, pp. 64–71, September 1999.

[9] S. Blake, D.Black, M.Carlson, E.Davies, Z.Wang, and W.Weiss. *An Architecture for Differentiated Services*, December 1998. IETF RFC 2475.

[10] S. Bodamer, "A Scheduling Algorithm for Relative Delay Differentiation," In *Proceedings of the IEEE Conference on High Performance Switching and Routing (ATM 2000)*, June 2000.

[11] U. Bodin, A. Jonsson, and O. Schelen, "On Creating Proportional Loss Differentiation: Predictability and Performance," Technical report, Department of Computer Science and Electrical Engineering, Lulea University of Technology, Sweden, July 2000.

[12] U. Bodin, O. Schelen, and S. Pink, "Load-Tolerant Differentiation with Active Queue Management," *ACM Computer Communication Review (CCR)*, July 2000.

[13] G. Bolch, S.Greiner, H.Meer, and K.S.Trivedi, *Queueing Networks and Markov Chains*, John Wiley and Sons, 1999.

[14] J. L. Boudec, M. Hamdi, L. Blazevic, and P.Thiran, "Asymmetric Best Effort Service for Packet Networks," In *Proceedings Global Internet Symposium*, December 1999.

[15] R. Braden, D.Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview*, July 1994. RFC 1633.

[16] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource ReSerVation Protocol (RSVP) - Version 1, Functional Specificication*, September 1997. RFC 2205.

[17] R. Caceres, N.Duffield, and A.Feldmann, "Measurement and Analysis of IP Network Usage and Behavior," *IEEE Communications Magazine*, pp. 144–152, May 2000.

[18] C. Cetinkaya and E.W.Knightly, "Egress Admission Control," In *Proceedings INFOCOM*, March 2000.

[19] H. Chao, H.Cheng, Y-R.Jeng, and D.Jeong, "Design of a Generalized Priority Queue Manager for ATM Switches," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 5, pp. 867–879, June 1997.

[20] H. Chao and N.Uzun, "A VLSI Sequencer Chip for ATM Traffic Shaper and Queue Manager," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1634–1643, November 1992.

[21] A. Charny and J. L. Boudec, "Delay Bounds in a Network with Aggregate Scheduling," In *Proceedings QOFIS*, October 2000.

[22] S. Chen and K. Park, "An Architecture for Noncooperative QoS Provision in Many-Switch Systems," In *Proceedings IEEE INFOCOM*, 1999.

[23] A. Choudhury and E.L.Hahne, "Dynamic Queue Length Thresholds for Multipriority Traffic," In *15th International Teletraffic Congress*, June 1997.

[24] D. D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362–373, August 1998.

[25] D. Clark, "Adding Service Discrimination to the Internet," In *Internet Economics*, L.W.McKnight and J.P.Bailey, editors, The MIT Press, 1997.

[26] E. Coffman and I.Mitrani, "A Characterization of Waiting Time Performance Realizable by Single-Server Queues," *Operations Research*, vol. 28, no. 3, pp. 810–821, May 1980.

[27] R. L. Cruz, "A Calculus for Network Delay, Part I: Network Elements in Isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, January 1991.

[28] R. L. Cruz, "A Calculus for Network Delay, Part II: Network Analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132–141, January 1991.

[29] A. Demers, S.Keshav, and S.Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," In *Internetworking: Research and Experience*, pp. 3–26, 1990.

[30] C. Dovrolis and D.Stiliadis, "Relative Differentiated Services in the Internet: Issues and Mechanisms," In *ACM SIGMETRICS*, May 1999. Extended abstract.

[31] C. Dovrolis, D.Stiliadis, and P.Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," In *ACM SIGCOMM*, September 1999.

[32] C. Dovrolis and P.Ramanathan, "RAFT: Resource Aggregation for Fault Tolerance in Integrated Services Packet Networks," *ACM Computer Communication Review (CCR)*, April 1998.

[33] C. Dovrolis and P.Ramanathan, "A Case for Relative Differentiated Services and the Proportional Differentiation Model," *IEEE Network*, October 1999.

[34] C. Dovrolis and P.Ramanathan, "Proportional Differentiated Services, Part II: Loss Rate Differentiation and Packet Dropping," In *IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, June 2000.

[35] C. Dovrolis, P. Ramanathan, and D. Moore, "What do Packet Dispersion Techniques Measure?," In *Proceedings of IEEE INFOCOM*, April 2001.

[36] C. Dovrolis, D. Tull, and P. Ramanathan, "Hybrid Spatial/Temporal Loss Concealment for Packet Video," In *9th International Packet Video Workshop*, May 1999.

