UNIVERZITET U BEOGRADU

ELEKTROTEHNIČKI FAKULTET

VLADIMIR V. VUKADINOVIĆ

MAGISTARSKA TEZA

FLEKSIBILNI INTERNET SERVISI NA BAZI KONTROLE KAŠNJENJA I PROPUSNOSTI

Teza je odbranjena 15. oktobra 2004. godine pred komisijom u sastavu:

Prof. Grozdan Petrović, mentor
Elektrotehnički fakultet, Univerzitet u Beogradu
Srbija i Crna Gora

Prof. Ljiljana Trajković, mentor
School of Engineering Science, Simon Fraser University
Canada

Prof. Zoran Petrović
Elektrotehnički fakultet, Univerzitet u Beogradu
Srbija i Crna Gora

Prof. Zoran Bojković
Saobraćajni fakultet, Univerzitet u Beogradu
Srbija i Crna Gora

Beograd, oktobar 2004. godine

Abstrakt

Vladimir V. Vukadinović

Fleksibilni Internet servisi na bazi kontrole kašnjenja i propusnosti

Internet je transportna infrastruktura za aplikacije sa različitim zahtevima u pogledu kvaliteta servisa. Današnji Internet je međutim samo "best-effort" mreža bez implementiranih mehanizama za diferencijaciju servisa. Skorašnji napori da se obezbede takvi mehanizmi usmereni su ka takozvanim fleksibilnim servisima. Većina predloženih arhitektura za pružanje fleksibilnih servisa se bazira na principu pružanju servisa malog kašnjenja po cenu povećanja verovatnoće gubitka paketa. Ove arhitekture međutim ne uzimaju u obzir uticaj diferencijacije kašnjenja na performanse TCP-a, najčešće korišćenog transportnog protokola na Internetu. Prilikom projektovanja mehanizama za diferencijaciju servisa, neophodno je uzeti u obzir kompleksnost TCP algoritma za kontrolu zagušenja. U ovoj tezi pretstavljen je nova arhitektura za diferencijaciju kašnjenja i propusnosti zasnovana na konceptu fleksibilnih servisa. Neophodni uslovi za ostvarivanje željene diferencijacije servisa između razlicitih klasa mrežnog saobraćaja izvedeni su na osnovu analize kontrolnog sistema sa povratnom spregom koga čine TCP algoritam za kontrolu zagušenja i mehanizmi za diferencijaciju kašnjenja i kontrolu verovatnoće gubitaka.

Ključne reči—*diferencijacija servisa, fleksibilni servisi, QoS, aktivna kontrola bafera, TCP, mrežni saobraćaj.*

Abstract

Vladimir V. Vukadinović

Delay and throughput differentiation mechanisms for non-elevated services

Internet is a transport infrastructure intended for applications with various service requirements. However, Internet remains to be a best-effort network without widely deployed mechanisms for service differentiation and quality of service (QoS) provisioning. Research efforts to provide service differentiation in the Internet have been recently directed toward non-elevated services. Majority of proposed non-elevated mechanisms aim to provide low delay service at the expense of increased packet loss probability. However, these proposals do not consider the influence of delay and loss differentiation on the behaviour of TCP, the most widely used transport protocol in today's Internet. Service differentiation mechanisms cannot be properly designed without taking into account the complexity of TCP's congestion control algorithm. In this thesis, we propose a new non-elevated service differentiation architecture and derive necessary conditions to provide desired service differentiation among traffic classes by considering TCP's congestion control algorithm, delay differentiation mechanisms, and proposed packet loss controller as a feedback control system.

Keywords—*non-elevated services, service differentiation, TCP, QoS, queue management.*

Acknowledgements

First and foremost I would like to thank my academic advisors, Dr. Ljiljana Trajkovic and Dr. Grozdan Petrovic.

Dr. Ljiljana Trajkovic deserves my deepest gratitude and respect for her continued support during the writing of this thesis. She gave me a wonderful opportunity to spend a memorable time at Simon Fraser University, Vancouver, Canada. Special thanks for teaching me how to write academic papers and encouraging me to continue my studies.

Dr. Grozdan Petrovic has been providing me continuous support for many years, both during my undergraduate and graduate studies. He showed me different ways to approach a research problem and the need to be persistent to accomplish any goal.

I would also like to thank all the members of Communication Networks Laboratory, especially my dear friends Svetlana and Bozidar Vujicic, Nikola Cackov, Nenad Laskovic, Hui (Grace) Zhang, and Hao (Johnson) Chen for their help and companionship.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

ABE	Alternative Best-Effort
AF	Assured Forwarding
AQM	Active Queue Management
BEDS	Best-Effort Differentiated Services
BPR	Backlog-Proportional Rate
BPR^{+}	Optimized Backlog-Proportional Rate
CAC	Call Admission Control
CLS	Controlled Load Service
DiffServ	Differentiated Services
DS	Delay-Sensitive
DSD	Duplicate Scheduling with Deadlines
DSCP	DiffServ Codepoint
EDS	Equivalent Differentiated Services
EF	Expedited Forwarding
FIFO	First-In First-Out
FTP	File Transfer Protocol
GS	Guaranteed Service
HOL	Head-of-Line
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force

IntServ	Integrated Services
IP	Internet Protocol
ISP	Internet Service Provider
LHT	Loss History Table
NNTP	Network News Transfer Protocol
PLC	Packet Loss Controller
PHB	Per-Hop Behaviour
PLR	Proportional Loss Rate
QoS	Quality of Service
RED	Random Early Detection
RSVP	Resource Reservation Protocol
SMTP	Simple Mail Transfer Protocol
SLA	Service Level Agreement
TCA	Traffic Conditioning Agreement
TCP	Transport Control Protocol
TDP	Time-Dependent Priorities
TS	Throughput-Sensitive
UDP	User Datagram Protocol
WFQ	Weighted Fair Queuing
WTP	Waiting-Time Priority

1. Introduction

Current Internet applications have diverse service requirements. Real time applications, such as IP telephony and video on demand, require low delay and delay jitter while being relatively insensitive to packet losses. To the contrary, applications dealing with bulk data transfer, such as File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), and Network News Transfer Protocol (NNTP), require sufficient throughput and reliability while being tolerant to network delay. Current Internet is a best-effort network and offers no guarantees with respect to delay, throughput, and loss probability [1], [2]. All packets are treated equally, regardless of their service requirements. Numerous research efforts have been devoted recently to providing distinct levels of service to applications. As a result, two paradigms for quality of service (QoS) provisioning emerged in the 90's: Integrated Services (IntServ) and Differentiated Services (DiffServ). More recent approaches [3]-[5] are based on the concept of non-elevated services.

The IntServ model [6], [7] provides absolute end-to-end QoS guarantees for throughput, packet delay, and packet loss probability. It is focused on individual flows because each flow in IntServ is able to request a specific level of service from the network. However, IntServ suffers from several major drawbacks. It is not scalable because each router has to maintain per-flow states. Routers in the core of the Internet are usually traversed by thousands of traffic flows. Furthermore, implementation complexity of IntServ is substantial because IntServ requires major changes to management and accounting policies currently deployed in the Internet.

The DiffServ model [8]-[10] offers per-hop QoS guarantees to flow aggregates called classes, as opposed to end-to-end per-flow guarantees offered by IntServ. Since the large number of traffic flows in the network core does not permit implementation of complex QoS mechanisms, DiffServ model bundles flows with similar QoS requirements into traffic classes. DiffServ model is more scalable than IntServ because routers maintain only per-class states. Furthermore, network management in DiffServ is similar to the management currently employed in IP networks. However, DiffServ cannot be deployed incrementally because it requires all-or-nothing network upgrade. It also requires pricing different from the flat-rate, which is widely employed in Internet today.

Focus has been recently shifted toward simpler, lightweight QoS architectures. A concept of non-elevated services emerged as a promising framework for QoS provisioning because it retains the simplicity of the current best-effort Internet. Non-elevated service models assume that all traffic classes receive "different but equal" quality of service. These models offer weak QoS guarantees, which are rather qualitative, as opposed to quantitative guaranties offered by IntServ and DiffServ. No policing and only limited operational changes to the current Internet architecture are required. They can be deployed incrementally and flat pricing can be preserved because none of the classes receives better QoS than the other.

In non-elevated service models, service differentiation between classes is usually achieved by employing a trade-off between various QoS metrics (delay, loss rate, and throughput). For instance, class *A* may receive lower delay but higher loss rate than class *B*. In that case, service received by the class *A* is qualitatively different, but not better than service received by the class *B*, and vice versa. A number of mechanisms for delay

[12]-[23] and loss rate differentiation [15], [24]-[27] among classes emerged recently. However, combined service differentiation across multiple QoS metrics is still an open issue because these mechanisms are not applicable to flows whose performance depends both on delay and loss rate, such as TCP flows. In case of TCP flows, delay and loss rate differentiations cannot be performed independently because their combined effect on TCP throughput might be unpredictable. Introduction of a low-delay service for delay-sensitive packets might increase the queuing delay and round-trip time for TCP packets that are used for bulk data transfer. Since the throughput of TCP flows depends on the round-trip time, delay differentiation might have negative effect on TCP applications. Since TCP traffic represents up to 95 % of the total traffic in today's Internet [28], approaches that do not consider the interaction between service differentiation mechanism and TCP are not likely to be successful in practice.

In this thesis, we propose a new non-elevated service differentiation architecture that provides delay and throughput differentiation between *delay-sensitive* and *throughput-sensitive* traffic classes. An Internet application may choose to mark its packets either as throughput-sensitive or delay-sensitive by setting a field in packets' IP headers. Both classes benefit from the proposed architecture because delay-sensitive (real-time) applications receive low-delay service, while throughput-sensitive (TCP) applications are guarantied *at least* the same throughput as they would receive in a best-effort network. In addition to the new algorithm named Packet Loss Controller (PLC) [30], the proposed architecture employs two existing router algorithms: a scheduling algorithm for proportional delay differentiation called Backlog-Proportional Rate (BPR) [12] and an active queue management (AQM) called Random Early Detection (RED)

[29]. Detailed description of these algorithms is given in Sections 5 and 6. Relative delay ratio between classes is controllable and may be set by a network administrator as a parameter of the BPR algorithm. Even though other scheduling algorithms [12]-[23] can provide proportional delay differentiation, we selected the BPR scheduler because it explicitly allocates service rates for traffic classes (queues) in an output buffer of a router. Service rates allocated by the BPR to the throughput-sensitive and delay-sensitive classes are used by the PLC controller to perform packet loss differentiation that ensures that performance of TCP applications will not be degraded by the presence of the low-delay traffic. Hence, our approach is different from existing service differentiation mechanisms because it considers the interaction between the delay differentiation algorithm and TCP. We describe this interaction as a feedback control system, which is used for designing the PLC controller in Section 6. By analyzing this system, we derive conditions that need to be satisfied by PLC in order to provide predictable throughput of TCP flows. Even though our analysis was intended to solve a specific problem (the design of the PLC controller), it also contributes to better understanding of the interaction between TCP and service differentiation mechanisms. We implemented and tested the proposed service differentiation architecture using ns-2 network simulator [31]. We evaluated PLC's ability to provide consistent and controllable delay differentiation between classes and to protect the throughput of TCP flows in the presence of delay-sensitive traffic. Our conclusion is that the proposed architecture achieves the objectives of the non-elevated service model: it provides "different but equal" quality of service to its traffic classes, it is scalable since it does not maintain any per-flow state, and it does not require admission control, traffic policing, or major operational modifications in the current Internet.