[37] R. Edell and P.Varaiya, "Providing Internet Access: What we Learn from INDEX," *IEEE Network*, pp. 18–25, September 1999.

[38] A. Enwalid, D.Heyman, T.V.Lakshman, D.Mitra, and A.Weiss, "Fundamental Bounds and Approximations for ATM Multiplexers with Applications to Video Teleconferencing," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1004–1016, August 1995.

[39] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "A Self-Configuring RED Gateway," In *Proceedings IEEE INFOCOM*, April 1999.

[40] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Adaptive Packet Marking for Maintaining End-to-End Throughput in a Differentiated-Services Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 685–697, October 1999.

[41] P. Ferguson and G.Huston, *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*, John Wiley and Sons, January 1998.

[42] T. Ferrari and P.F.Chimento, "A Measurement-based Analysis of Expedited Forwarding PHB Mechanisms," In *Proceedings International Workshop on QoS (IWQoS)*, June 2000.

[43] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[44] S. Floyd and V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, August 1995.

[45] R. Gopalakrishnan and G.M.Parulkar, "Efficient User-Space Protocol Implementations with QoS Guarantees Using Real-Time Upcalls," *IEEE Transactions on Networking*, vol. 6, no. 4, pp. 374–388, August 1998.

[46] R. Guerin, S. Kamat, and S. Herzog. *QoS Path Management with RSVP*, March 1997. Internet Draft: draft-qos-path-mgmt-rsvp-00.txt.

[47] R. Guerin and V.Pla, "Aggregation and Conformance in Differentiated Service Networks: A Case Study," In *Proceedings ITC Specialist Seminar on IP Traffic Modeling, Measurement, and Management*, September 2000.

[48] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. *Assured Forwarding PHB Group*, June 1999. RFC 2597.

[49] IETF. *Internet Engineering Task Force.* www.ietf.org.

[50] V. Jacobson, "Congestion Avoidance and Control," In *Proceedings ACM SIGCOMM*, pp. 314–329, September 1988.

[51] V. Jacobson, K. Nichols, and K.Poduri. *An Expedited Forwarding PHB*, June 1999. RFC 2598.

[52] S. Jamin, P. B. Danzig, S. J. Shenker, and L. Zhang, "A Measurement-Based Admission Control Algorithm for Integrated Service Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 56–70, February 1997.

[53] S. Keshav and R.Sharma, "Issues and Trends in Router Design," *IEEE Communications Magazine*, pp. 144–151, May 1998.

[54] L. Kleinrock, *Queueing Systems, Volume II*, John Wiley and Sons, 1976.

[55] H. Kroner, G.Hebuterne, P.Boyer, and A.Gravey, "Priority Management in ATM Switching Nodes," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 3, pp. 418–427, April 1991.

[56] V. Kumar, T.V.Lakshman, and D.Stiliadis, "Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet," *IEEE Communications Magazine*, pp. 152–164, May 1998.

[57] T. V. Lakshman and D.Stiliadis, "High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching," In *Proceedings ACM SIGCOMM*, 1998.

[58] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, February 1994.

[59] M. K. H. Leung, J. Lui, and D. Yau, "Characterization and Performance Evaluation for Proportional Delay Differentiated Services," In *Proceedings International Conference on Network Protocols (ICNP)*, October 2000.

[60] C. C. Li, S.-L. Tsao, M. C. Chen, Y. Sun, and Y.-M. Huang, "Proportional Delay Differentiation Service Based on Weighted Fair Queueing," In *Proceedings IEEE International Conference on Computer Communications and Networks (ICCCN)*, October 2000.

[61] J. Liebeherr and N.Christin, "Buffer Management and Scheduling for Enhanced Differentiated Services," Technical Report CS-2000-24, University of Virginia, August 2000.

[62] J. Liebeherr, D. E. Wrege, and D. Ferrari, "Exact Admission Control for Networks with a Bounded Delay Service," *IEEE/ACM Transactions on Networking*, vol. 4, no. 6, pp. 885–901, December 1996.

[63] A. Y.-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*, vol. 9, no. 9, pp. 1524–1536, December 1991.

[64] A. Mankin, F.Baker, B.Braden, S.Bradner, M.O'Dell, A.Romanow, A.Weinrib, and L.Zhang. *RSVP Version 1: Applicability Statement, Some Guidelines on Deployment*, September 1997. IETF RFC 2208.

[65] M. Mathis, J. Semke, and J. Madhavi, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM Computer Communications Review*, vol. 27, no. 3, pp. 67–82, July 1997.