This thesis is organized as follows: Section 2 is an overview of QoS fundamentals and existing service models. A brief survey of existing proposals for non-elevated services is given in Section 3. Section 4 describes the proposed service differentiation architecture. Scheduling algorithms for proportional delay differentiation and their influence on TCP throughput are discussed in Section 5. We describe the Packet Loss Controller in Section 6 and evaluate its performance in Section 7 using various simulation scenarios. Our conclusions are presented in Section 8.

2. Quality of Service in Internet: an overview

Quality of Service (QoS) is usually defined as the ability of an application to obtain the required network service for successful operation. In order to provide QoS, a network has to offer predictable *QoS parameters*, such as throughput, packet loss probability, delay, and delay jitter. Supporting such guarantees requires cooperation between the sending and receiving hosts and network devices in intermediate nodes (switches and routers). Appropriate QoS control mechanisms need to be designed and implemented in these nodes in order to support QoS.

2.1. QoS control mechanisms

QoS control mechanisms can be classified as network-level mechanisms (employed in core routers) and application-level mechanisms (employed on network edges). We focus our attention on router mechanisms because the proposed new solution to service differentiation in the Internet relies on traffic control mechanisms employed in routers. An architecture of a QoS capable router is shown in Fig. 1. It includes several traffic control mechanisms: *packet classification*, *admission control*, *traffic policing, traffic shaping*, *buffer management*, and *packet scheduling*.

Packet classification mechanisms provide the capability to partition network traffic into multiple classes of service. Packets from different classes can be marked with distinct codes in their IP headers and, hence, receive different treatment. Proper design of packet classification mechanisms is important because, as the routers' speed increases, these mechanisms could easily become performance bottlenecks.

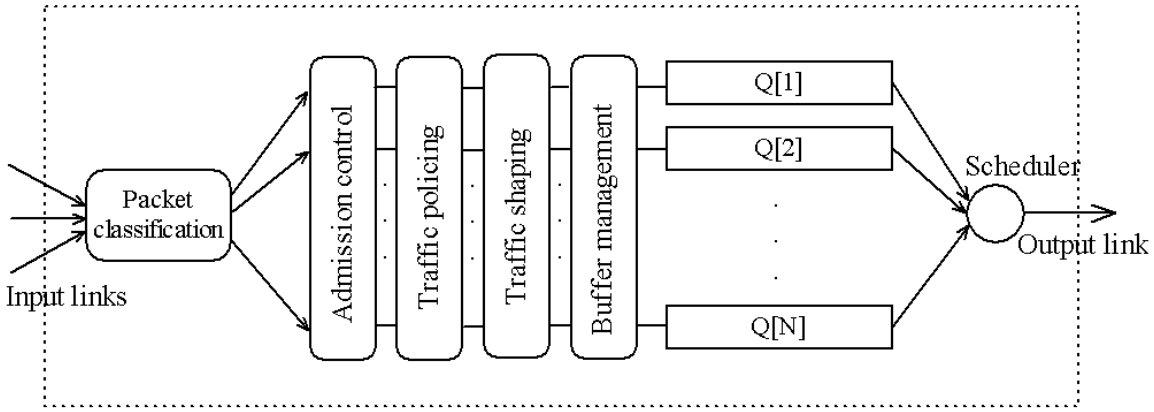

Fig. 1. Traffic control mechanisms employed in a QoS capable router.

Admission control implements a decision algorithm that a router employs to determine whether an incoming request for service can be accepted without disrupting the service guarantees to established data flows. Unlike in a best-effort network, where a new connection request is accepted from any source anytime, a Call Admission Control (CAC) algorithm is executed in a QoS-enabled network. It decides whether to accept or reject a new connection. The CAC algorithm estimates whether the available resources meet the set of QoS parameters requested by the connection [32].

Traffic policing ensures that an accepted connection conforms to the QoS parameters negotiated during the admission control phase. For each packet entering the network, the policing mechanism detects whether it conforms to the "QoS contract". If conforming, the packet is admitted to the router. Otherwise, the policing mechanism can either drop the packet or decide to accept it as a lower priority packet that will be dropped if congestion occurs. Policing is necessary only in access routers. There is no need to police the traffic in core routers [33].

Traffic shaping is used to smooth out packets bursts in cases when incoming traffic does not conform to negotiated "QoS contract". Unlike policing mechanisms,

whose sole purpose is to detect and discard violating packets, traffic shaping mechanisms store packets in buffers in order to smooth out the bursts that might affect the performance of the network. *Leaky bucket* is a well-known mechanism that can be used both as a traffic policing and a traffic shaping mechanism.

Buffer management algorithm defines the buffer sharing policy and decides whether an incoming packet should be accepted to the queue or discarded. Most widely implemented buffer management policy is Drop-Tail, which drops all incoming packets when the buffer overflows. More sophisticated algorithms react to incipient congestion and act before the buffer overflows. These algorithms are called Active Queue Management (AQM) algorithms.

Packet scheduling specifies the packet serving discipline in a node. Incoming packets are organized by a buffer management algorithm into different logical queues. Different queuing strategies are possible, such as per-flow queuing (each flow has its own logical queue) or per-class queuing (an aggregate of flows with similar QoS requirements shares a single logical queue). After a packet has been transmitted from the buffer, the packet scheduling algorithm decides which queue should be served next. Since QoS parameters of a traffic flow are significantly affected by the choice of scheduling mechanism, considerable research efforts have been focused on designing various scheduling schemes.

In addition to described network-level mechanisms in routers, application-level mechanisms for QoS monitoring, network management, pricing, and billing also need to be employed. Furthermore, in order to make use of a QoS-enabled architecture, an application must possess a level of QoS awareness so that it could request certain service

guarantees (QoS parameters). An application may request the service guarantees explicitly by a QoS signalling protocol or implicitly by marking its packet to belong to certain traffic class.

2.2. Proposed QoS architectures

We use the term *QoS architecture* to encompass a combination of mechanisms that cooperatively provide a specific QoS level to application traffic traversing a network. Today's Internet is not a QoS architecture because of the lack of widely implemented QoS mechanisms. It provides a best-effort service without any guarantees on QoS parameters. Nevertheless, majority of existing Internet applications obtain required network services for a successful operation because backbone Internet links are currently *overprovisioned*. Major drawback for possible deployment of new QoS architectures is their cost. Overprovisioning of link's capacities still remains to be more cost-effective than deployment of new QoS architectures. However, future bandwidth-hungry (also called "killer" applications) and existing applications with diverse service requirements will eventually force major Internet Service Providers (ISPs) to consider the deployment of certain QoS mechanisms.

Proposed QoS architectures can be classified based on various criteria. Based on the granularity of QoS guarantees, a QoS architecture may provide QoS guarantees to individual flows or to flow aggregates, called classes. Based on the type of QoS differentiation, the guarantees on QoS parameters (throughput, loss probability, delay, and delay jitter) may be absolute (deterministic) or relative (proportional to QoS parameters of other flows or traffic classes). Based on the scope of the guarantees, a QoS architecture may provide local (per-hop) or end-to-end QoS guarantees. Existing QoS

architectures (implemented or proposed) are generally classified into three *service models*: Non-Elevated Services, Differentiated Services, and Integrated Services. A basic difference between these service models is in the level of QoS guarantees that they offer and their implementation complexity, as illustrated in Fig. 2. In the reminder of this Section, we discuss the advantages and disadvantages of the proposed models.

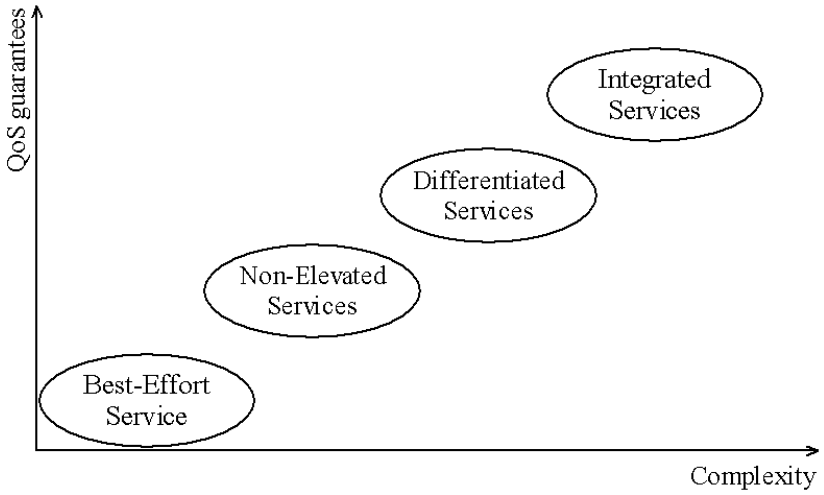

Fig. 2. QoS guarantees vs. complexity of existing QoS service models.

2.2.1. Integrated Services (IntServ)

The Integrated Services (IntServ) model has been proposed in the early 90s by the Internet Engineering Task Force (IETF) [6], [7]. IntServ model focuses on providing absolute QoS guarantees to individual flows. Its key building blocks are Resource Reservation Protocol (RSVP) and admission control. RSVP defines signalling procedures used by a sender to reserve sufficient resources at each router on the path to the receiver and to ensure that its QoS requirements are satisfied. In order to reserve the resources, RSVP maintains *flow-state* information (e.g., throughput requirements, maximum tolerable delay) in routers. in addition to existing best-effort class ,IntServ model defines two traffic classes: *guaranteed service* (GS) [6] and *controlled load service* (CLS) [7].

Guaranteed service (GS) class provides firm bounds on throughput and maximum end-to-end packet delay. This service is designed for real-time applications that are intolerable to delay, such as Internet telephony. A GS flow is characterized by peak rate and leaky bucket parameters (token generation rate and buffer size). GS provides zero loss probability for policed flows.

Controlled load service (CLS) class provides a level of QoS to a flow that closely approximates the QoS level that the flow would receive in a non-congested node of a best-effort network. CLS class is designed for real-time applications that can tolerate variations in packet delay and delay jitter, such as audio/video streaming. A receiver uses a *playout buffer* to smooth out the delay jitter. CLS class does not provide maximum delay bounds such as the GS class. Traffic flows belonging to CLS class do not negotiate their rate during the admission phase. When the limited amount of bandwidth allocated for the CLS class gets exhausted, additional packets are likely to receive only the best-effort service.

Major reason for abandoning the IntServ model is scalability problem. In addition to the admission control, RSVP signalling, and per-flow state maintenance, IntServ model requires packet scheduling and buffer management on per-flow basis. Backbone routers are usually traversed by an enormous number of flows. Hence, the amount of state information that has to be processed imposes an unacceptable overhead on the routers. Furthermore, every network node must implement QoS mechanisms required by the IntServ model. Hence, the IntServ can not be deployed incrementally and, hence, requires substantial initial infrastructural investments.

2.2.2. Differentiated Services (DiffServ)

The Differentiated Services (DiffServ) model is a more recent approach proposed by the IETF [8]-[10]. It provides scalable QoS differentiation by avoiding the maintenance of per-flow state in core routers and by shifting the complexity to the network edges. In the DiffServ model, edge routers are responsible for traffic policing and shaping according to the *traffic conditioning agreement* (TCA) established between a user and an ISP as a part of a *service level agreement* (SLA). The TCA typically includes traffic characteristics (leaky bucket parameters) enforced by the ISP and QoS parameters requested by the user. Edge routers are also responsible for packet classification into traffic aggregates called classes. Each class is associated with a specific Per-Hop Behaviour (PHB) descriptor that provides the class with a specific treatment by network nodes. In the network core, packets are forwarded based on PHB descriptors indicated by DiffServ codepoints (DSCPs) in their IP headers. Since QoS differentiation in the core is provided on packet level, there is no need to maintain per-flow states. Unlike in the IntServ model, DiffServ routers employ per-class buffer management and packet scheduling, where the number of classes is significantly lower than the number of flows. Multiple classes (PHBs) are grouped together to form a *PHB group*. Currently, two PHB groups are defined by the IETF: Expedited Forwarding PHB group (EF) [9] that provides *premium service* and Assured Forwarding PHB group (AF) [10] that provides *assured service*.

The Expedited Forwarding (EF) provides users with an abstraction of a leased line. In a DiffServ router, the EF traffic should receive specified rate irrespective of any other traffic transiting the router. To enforce this specified rate, edge routers must police

all incoming traffic. User is not allowed to exceed the peak rate specified by the SLA. Hence, premium service provided by EF is low-delay, low-jitter, and nearly constant bit rate service suitable for Internet telephony and video conferencing.

The Assured Forwarding (AF) defines four forwarding classes with three drop priorities within each class. Each combination of AF service class and drop priority is associated with distinct DSCP code. In every DiffServ node, certain amount of buffer space and bandwidth is allocated to each class. In case of congestion, the drop priority of a packet determines its relative importance within its class. In order to protect packets with lower drop priority from being discarded, a DiffServ router preferentially discards packets with higher drop priorities. Hence, in the case of network congestion, the assured traffic with lower priority may still experience packet losses and high delay as in a best-effort network.

Even though DiffServ model eliminates certain drawbacks of IntServ, such as the scalability problem, it has not been widely accepted and deployed. Since DiffServ provides per-hop QoS differentiation on a class level, it is difficult to ensure end-to-end performance defined by a SLA contract. From a commercial viewpoint, this is a major disadvantage because it implies that it is impossible to charge the service defined by the SLA. Furthermore, DiffServ model cannot be deployed incrementally because all network nodes are required to support DiffServ mechanisms in order to provide specified QoS parameters. In 2001, Internet2 QoS Working Group discontinued deployment and testing of the premium service because of the lack of demand and support from major router vendors [11].

2.2.3. Non-Elevated Services

Non-Elevated Services model is a recently proposed framework for service differentiation in the Internet [11]. This model provides a set of traffic classes with asymmetrically differentiated QoS parameters (throughput, delay, loss probability). For instance, one class may receive lower packet delay but higher loss probability compared to the best-effort class, while another class may receive higher delay but lower loss probability. Neither class is absolutely better than the other: each class has a set of QoS parameters that make it more appropriate for certain applications but less appropriate for the other. Hence, the non-elevated model provides "different but equal" QoS to traffic classes. Pricing and billing may remains the same as in the current Internet because there is no need to price the classes differently. Applications are free to choose the class that best suits their performance requirements by setting a marker in packets' IP headers. No admission control, traffic policing, or shaping are required. In addition to an application-level algorithm used to choose the traffic class, non-elevated architectures rely solely on buffer management and packet scheduling mechanisms.

Non-elevated architectures usually provide relative per-hop guarantees, although different solutions are possible. Service guarantees provided by these architectures are weak. Instead of QoS provisioning, *service differentiation* is a more appropriate name to describe the functionality of non-elevated architectures. Unlike IntServ and DiffServ architectures, non-elevated architectures cannot provide absolute guarantees on QoS parameters. However, low implementation complexity and flat pricing are strong incentives for their deployment. Non-elevated mechanisms can be deployed incrementally because they provide local, per-hop service differentiation. In a non-

elevated network, a real-time application has high probability to actually receive lower delay than in a best-effort network even though not every router in the network supports the non-elevated mechanisms. Moreover, price for this service is the same as for the best-effort service. Various properties of Integrated, Differentiated, and Non-elevated Services are summarized in Table 1.

Table 1. Comparison of service differentiation models.

	Integrated Services	Differentiated Services	Non-Elevated Services
Granularity of service guarantees	individual flows	flow aggregates (classes)	flow aggregates (classes)
Type of service guarantees	absolute	absolute or relative	relative
Scope of service guarantees	end-to-end	per-hop	per-hop
State in routers	per-flow	per-class	not required
Admission control	required	required for absolute differentiation	not required
Resource reservation	required	required for absolute differentiation	not required
All-or-nothing upgrade	required	required	not required

Before presenting a new architecture for non-elevated services, in Section 3 we describe several existing proposals.

3. Survey of existing proposals for non-elevated services

There are currently three proposed architectures for non-elevated services: Alternative Best-Effort (ABE) [3], Best-Effort Differentiated Services (BEDS) [4], and Equivalent Differentiated Services (EDS) [5]. We describe here mechanisms employed in these architectures and address some of their drawbacks.

3.1. Alternative Best-Effort (ABE)

ABE [3] introduced two traffic classes: a low delay class named "green" and a "blue" class that corresponds to the best-effort class. Green packets are guarantied low bounded delay, while blue traffic receives at least as much throughput as it would receive in a flat best-effort network. In order to ensure that "green does not hurt blue", two conditions must hold: *local transparency* and *throughput transparency* for blue packets. Local transparency assumes that the delay experienced by blue packets in ABE is not higher than the delay they would experience in a best-effort network. Furthermore, a blue packet is not discarded if it would not be discarded in a best-effort network. Throughput transparency implies that blue flows receive at least the same throughput as they would receive in a best-effort network. ABE provides local transparency for blue, while throughput transparency condition is considered to be loose.

A router implementation of ABE called Duplicate Scheduling with Deadlines (DSD) [3] is illustrated in Fig. 3. Duplicates of all incoming packets are sent to a virtual queue and accepted if the virtual buffer is not full. The duplicates in the virtual queue are served according to the *first-in first-out* (FIFO) discipline, as they would be served in a

best-effort network. If its duplicate is accepted into the virtual queue, a blue packet is assigned a deadline based on the expected finish time of the duplicate and it is placed in the blue queue. A green packet is accepted into the green queue if it passes the so-called "*green acceptance test*". A green packet arriving to the queue passes the test if there is a possibility that it will be served before a specified delay bound *d* expires, and fails otherwise. This possibility is evaluated based on the number of green and blue packets that are already waiting in queues. Blue packets are always served no later than their deadline. Green packets are served in the meantime if they have been in the queue for less than the specified delay bound *d* and are dropped otherwise. A control loop is used to decide which packet should be served first if the deadlines of both head-of-line packets still can be met, even if the other queue is served beforehand.

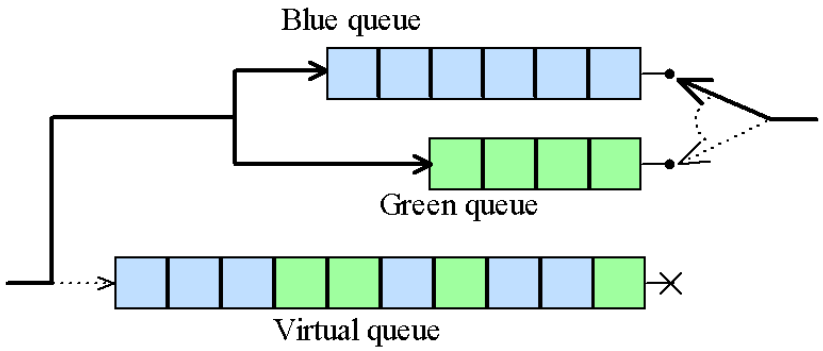

Fig. 3. DSD architecture: duplicates of all incoming packets are sent to a virtual queue. Original packets are classified according to their colours into the blue or green queue.

3.2. Best-Effort Differentiated Services (BEDS)

BEDS [4] defines two types of services: *drop-conservative* for TCP and *delay-conservative* for UDP flows. TCP and UDP packets are queued in two Random Early Detection (RED) [29] queues with distinct sets of parameters.

RED [29] is an active queue management (AQM) mechanism recommended by IETF. RED has been widely accepted and implemented by major router manufacturers. Unlike in Drop Tail mechanism, which drops incoming packets when buffer overflows, drop probability p in RED is calculated as an increasing function of an average queue size $\overline{q}$:

$$p := H(\overline{q}) = \begin{cases} 0 & if\ \overline{q} \le q_{\min} \\ \dfrac{\overline{q} - q_{\min}}{q_{\max} - q_{\min}} p_{\max} & if\ q_{\min} < \overline{q} < q_{\max} \\ 1 & if\ \overline{q} \ge q_{\max} \end{cases}, \tag{1}$$

where q_{min} and q_{max} are minimum and maximum queue thresholds and p_{max} is a maximum drop probability. The function of the drop probability p is shown in Fig. 4. The average queue size $\overline{q}$ is calculated as a weighted moving average of the instantaneous queue size q. It is updated upon each packet arrival to the queue as:

$$\overline{q}_k = (1 - w)\overline{q}_{k-1} + wq_k , \tag{2}$$

where k is the packet number and w is an averaging weight.

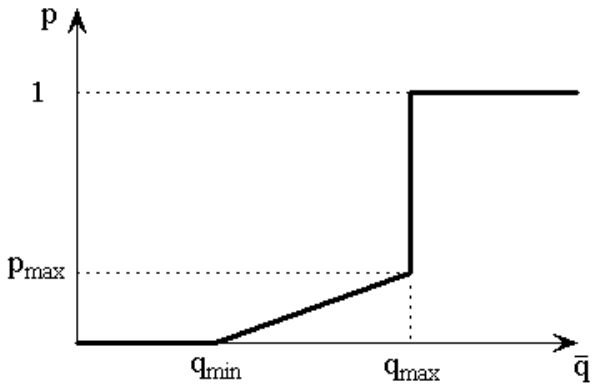

Fig. 4. RED's drop probability *p* is a function of the average queue size $\overline{q}$.

RED has been designed to cooperate with TCP's congestion control algorithm. TCP flows recognize packet losses as a sign of congestion and reduce their sending rate. RED begins to drop packets before the buffer overflows and, thus, provides an early feedback about congestion level on the output link and allows TCP flows to react in a timely manner. The interval where average queue sizes lie between q_{min} and q_{max} is called *early drop region*.

In [4], two types of RED control have been defined depending on the slope of the function $p = H(\overline{q})$ in the early drop region: *drop-conservative* and *delay-conservative.* A drop-conservative RED control results in lower packet loss probability and larger delay. To the contrary, a delay-conservative RED control results in higher packet loss probability and lower delay. In the case of FCFS scheduling, queuing delay is always proportional to the average queue size $\overline{q}$. The idea employed in BEDS is illustrated in Fig. 5.