[66] M. May, J.C.Bolot, C.Diot, and A.Jean-Marie, "Simple Performance Models of Differentiated Services Schemes for the Internet," In *Proceedings IEEE INFOCOM*, March 1999.

[67] D. L. Mills, "The Fuzzball," In *Proceedings ACM SIGCOMM*, pp. 115–122, August 1988.

[68] V. Misra, W. B. Gong, and D. Towsley, "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," In *Proceedings ACM SIGCOMM*, September 2000.

[69] I. Mitrani and J.H.Hine, "Complete Parameterized Families of Job Scheduling Strategies," *Acta Informatica*, vol. 8, pp. 61–73, 1977.

[70] Y. Moret and S.Fdida, "A Proportional Queue Control Mechanism to Provide Differentiated Services," In *International Symposium on Computer and Information Systems (ISCIS)*, October 1998.

[71] R. Morris and D. Lin, "Variance of Aggregated Web Traffic," In *Proceedings IEEE INFOCOM*, April 2000.

[72] T. Nandagopal, N.Venkitaraman, R.Sivakumar, and V.Bharghavan, "Delay Differentiation and Adaptation in Core Stateless Networks," In *Proceedings IEEE INFOCOM*, March 2000.

[73] A. Neogi, T.Chiueh, and P.Stirpe, "Performance Analysis of an RSVP-Capable Router," *IEEE Network*, pp. 56–63, September 1999.

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

[75] K. Nichols, V.Jacobson, and L.Zhang. *A Two-Bit Differentiated Services Architecture for the Internet*, July 1999. IETF RFC 2638.

[76] A. M. Odlyzko, "Paris Metro Pricing: The Minimalist Differentiated Services Solution," In *Proceedings IEEE/IFIP International Workshop on Quality of Service*, June 1999.

[77] A. Orda and N. Shimkin, "Incentive Pricing in Multi-Class Communication Networks," In *Proceedings IEEE INFOCOM*, 1997.

[78] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: Stabilized RED," In *Proceedings IEEE INFOCOM*, April 1999.

[79] A. K. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, PhD thesis, Massachusetts Institute of Technology, 1992. LIDS-TH-2089.

[80] C. Partridge, *Gigabit Networking*, Addison-Wesley, 1994.

[81] V. Paxson, "End-to-End Routing Behavior in the Internet," In *Proceedings SIGCOMM Symposium*, pp. 25–38, August 1996.

[82] V. Paxson and S.Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, June 1995.

[83] M. Podolsky, C. Romer, and S. McCanne, "Simulation of FEC-Based Error Control for Packet Audio on the Internet," In *Proceedings IEEE INFOCOM*, April 1998.

[84] J. Postel. *Internet Protocol*, September 1981. IETF RFC 791.

[85] B. Rajagopalan, D. Pendarakis, D. Saha, R. Ramamoorthy, and K. Bala, "IP over Optical Networks: Architectural Aspects," *IEEE Communications Magazine*, pp. 94–102, September 2000.

[86] R. Rajan, D.Verma, S.Kamat, E.Felstaine, and S.Herzog, "A Policy Framework for Integrated and Differentiated Services in the Internet," *IEEE Network*, pp. 36–41, September 1999.

[87] R. Ramjee, J.Kurose, D.Towsley, and H.Schulzrinne, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," In *Proceedings IEEE INFOCOM*, pp. 680–688, 1994.

[88] J. Regnier, "Priority Assignment in Integrated Services Networks," Technical Report LIDS-TH-1565, LIDS-MIT, December 1986.

[89] H. Ren and K. Park, "Toward a Theory of Differentiated Services," In *Proceedings IWQoS*, June 2000.

[90] B. Reynolds, "RED Analysis for Congested Network Core and Customer Egress," Technical report, QualNet, January 1999. North American Network Operators' Group (NANOG) meeting.

[91] I. Rhee, "Error Control Techniques for Interactive Low-Bit Rate Video Transmission over the Internet," In *Proceedings ACM SIGCOMM*, 1998.

[92] J. S. Sahu, D.Towsley, "A Quantitative Study of Differentiated Services for the Internet," In *Proceedings Global Internet Symposium*, December 1999.

[93] S. Sahu, P.Nain, D.Towsley, C.Diot, and V.Firoiu, "On Achievable Service Differentiation with Token Bucket Marking for TCP," In *Proceedings ACM SIGMETRICS*, June 2000.

[94] J. Sairamesh, D. F. Ferguson, and Y. Yemini, "An Approach to Pricing, Optimal Allocation and Quality of Service Provisioning in High-Speed Packet Networks," In *Proceedings IEEE INFOCOM*, pp. 1111–1119, 1995.