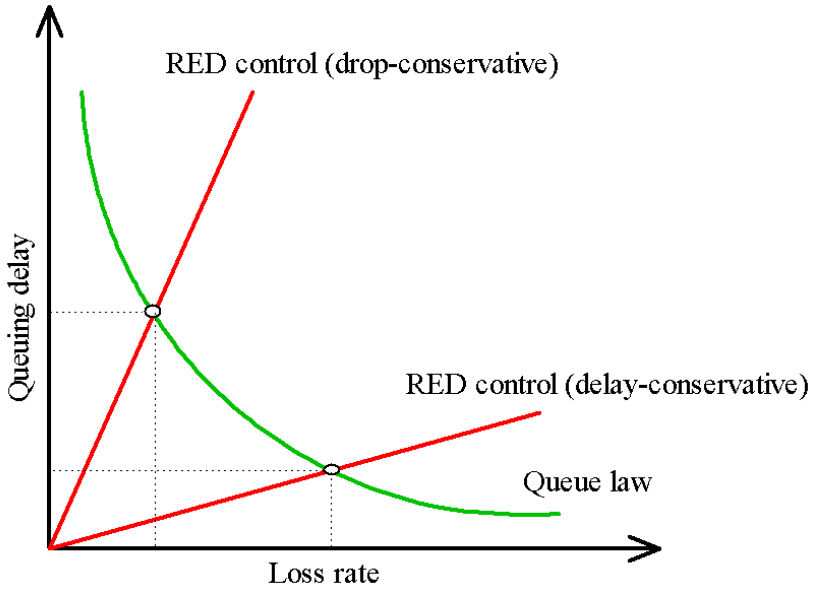

Fig. 5. The intersections of RED control functions and queue law determine average queuing delays and average loss probabilities of traffic classes in BEDS.

The intersections of the RED control functions and a *queue law* in Fig. 5 determine the operating points for drop-conservative (TCP) and delay-conservative (UDP) traffic in BEDS. The queue law describes a relationship between queue length and packet loss probability. In case of *n* identical TCP flows and *m* identical UDP flows, the queue law $q = G(p)$ is implicitly defined by:

$$nT(p, R_0 + q/C) + m(1-p)U = C\,, \tag{3}$$

where $T(p, R_0 + q/C)$ is the average throughput of TCP flows derived in [34]:

$$T(p, R_0 + q/C) = \sqrt{\frac{3}{2}}\frac{M}{(R_0 + q/C)\sqrt{p}}\,, \tag{4}$$

R_0 is the round-trip propagation delay of packets, C is the capacity of the output link, U is UDP rate, and M is the average packet size.

In addition to the RED buffer management, BEDS employs a scheduling mechanism to decide which RED queue should be served next. Based on results of the packet scheduling, three versions of BEDS have been defined: BEDS with forced delay ratio, BEDS with forced loss ratio, and BEDS with forced delay and loss ratio [4].

BEDS with forced delay ratio employs Weighted Fair Queuing (WFQ) scheduler [36] to provide services to the drop-conservative (TCP) and delay-conservative (UDP) queues such that the ratio of queuing delays in these two queues is close to a specified value δ. In order to describe the WFQ scheduler, let us assume an ideal fluid flow system where queues are served by a bit-by-bit round-robin scheduler. A certain *queue weight* is assigned to each queue. Let a *packet finishing time* denote the time when the last bit of the packet is served in such an idealized system. Finishing times of head-of-line packets

in each queue are scaled by corresponding queue weights. WFQ can be then implemented by serving the head-of-line packets in the increasing order of their scaled finishing times. In BEDS with forced delay ratio, WFQ's weights w_{TCP} and w_{UDP} are adjusted at each packet arrival so that:

$$\frac{w_{UDP}}{w_{TCP}} = \delta \frac{q_{UDP}}{q^{+}_{TCP}}, \tag{5}$$

where $q^{+}_{TCP} = \max(q_{TCP}, 1)$. If the queue weights for TCP and UDP queues satisfy (5), the ratio of queuing delays in these two queues will be close to the specified value δ [4]. The deficiency of BEDS with forced delay ratio is that only UDP traffic benefits from the proposed mechanism. TCP traffic suffers from increased queuing delay and, consequently, lower throughput compared to the throughput achieved in a best-effort network. This type of service differentiation does not comply with the concept of non-elevated services because traffic classes do not receive "different but equal" quality of service.

BEDS with forced loss ratio employs strict priority scheduling, with UDP queue having higher priority. Drop probability for a packet is calculated on every packet arrival to the TCP or UDP queue. For TCP packets, drop probability p_{TCP} is calculated based on the drop-conservative RED control function in the TCP queue. For UDP packets, the drop probability p_{UDP} is calculated from the TCP drop probability p_{TCP} as:

$$p_{UDP} = \lambda p_{TCP}, \tag{6}$$

where λ is the desired loss rate ratio. However, enforcing the loss ratio λ might be futile, considering the different effect of packet losses on the performance of TCP and UDP

flows. Quality of services received by TCP and UDP applications can not be compared with certitude based on the ratio of their packet loss probabilities. UDP applications are usually less affected by occasional packet losses than TCP applications. Furthermore, combined effect of strict priority scheduling and proportional loss rate differentiation on TCP throughput is neither controllable nor predictable when UDP load varies.

BEDS with forced delay and loss ratio is a combination of the previous two versions of BEDS. In this case, relationship between delay ratio δ and loss ratio λ is an important issue that has not been discussed in [4]. Since the throughput of TCP flows depends both on delay (round-trip time) and loss probability, parameters δ and λ cannot be set independently. Furthermore, no results have been presented to illustrate the influence of the proposed mechanism on throughput of TCP and UDP flows.

3.3. Equivalent Differentiated Services (EDS)

EDS [5] defines a set of classes with asymmetric delay and loss differentiation. Delay and loss differentiation parameters are set to provide lower delay and higher loss probability to class *i* and higher delay and lower loss probability to class *j*, as illustrated in Fig. 6. EDS relies on delay and loss differentiation mechanisms proposed in [12] and [24]: Waiting-Time Priority (WTP) scheduler and Proportional Loss Rate (PLR) dropper.

WTP [12], [24] is a priority scheduler where the priority of a packet is proportional to its waiting time in a queue. The priority of the head-of-line packet in queue (class) *i* at time *t* is:

$$p_i(t) = w_i(t)s_i \tag{7}$$

where $w_i(t)$ is the waiting time of the packet and s_i is the *delay differentiation parameter* of class i. Setting the delay differentiation parameter s_i to a larger value results in a lower delay experienced by packets in class i. It has been shown [12], [24] that average waiting time for any two classes i and j serviced by WTP satisfies:

$$\frac{\bar{d}_i}{\bar{d}_j} \to \frac{s_j}{s_i} \tag{8}$$

under heavy-load conditions. Therefore, choice of delay differentiation parameters s_i and s_j determines the ratio of average queuing delays in these two classes.

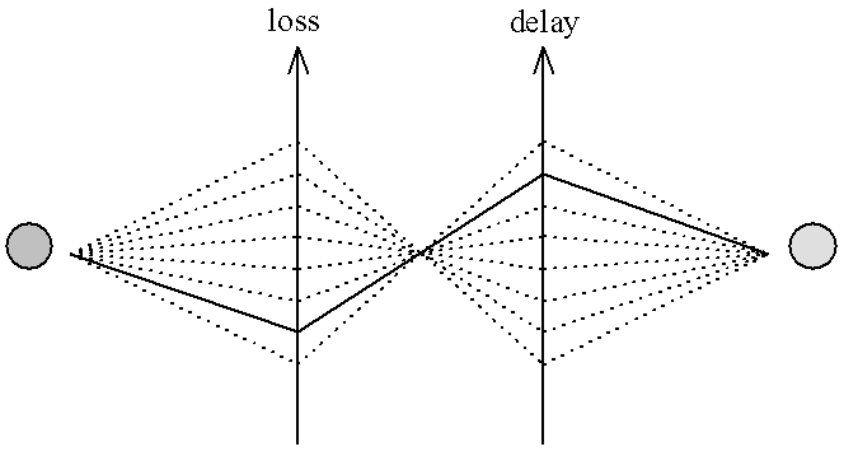

Fig. 6. Traffic classes in EDS model receive asymmetric loss and delay performance.

PLR dropper [24] is a packet dropping mechanism that closely approximates proportional loss rate differentiation between classes. PLR maintains estimates of loss rates l_i in each class. When an active queue management (AQM) algorithm in a router decides that a packet should be dropped, PLR selects the backlogged class with the minimum normalized loss rate l_i/σ_i, where σ_i is the loss differentiation parameter of class i, and drops a packet from class i. Dropping a packet from class i tends to equalize

normalized loss rates among classes. Therefore, for any two classes *i* and *j*, PLR ensures that:

$$\frac{\bar{l}_i}{\bar{l}_j} \to \frac{\sigma_j}{\sigma_i}, \tag{9}$$

where $\bar{l}_i, \bar{l}_j$ and σ_i, σ_j are average loss rates and loss differentiation parameters for classes *i* and *j*, respectively. Therefore, the choice of loss differentiation parameters σ_i and σ_j determines the ratio of average loss rates in classes *i* and *j*. Two versions of PLR introduced in [24], PLR(M) and PLR(∞), differ in the duration of time intervals over which loss rates l_i are estimated. In the PLR(M) dropper, the loss rate of class *i* is estimated as a fraction of dropped packets from class *i* for the last *M* packet arrivals. A cyclic queue with *M* entries, called Loss History Table (LHT), contains records of the drop status of each packet for the last *M* arrivals. Based on the information stored in the LHT, the PLR(M) dropper calculates the number of arrivals and drops from class *i* in the last *M* arrivals [24]. The PLR(∞) dropper estimates the loss rates l_i using simple counters for arrivals and drops in every class. For 32-bit counters, length of the time interval over which the loss rates l_i are estimated is over four billion packet arrivals. Hence, we may assume that the PLR(∞) dropper has "infinite" memory. Estimation of the loss rates based on long history of packet arrivals makes PLR(∞) more precise, but less adaptive to changes in class load distributions [24].

Tuning WTP's delay differentiation parameters s_i and PLR's loss differentiation parameters σ_i is not performed by the EDS mechanism itself, but rather by an upper layer protocol. Another option, suggested by authors in [5], is to offer a set of classes with

fixed delay and loss differentiation parameters and let senders decide which class satisfies their service requirements.

Major problem with EDS is that it does not consider the behaviour of transport protocols employed by various applications. For instance, TCP flows are responsive and they decrease their sending rate if a packet loss has been detected. Furthermore, sending rate of a TCP flow depends on the round-trip time of its packets. Therefore, in order to achieve predictable performance of TCP flows, loss and delay differentiation parameters in EDS cannot be set arbitrarily. In [5] authors suggest that complexity of TCP may lead them to design a new transport protocol, rather than to adapt TCP to EDS. However, since TCP is already widely deployed, the only feasible solution is to adapt service differentiation mechanisms to TCP, not the opposite.

In the Section 4, we introduce a new service differentiation mechanism for non-elevated services that alleviates this problem. We define the objectives of the proposed mechanism and describe its architecture.

4. Proposed service differentiation architecture for non-elevated services

In this Section, we propose a new solution to service differentiation in Internet. Proposed architecture relies entirely on router mechanisms to provide weak service guaranties. In addition to existing scheduling and buffer management algorithms, proposed architecture employs a new algorithm named Packet Loss Controller (PLC) to ensure that traffic classes receive "different but equal" quality of service.

Internet applications can be classified as delay-sensitive (DS) and throughput-sensitive (TS). Delay-sensitive applications, such as IP Telephony and Video on Demand, employ UDP or similar unresponsive protocols. Throughput-sensitive applications, such as File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), and Simple Mail Transfer Protocol (SMTP), employ TCP as a transport protocol. Therefore, service differentiation problem can be considered as a problem of providing different treatment to TCP and UDP flows. An application may use last two bits of DS field in the IP header, which are not yet assigned, to mark its packets as delay-sensitive or throughput-sensitive [37]. Two objectives of a new service differentiation architecture that we propose in this thesis are to provide:

1) proportionally lower delay to delay-sensitive class (UDP), compared to the delay experienced by throughput-sensitive class (TCP);
2) at least the same throughput to throughput-sensitive flows (TCP) as they would receive in a best-effort network.

The first objective is accomplished by using a Backlog-Proportional Rate (BPR) scheduler, introduced in [12]. The BPR classifies the throughput-sensitive and delay-

sensitive flows into two different queues and assignees service rates to these queues that will ensure that the ratio of average queuing delays for throughput-sensitive and delay-sensitive classes is equal to the ratio specified by a network operator. In Section 5, we present an ideal rate assignment strategy for proportional delay differentiation named Optimized BPR (BPR^+) in order to evaluate the performance of BPR scheduler. BPR^+ [38] is an extension of BPR. It employs more complete information about backlogged traffic to assign service rates to TS and DS queues in order to more precisely approximate the specified delay ratio δ. Even though BPR^+ is to complex to warrant practical implementations, it is useful as a benchmark for evaluating other possible rate assignment strategies for proportional delay differentiation. Detailed description and performance comparison of BPR^+ and BPR is given in Section 5.

The second objective is accomplished using a new algorithm, named Packet Loss Controller (PLC). Since BPR scheduler is work-conserving, average queuing delay is identical as in the case of FCFS scheduling employed in a best-effort network. If the total number of packets accepted in the buffer is maintained to be the same as it would be in a best-effort network, lower delay for delay-sensitive packets can be achieved only by increasing the queuing delay for throughput-sensitive packets. Since throughput of a TCP flow depends on its round-trip time, introduction of low-delay service for delay-sensitive class will deteriorate the performance of throughput-sensitive applications that use TCP as a transport protocol. The PLC controller is used to overcome this problem by protecting the throughput of TCP flows. It guarantees that the TCP throughput in the presence of delay differentiation is not lower than in a best-effort network. PLC

controller can be considered as an addition to the well-known RED algorithm [29]. Details of the PLC controller are given in Section 6.

The proposed architecture for service differentiation is shown in Fig. 7.

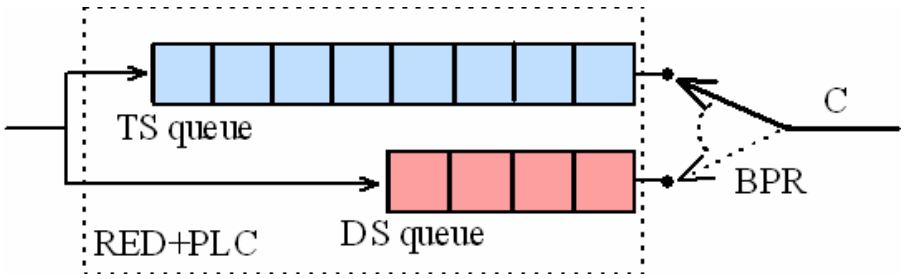

Fig. 7. The proposed architecture for service differentiation: throughput-sensitive (TCP) and delay-sensitive (UDP) queues are managed by RED with PLC controller. Packets are scheduled by BPR.

The new architecture differs from other proposals because it considers the interaction between the delay differentiation algorithm and TCP's congestion control algorithm. Negative effect of proportional delay differentiation on TCP throughput is eliminated by the PLC controller. Even though router mechanisms [12]-[23] are also able to provide proportional delay differentiation, they are only applicable to protocols whose throughput does not depend on round-trip time, such as UDP. Since the majority of Internet applications employ TCP as a transport protocol, practical value of these mechanisms is questionable. Work presented in this thesis contributes to better understanding of requirements that need to be satisfied in order to extend router mechanisms for service differentiation to the "real world" scenarios.

5. Scheduling for proportional delay differentiation

Proportional delay differentiation assumes that the ratio of average queuing delays of throughput-sensitive and delay-sensitive packets should be approximately equal to a specified delay differentiation parameter. We consider a work-conserving scheduler serving two FIFO queues managed by an active queue management (AQM) algorithm, which makes drop decisions upon packet arrivals (once being admitted, the packet is never dropped from the queue). The work-conserving property assumes that the scheduler does not idle when there are packets in a queue waiting to be transmitted.

Scheduling algorithms for proportional delay differentiation provide relative per-hop delay guarantees. One may argue that real-time applications require absolute end-to-end guarantees because their performance depends on probability of being bellow certain end-to-end delay threshold [13]. Proportional delay differentiation decreases the probability of exceeding the threshold for the delay-sensitive class. This makes it more suitable for real-time applications than the existing best-effort service. It has been also shown that per-hop proportional delay differentiation easily translates into proportional end-to-end delay differentiation [23]

In this Section, we discuss two schedulers for proportional delay differentiation: the well-known Backlog-Proportional Rate scheduler (BPR) scheduler [12] and a newly proposed idealized scheduler named Optimized Backlog-Proportional Rate (BPR^{+}) scheduler [38] that we introduced in order to evaluate the performance of BPR compared

to a close-to-optimal rate assignment strategy that considers waiting times of individual packet in queues.

5.1. Backlog-Proportional Rate scheduler (BPR)

Backlog Proportional Rate (BPR) scheduler for proportional delay differentiation has been introduced in [12]. It is also known as Proportional Queue Control Mechanism (PQCM) [22].The basic idea used in BPR is to assign class service rates (r_{TS} and r_{DS}) proportionally to queue lengths (q_{TS} and q_{DS}) and a desired delay differentiation parameter $\delta \geq 1$:

$$\frac{r_{TS}}{r_{DS}} = \frac{1}{\delta}\frac{q_{TS}}{q_{DS}}. \tag{10}$$

If $C = r_{TS} + r_{DS}$ is the output link's capacity, service rates r_{TS} and r_{DS} can be calculated from (10). According to Little's law, average queuing delays of throughput-sensitive and delay-sensitive packets are $d_{TS} = q_{TS} / r_{TS}$ and $d_{DS} = q_{DS} / r_{DS}$, respectively. The ratio of these delays approaches δ, as shown in [12]:

$$\frac{d_{TS}}{d_{DS}} \to \delta . \tag{11}$$

BPR scheduler is based on the fluid server model and, hence, can be only approximated in practice. One possible way to approximate BPR is given in [12].

BPR provides predictable and controllable proportional delay differentiation. The differentiation is predictable in the sense that it is consistent regardless of variations in traffic load. It is also controllable in the sense that network administrator is able to adjust average delay ratio between classes.

5.2. BPR$^+$: an optimized BPR scheduler

Optimized Backlog-Proportional Rate (BPR$^+$) is a new scheduler that considers not only the lengths of throughput-sensitive and delay-sensitive queues, but also the arrival times of individual packets when it assigns service rates to the queues [38]. We were motivated by the need to know the form of an optimal scheduler for proportional delay differentiation, even if it might be to complex to be practical. In our simulation study, we use BPR$^+$ as a benchmark for evaluating BPR performance.

BPR$^+$ is based on a fluid traffic model. We provide here a set of definitions that are used to characterize the fluid traffic model.

We say that a server is busy if there are packets in queues waiting to be transmitted. The input curve $R^{in}(t)$ is defined as a cumulative amount of traffic that has entered a queue since the beginning of the current busy period. The output curve $R^{out}(t)$ is the cumulative amount of traffic that has been transmitted from the queue since the beginning of the current busy period [39], [40]. An example of input and output curves is shown in Fig. 8. At every moment, service rate of a queue is equal to the slope of its output curve. Vertical and horizontal distance between input and output curve are current backlog and delay, respectively.

Input and output curves for throughput-sensitive class ($R_{TS}^{in}(t)$ and $R_{TS}^{out}(t)$) and delay-sensitive class ($R_{DS}^{in}(t)$ and $R_{DS}^{out}(t)$) are shown in Fig. 9. If TS queue is served with rate $\dot{R}_{TS}^{out}(\tau)$ at time τ, the average delay of TS backlog $\bar{d}_{TS}$ can be calculated as a sum of experienced delay $\bar{d}_{TS}^{E}$ and expected residual delay $\bar{d}_{TS}^{R}$:

$$\bar{d}_{TS}(\tau) = \bar{d}_{TS}^{E}(\tau) + \bar{d}_{TS}^{R}(\tau). \tag{12}$$

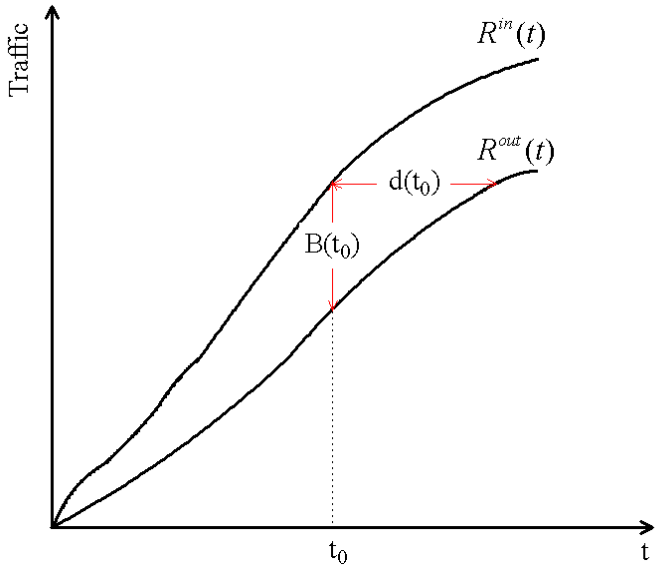

Fig. 8. Example of an input and output curves: vertical and horizontal distances between the curves are current backlog and delay, respectively.

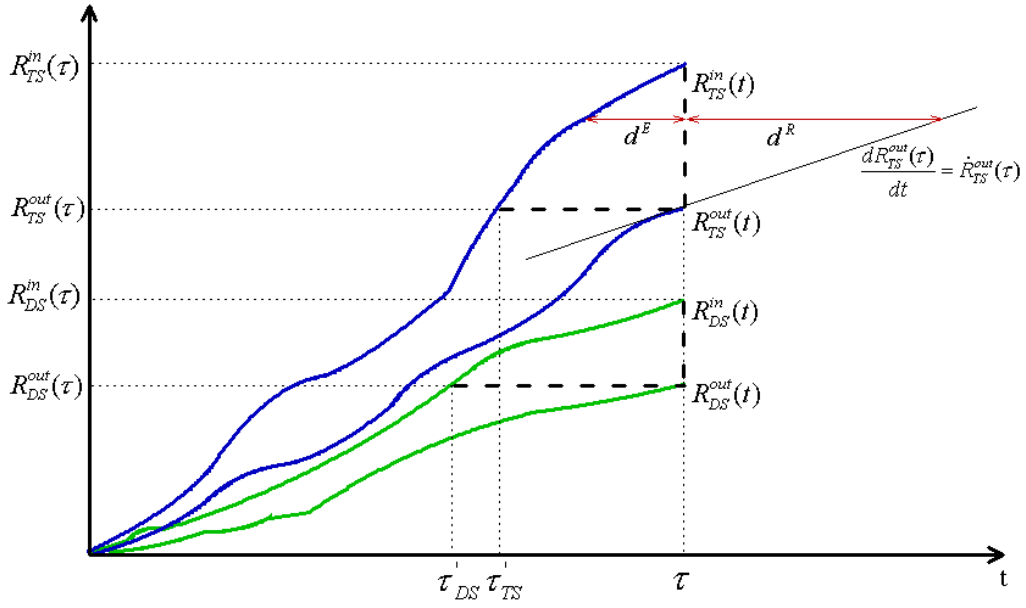

Fig. 9. Input and output curves for throughput-sensitive and delay-sensitive class: average delay is a sum of experienced and residual delays.