[95] H. Saito, C. Lukovszki, and I. Moldovan, "Local Optimal Proportional Differentiated Services Scheduler for Relative Differentiated Services," In *Proceedings IEEE International Conference on Computer Communications and Networks (ICCCN)*, November 2000.

[96] H. Schulzrinne, S.Casner, R.Frederick, and V.Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, January 1996. RFC 1889.

[97] U. Schwantag, "An analysis of the applicability of RSVP," Technical report, Institute of Telematics, University of Karlsruhe, July 1997.

[98] S. Shenker and J. Wroclawski. *General Characterization Parameters for Integrated Service Network Elements*, September 1997. RFC 2215.

[99] M. Shreedhar and G.Varghese, "Efficient Fair Queuing using Deficit Round Robin," In *Proceedings ACM SIGCOMM*, pp. 231–242, 1995.

[100] V. Srinivasan, S.Suri, and G.Varghese, "Packet Classification using Tuple Space Search," In *Proceedings ACM SIGCOMM*, September 1999.

[101] D. C. Stephens, J.C.R.Bennett, and H.Zhang, "Implementing Scheduling Algorthms in High-Speed Networks," *IEEE Journal on Selected Areas of Communications*, September 1999.

[102] D. Stiliadis and A.Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 611–625, October 1998.

[103] D. Stiliadis and A.Varma, "Rate-Proportional Servers: A Design Methodology for Fair Queueing Algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 164–174, April 1998.

[104] I. Stoika and H.Zhang, "LIRA: An Approach for Service Differentiation in the Internet," In *Proceedings NOSSDAV*, 1998.

[105] I. Stoika and H.Zhang, "Providing Guaranteed Services Without Per Flow Management," In *Proceedings ACM SIGCOMM*, September 1999.

[106] I. Stoika, S.Shenker, and H.Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," In *Proceedings ACM SIGCOMM*, September 1998.

[107] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury, "Design Considerations for Supporting TCP with Per-Flow Queueing," In *Proceedings IEEE INFOCOM*, 1998.

[108] B. Suter, T.V.Lakshman, D.Stiliadis, and A.K.Choudhury, "Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-Flow Queueuing," *IEEE Journal on Selected Areas of Communications*, September 1999.

[109] M. S. Taqqu, W.Willinger, and R.Sherman, "Proof of a Fundamental Result in Self-Similar Traffic Modeling," *ACM Computer Communications Review*, pp. 5–23, April 1997.

[110] B. Teitelbaum, S.Hares, L.Dunn, R.Neilson, V.Narayan, and F.Reichmeyer, "Internet2 QBone: Building a Testbed for Differentiated Services," *IEEE Network*, pp. 8–16, September 1999.

[111] A. Terzis, L.Wang, J.Ogawa, and L.Zhang, "A Two-Tier Resource Management Model for the Internet," In *Proceedings Global Internet Symposium*, December 1999.

[112] K. Thompson, G. J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," *IEEE Network*, pp. 10–23, November 1997.

[113] A. Viswanathan, N. Feldman, Z. Wang, and R. Callon, "Evolution of Multiprotocol Label Switching," *IEEE Communications Magazine*, May 1998.

[114] Z. Wang, "A Case for Proportional Fair Sharing," In *International Workshop on QoS*, May 1998.

[115] P. P. White, "RSVP and Integrated Services in the Internet: A Tutorial," *IEEE Communications Magazine*, pp. 100–106, May 1997.

[116] W. Willinger, M.S.Taqqu, R.Sherman, and D.V.Wilson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," In *Proceedings SIGCOMM Symposium*, pp. 100–113, September 1995.

[117] J. Wroclawski. *Specification of the Controlled-Load Network Element Service*, September 1997. RFC 2211.

[118] J. Wroclawski. *The Use of RSVP with IETF Integrated Services*, September 1997. RFC 2210.

[119] T. Yang and J.Pan, "A Measurement-Based Loss Scheduling Scheme," In *Proceedings INFOCOM*, 1996.

[120] D. K. Y. Yau and S.S.Lam, "Adaptive Rate-Controlled Scheduling for Multimedia Applications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 4, pp. 475–488, August 1997.

[121] I. Yeom and A. N. Reddy, "Modeling TCP Behavior in a Differentiated Services Network," Technical report, Texas A&M University, February 2000.

[122] N. Yin and M.G.Hluchyj, "Implication of Dropping Packets from the Front of a Queue," *IEEE Transactions on Communications*, vol. 41, no. 6, pp. 846–851, June 1993.

[123] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource Reservation Protocol," *IEEE Network*, pp. 8–18, September 1993.