The average experienced delay of TS backlog is:

$$\bar{d}^{E}_{TS}(\tau) = \frac{1}{R^{in}_{TS}(\tau) - R^{out}_{TS}(\tau)} \int_{\tau_{TS}}^{\tau} (R^{in}_{TS}(t) - R^{out}_{TS}(\tau))dt\,, \qquad (13)$$

where $\tau_{TS}^{'}$ is obtained from the condition $R_{TS}^{in}(\tau_{TS}^{'}) = R_{TS}^{out}(\tau)$. The average residual delay of TS backlog is:

$$\bar{d}_{TS}^{R}(\tau) = \frac{1}{\dot{R}_{TS}^{out}(\tau)(\tau - \tau_{TS}^{'})} \int_{\tau_{TS}^{'}}^{\tau} (R_{TS}^{in}(t) - R_{TS}^{out}(\tau))dt \,. \tag{14}$$

From (12), (13), and (14), expected average queuing delay for throughput-sensitive class is:

$$\bar{d}_{TS}(\tau) = \left(\frac{1}{B_{TS}(\tau)} + \frac{1}{\dot{R}_{TS}^{out}(\tau)(\tau - \tau_{TS}^{'})} \right) I_{TS}(\tau)\, dt \,, \tag{15}$$

where $B_{TS}(\tau) = R_{TS}^{in}(\tau) - R_{TS}^{out}(\tau)$ and $I_{TS}(\tau) = \int_{\tau_{TS}^{'}}^{\tau} \left(R_{TS}^{in}(t) - R_{TS}^{out}(\tau) \right) dt$.

Using a similar procedure, expected average queuing delay for the delay-sensitive class can be expressed as:

$$\bar{d}_{DS}(\tau) = \left(\frac{1}{B_{DS}(\tau)} + \frac{1}{\dot{R}_{DS}^{out}(\tau)(\tau - \tau_{DS}^{'})} \right) I_{DS}(\tau) \,, \tag{16}$$

where $B_{DS}(\tau) = R_{DS}^{in}(\tau) - R_{DS}^{out}(\tau)$, $I_{DS}(\tau) = \int_{\tau_{DS}^{'}}^{\tau} \left(R_{DS}^{in}(t) - R_{DS}^{out}(\tau) \right) dt$, and $R_{DS}^{in}(\tau_{DS}^{'}) = R_{DS}^{out}(\tau)$.

Service rates $\dot{R}_{TS}^{out}(\tau)$ and $\dot{R}_{DS}^{out}(\tau)$ that satisfy $\frac{\bar{d}_{TS}}{\bar{d}_{DS}} = \delta$ and $\dot{R}_{TS}^{out}(\tau) + \dot{R}_{DS}^{out}(\tau) = C$ can be calculated from:

$$\begin{aligned} &\dot{R}_{TS}^{out\,2} + \dot{R}_{TS}^{out} \left(\frac{1}{B_{TS}(\tau) - \delta Z(\tau) B_{DS}(\tau)} \left(\frac{1}{\tau_{TS}^{'}} + \frac{\delta Z(\tau)}{\tau_{DS}^{'}} \right) - C \right) - \frac{1}{B_{TS}(\tau) - \delta Z(\tau) B_{DS}(\tau)} \frac{C}{\tau_{TS}^{'}} = 0 \\ &\dot{R}_{DS}^{out} = C - \dot{R}_{TS}^{out} \end{aligned} \,, \tag{17}$$

where $Z(\tau) = \frac{I_{DS}(\tau)}{I_{TS}(\tau)}$. If (17) is satisfied for every τ, the ratio of average queuing delays for throughput-sensitive and delay-sensitive class during busy periods will be δ.

Since the described BPR$^+$ server is based on a fluid model of Internet traffic, it cannot be implemented in practice. Therefore, a heuristic approximation is used.

5.2.1. A heuristic approximation of BPR$^+$

We describe a heuristic approximation of BPR$^+$ fluid server that reflects the packet nature of Internet traffic.

A timestamp is associated with every packet accepted to TS or DS queues. These timestamps are used to calculate the average experienced delay of backlogged packets. For throughput-sensitive packets, the average experienced delay is:

$$\bar{d}_{TS}^{E} = \frac{1}{N_{TS}} \sum_{k=1}^{N_{TS}} (t - t_k), \tag{18}$$

where N_{TS} is number of the packets in TS queue and t_k is the arrival time of the k^{th} packet. If served with rate r_{TS}, the average residual time of throughput-sensitive packets is:

$$\bar{d}_{TS}^{R} = \frac{1}{N_{TS}} \sum_{k=1}^{N_{TS}} \frac{q_k}{r_{TS}}, \tag{19}$$

where q_k is the amount of traffic in TS queue that has to be served before the k^{th} packet (Fig. 10).

Therefore, expected average delay of throughput-sensitive packets is:

$$\bar{d}_{TS} = D_{TS} + \frac{1}{r_{TS}} Q_{TS}, \tag{20}$$

where $D_{TS} = t - \frac{1}{N_{TS}} \sum_{k=1}^{N_{TS}} t_k$ and $Q_{TS} = \frac{1}{N_{TS}} \sum_{k=1}^{N_{TS}} q_k$.

Similarly, expected average delay of delay-sensitive packets is:

$$\bar{d}_{DS} = D_{DS} + \frac{1}{r_{DS}} Q_{DS}, \tag{21}$$

where $D_{DS} = t - \frac{1}{N_{DS}} \sum_{k=1}^{N_{DS}} t_k$ and $Q_{DS} = \frac{1}{N_{DS}} \sum_{k=1}^{N_{DS}} q_k$.

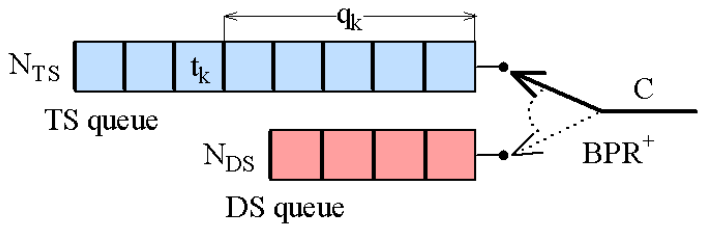

Fig. 10. The heuristic approximation of BPR+ uses timestamps to calculate the average experienced delay of backlogged packets.

On each packet departure, service rates r_{TS} and r_{DS} that satisfy $\frac{\bar{d}_{TS}}{\bar{d}_{DS}} = \delta$ and $r_{TS} + r_{DS} = C$ are calculated from:

$$\begin{aligned} & r_{TS}^2 + \left(\frac{Q_{TS} + \delta Q_{DS}}{D_{TS} - \delta D_{DS}} - C \right) r_{TS} - \frac{Q_{TS} C}{D_{TS} - \delta D_{DS}} = 0 \\ & r_{DS} = C - r_{TS} \end{aligned} . \tag{22}$$

Packets are scheduled based on virtual service functions $V_{TS}(t)$ and $V_{DS}(t)$. These functions approximate the difference between an amount of traffic that would have been transmitted from TS and DS queues during the current busy period if these queues were serviced by rates r_{TS} and r_{DS}, respectively and an amount of traffic that has been actually transmitted from these queues. The virtual service functions are updated upon every

packet departure as:

$$V_{TS}(t) = \begin{cases} 0 & if\ a_{TS} \geq t^d \\ V_{TS}(t^-) + r_{TS}(t)(t - t^d) & if\ a_{TS} < t^d \end{cases} \tag{23}$$

$$V_{DS}(t) = \begin{cases} 0 & if\ a_{DS} \geq t^d \\ V_{DS}(t^-) + r_{DS}(t)(t - t^d) & if\ a_{DS} < t^d \end{cases}, \tag{24}$$

where $r_{TS}(t)$ and $r_{DS}(t)$ are service rates obtained from (22), t^d is the time of the previous departure, and a_{TS} and a_{DS} are arrival times of head-of-line packets in TS and DS queues, respectively. On each packet departure from TS queue, $V_{TS}(t)$ is updated as:

$$V_{TS}(t) = V_{TS}(t^-) - l_{TS}, \tag{25}$$

where l_{TS} is the length of the departed packet. Similarly, on each departure from DS queue, $V_{DS}(t)$ is updated as:

$$V_{DS}(t) = V_{DS}(t^-) - l_{DS}, \tag{26}$$

where l_{DS} is the length of the departed packet.

After the virtual service functions have been updated, BPR$^+$ scheduler chooses the next packet to be transmitted. If L_{TS} and L_{DS} are lengths of the head-of line packets in TS and DS queues, respectively, TS queue is serviced if:

$$L_{TS} - V_{TS}(t) < L_{DS} - V_{DS}(t). \tag{27}$$

Otherwise, DS queue is serviced. Pseudocode for the heuristic approximation of BPR$^+$ is given in Appendix A.

There are other options for packet scheduling that may not use the virtual service functions $V_{TS}(t)$ and $V_{DS}(t)$. One solution would be to randomly choose between TS and

DS queue with probabilities r_{TS}/C and r_{DS}/C, respectively.

5.3. Performances of BPR and BPR$^+$

In the following simulation scenarios, we evaluate the ability of BPR scheduler to provide proportional delay differentiation between the throughput-sensitive and delay-sensitive class. We compare its performance with the performance of BPR$^+$ in order to evaluate the performance improvement that could be achieved by considering arrival times of backlogged packets.

The simulated topology in ns-2 network simulator [31] is shown in Fig. 11. There are $N_{TS}=5$ throughput-sensitive sources and $N_{DS}=5$ delay-sensitive sources sending their packets to the sink. All sources are ON/OFF Pareto sources with average packet size of 500 bytes, average durations of ON and OFF periods 50 ms, and shape parameter 1.5. Capacities and propagation delays of links are indicated in Fig. 11. BPR and BPR$^+$ are implemented in the access router *R*. The capacity of the output buffer in the router is 250 packets, except for the scenario where we vary the buffer size.

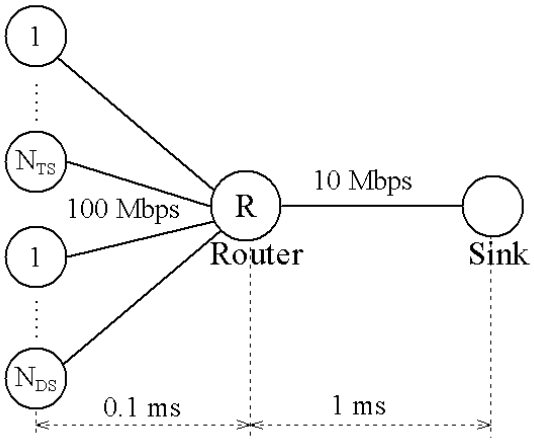

Fig. 11. Simulated topology used for evaluating the performances of BPR and BPR$^+$.

5.3.1. Influence of traffic load

In this simulation scenario we varied the traffic load by changing the sending rate of Pareto traffic sources during ON periods. The traffic load has been varied from 70 % to 120 % of the output link's capacity. Results for various delay differentiation parameters δ are shown in Fig. 12.

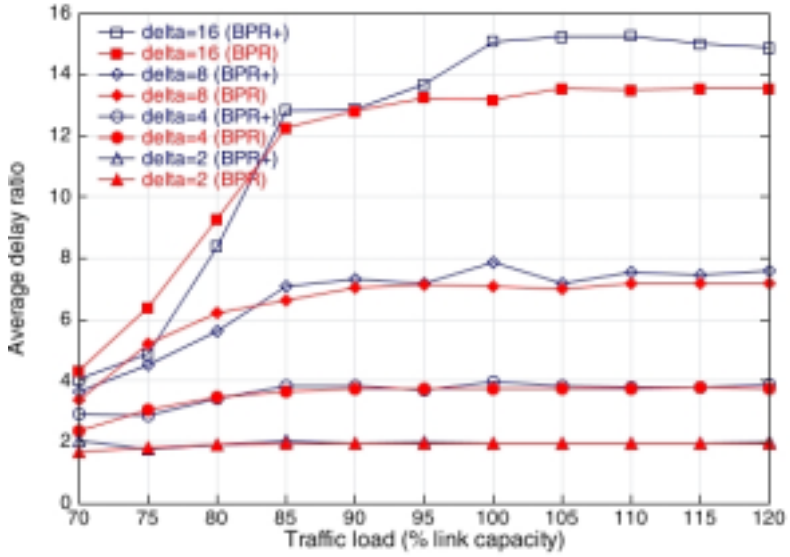

Fig. 12. Average delay ratio achieved by BPR and BPR$^+$ for various traffic loads and delay differentiation parameters δ.

BPR$^+$ is able to provide more precise delay differentiation than BPR for large values of δ (δ>8). Ratio of average delays in TS and DS queues is proportional to the ratio of queue lengths. Hence, for large δ number of packets in the DS queue is relatively small compared to the number of packet in the TS queue. The lesser the number of packets in a queue, the more important is when those packets arrived. Since, unlike BPR, BPR$^+$ considers the arrival times of packets to make a scheduling decision, it performs better for large values of δ. For δ<8 performance gain of BPR$^+$ compared to BPR is minor. Both BPR and BPR$^+$ perform better in a heavily congested network. This result is

expected since both BPR and BPR$^+$ rely on the information about the backlogged packets to make a scheduling decision. When a network is underutilized, buffers in routers are empty most of the time and schedulers are not able to provide desired delay differentiation. However, in the case of an underutilized network delay differentiation is not needed because all packets experience low delay.

5.3.2. Influence of load distribution

In this scenario we varied the load distribution between throughput-sensitive and delay-sensitive classes while maintaining the total load constant and equal to 100 % of the output link's capacity. Achieved average delay ratios are shown in Fig. 13.

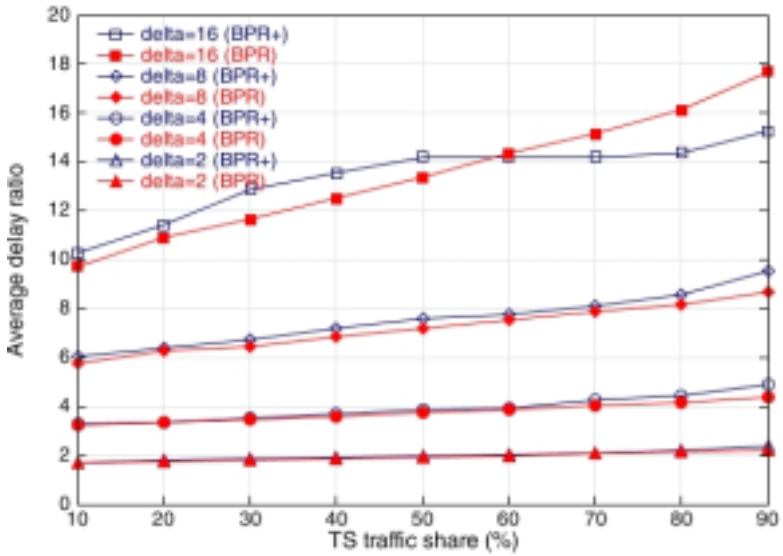

Fig. 13. Average delay ratio achieved by BPR and BPR$^+$ for various traffic load distributions and delay differentiation parameters δ.

It would be ideal if the achieved average delay ratio would not depend on the load distribution between classes. However, both BPR and BPR$^+$ are, to a certain degree, dependant on the traffic load distribution. The traffic class that constitutes larger portion of the total traffic load is likely to experience higher delay than in the case of ideal delay

differentiation. Similar result has been reported in [12]. The most likely reason for this is certain bias toward the class with a smaller number of packets in the buffer, which is inherent to approximations that are introduced in order to adapt BPR and BPR$^+$ to packetized traffic. For larger values of δ (δ>8), BPR$^+$ is able to provide a steady delay ratio over a wider range of load distributions than BPR. Reasons for better performance of BPR$^+$ for large values of δ have been discussed in Section 5.3.1.

5.3.3. Influence of buffer size

In this scenario, we varied the buffer size from 150 packets (60 ms of buffering on a 10 Mb/s link) to 350 packets (140 ms). Current trend in dimensioning buffers in Internet routers is to provide approximately 100 ms of buffering. Simulation results shown in Fig. 14 indicate that the buffer size does not affect the performance of BPR and BPR$^+$.

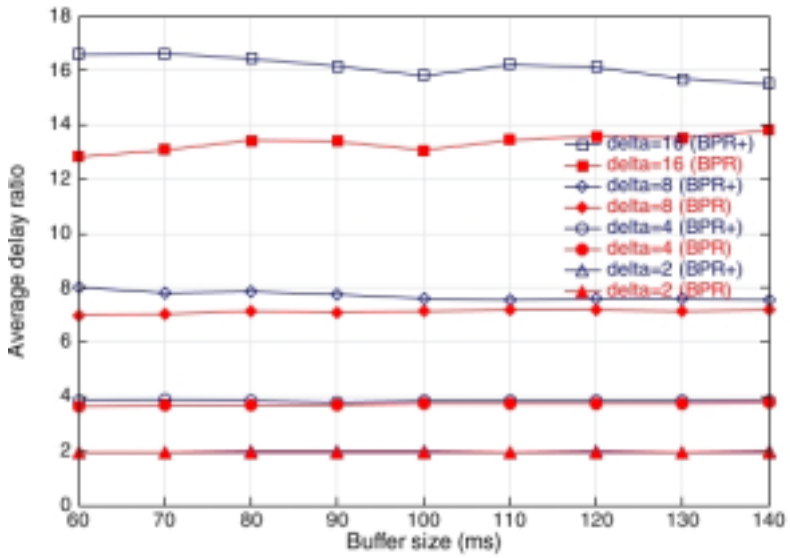

Fig. 14. Average delay ratio achieved by BPR and BPR$^+$ for various buffer sizes and delay differentiation parameters δ.

Extremely small buffer sizes might affect the ability of these schedulers to provide desired delay differentiation because both schedulers rely on information about backlog to make a scheduling decision. We did not include such an unrealistic scenario in our simulations.

5.3.4. Influence of timescale

The results in the previous scenarios are obtained by averaging achieved delay ratios over entire simulation run. In order to evaluate the performance of BPR and BPR$^+$ on shorter timescales, we created an additional scenario where the timescales are expressed in terms of *packet time constants*. In our case, one packet time constant is equal to the transmission time of a 500-byte packet on a 10 Mb/s link. For example, Fig. 15 shows queuing delays for throughput-sensitive and delay-sensitive classes averaged over 1,000 packet time constants (0.4 s) for δ=4 and 100 % utilization. On this timescale, both BPR and BPR$^+$ are able to provide consistent delay differentiation, i.e., throughput-sensitive traffic always experiences higher delay than delay-sensitive traffic.

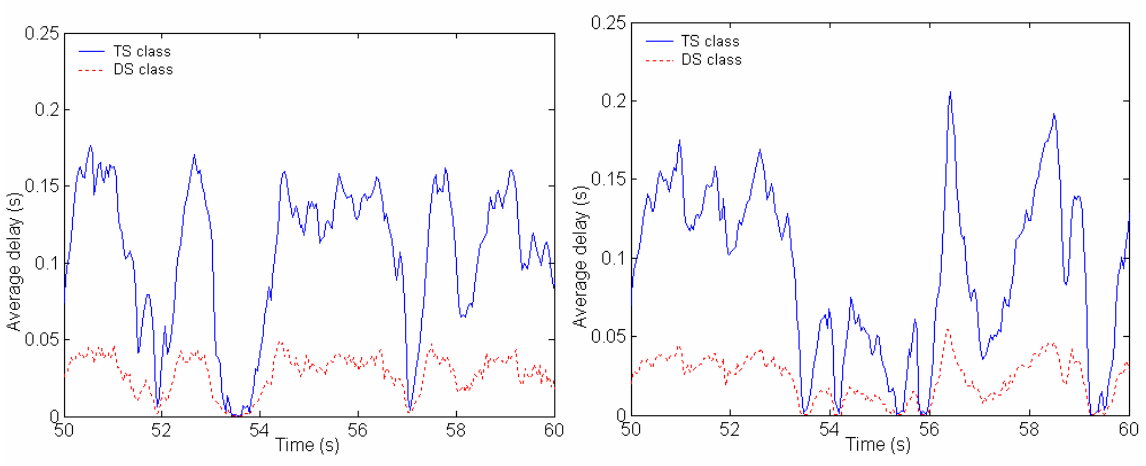

Fig. 15. Queuing delays of throughput-sensitive and delay-sensitive packets averaged over 1,000 packet time constants for BPR (left) and BPR$^+$ scheduler (right) and δ=4.

In order to investigate the performance of the schedulers on various timescales, the entire simulation run is divided to a number of intervals whose duration is equal to the monitoring timescale. Average delay ratios are calculated for each interval. The procedure has been repeated for four different timescales, corresponding to 10, 100, 1,000, and 10,000 packet time constants. Fig. 16 shows 5 %, 25 %, 50 %, 75 %, and 95 % percentiles for calculated ratios.

As the monitoring timescale increases, the percentiles for average delay ratio converge to the specified value δ=4. Both BPR and BPR$^+$ provide more precise delay differentiation on coarser timescales because these schedulers aim to provide specified ratio of *average* delays. Certain schedulers, such as Waiting-Time Priority (WTP) [12], aim to provide specified delay ratio on packet-by-packet basis. For shorter timescales, BPR$^+$ performs better than BPR because it uses more complete information about backlogged packets to make a scheduling decision.

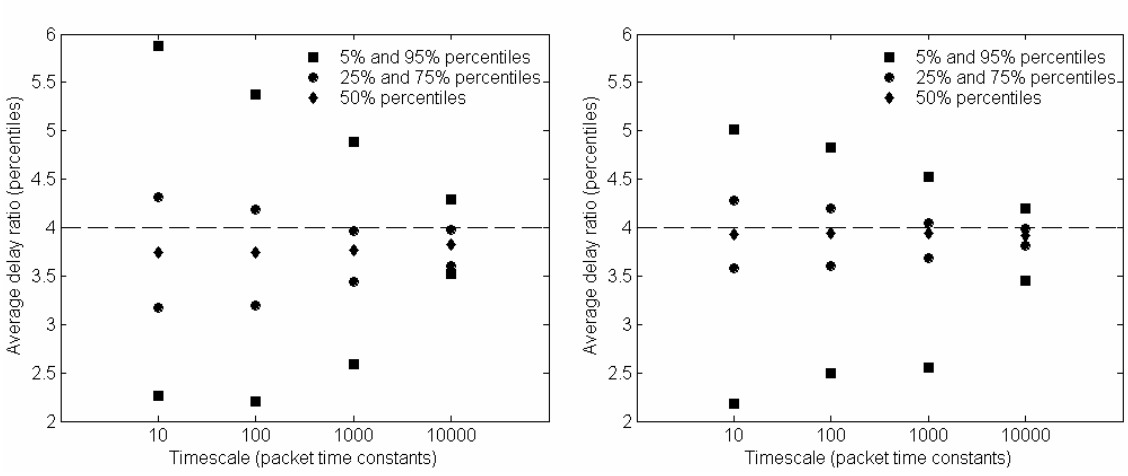

Fig. 16. Percentiles for average delay ratio on four different timescales for BPR (left) and BPR$^+$ scheduler (right) and δ=4.

Previous simulation results imply that BPR is able to provide proportional delay differentiation between throughput-sensitive and delay-sensitive class. Also, since the performance of BPR is very close to BPR^+, considering arrival times of backlogged packets would not improve the performance of BPR significantly. We believe that BPR may be improved by exploring new approaches to "packetize" the fluid BPR server rather than by adding more complexity to its rate assignment mechanism.

5.4. Influence of proportional delay differentiation on TCP throughput

TCP is a complex protocol whose performance depends both on packet delay (round-trip time) and packet loss probability. Packet schedulers for proportional delay differentiation are able to provide low delay service for delay-sensitive flows at the expense of increasing delay for throughput-sensitive TCP flows. Consequently, the round-trip time of TCP flows will be increased as well. A simple steady-state model for TCP throughput has been derived in [34], [35]:

$$T(p,R) = \sqrt{\frac{3}{2}} \frac{M}{R\sqrt{p}}, \tag{28}$$

where M is the average packet size, R is the round-trip time, and p is the loss probability of a TCP connection. Hence, throughput of TCP flows is affected by proportional delay differentiation because round-trip time of TCP packets increases.

We illustrate the problem in a simple ns-2 scenario. In this scenario, an equal number of TCP and UDP flows share the output link of a router (Fig. 17). We assume that the router employs Random Early Detection (RED) [29] as a buffer management algorithm. In addition to the Backlog-Proportional Rate (BPR), we consider several

schedulers for proportional delay differentiation: Dynamic Weighted Fair Queuing (DWFQ) [14], Mean-Delay Proportional (MDP) [13], Proportional Average Delay (PAD) [12], and Waiting-Time Priority (WTP) [12], [24]. Delay differentiation parameters of these schedulers have been varied from 1 to 16. Achieved delay ratio between TCP and UDP packets and throughput of TCP flows are shown in Fig. 18 and Fig. 19, respectively. In Fig. 19, the TCP throughput achieved in the absence of delay differentiation (i.e., FIFO scheduling) has been also indicated.

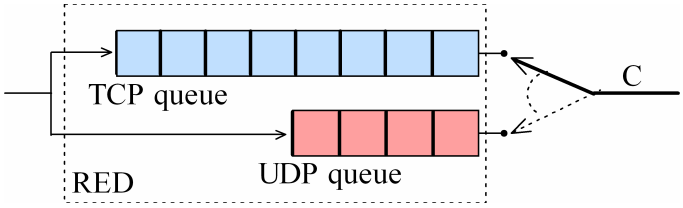

Fig. 17. An ns-2 scenario: TCP and UDP packets are served by a scheduler for proportional delay differentiation.

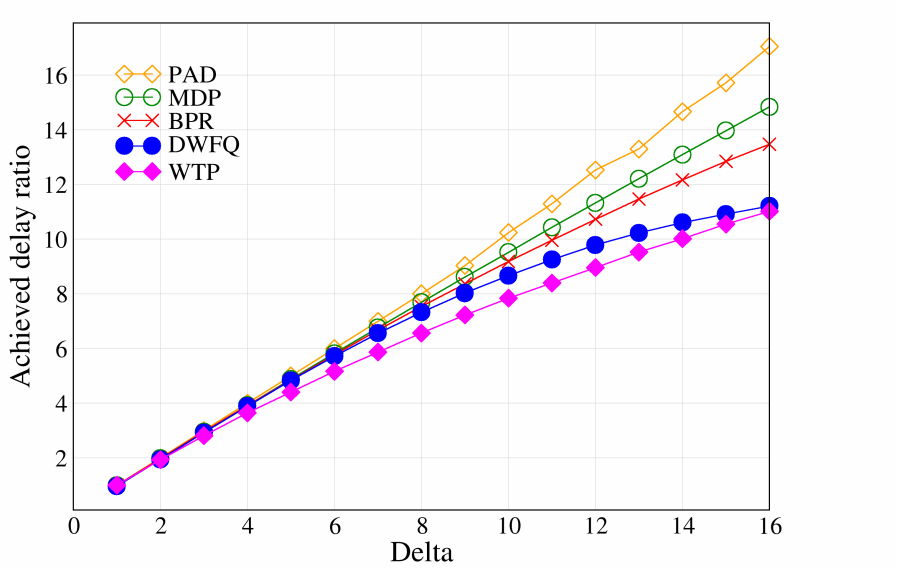

Fig. 18. Achieved delay ratio between TCP and UDP packets for various delay differentiation parameters δ.

While all these scheduling schemes are able to provide desired delay differentiation, they all reduce the throughput achieved by TCP flows. Significant

research efforts have been focused on making buffer management algorithms more TCP-friendly (i.e., RED has been widely accepted and implemented by major router vendors). Nevertheless, existing packet schedulers for delay differentiation are still not aware of specific requirements that arose from ubiquitous deployment of TCP protocol. They are not able to deal with performance deterioration that TCP applications may experience in the presence of delay differentiation. In Section 6, we present the Packet Loss Controller (PLC), a new mechanism that interacts with RED in order to alleviate this problem.

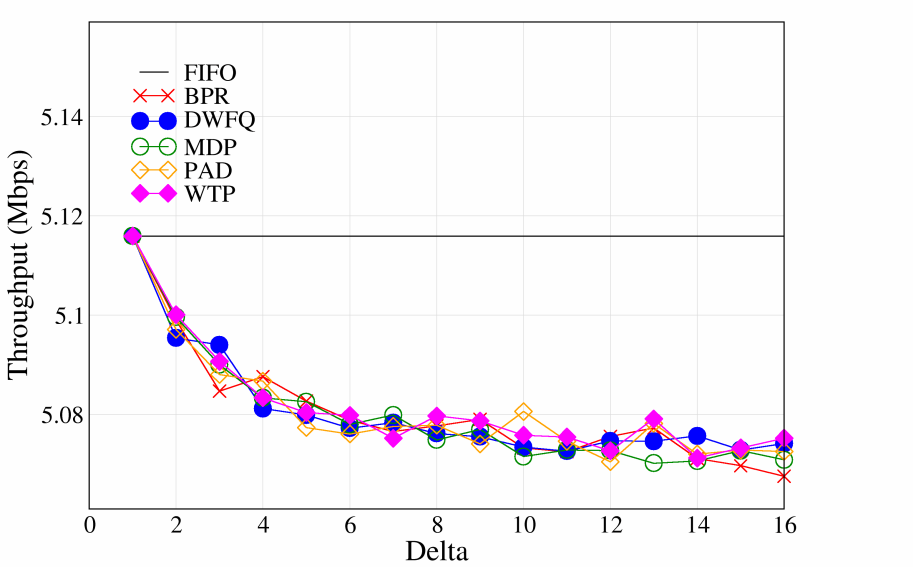

Fig. 19. Throughput achieved by TCP flows for various delay differentiation parameters δ.

6. Packet Loss Controller

Packet Loss Controller (PLC) is a new mechanism that regulates the throughput of TCP flows in the presence of proportional delay differentiation by controlling the packet loss probability of these flows. By adjusting the packet loss probability, PLC is able to compensate for increase in round-trip time that might affect the throughput of TCP flows. PLC interacts with Random Early Drop (RED) and Backlog Proportional Rate (BPR) to ensure that the second objective of the proposed architecture for service differentiation is satisfied. Its role is to provide at least the same throughput to throughput-sensitive (TCP) flows, as they would receive in a best-effort network.

Before we present the details of PLC algorithm, we introduce two assumptions. Let us assume that TCP throughput is a function of round-trip time R and packet loss probability p similar to (28). Suppose that loss probability

$$p_{tcp} = \sigma\, p\,, \tag{29}$$

where $0 \le \sigma \le 1$ is a PLC's loss differentiation parameter and p is the loss probability calculated by RED, compensates for the effect of increased round-trip time R. PLC interacts with RED in order to provide the desired loss probability p_{tcp}. After a packet has been marked for dropping by RED algorithm, PLC drops the packet with probability σ ($0 \le \sigma \le 1$). Otherwise, if PLC decides to rescue the packet, it is admitted to the TCP queue regardless of being marked by RED and a packet from the UDP queue is dropped instead. If UDP queue is empty, the TCP packet has to be dropped. PLC decreases the loss probability for TCP packets and increases the loss probability for UDP packets,

while maintaining the total loss probability p to be the same as without PLC. Hence, total number of dropped packets remains the same as it would be without PLC.

TCP congestion control algorithm, RED, BPR, and PLC are shown in Fig. 20 as a feedback control system.

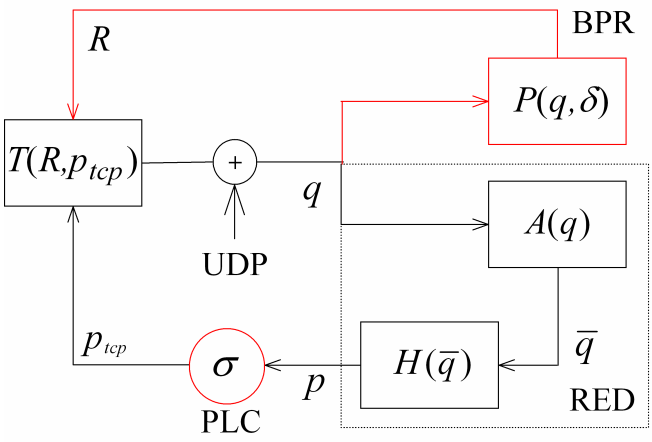

Fig. 20. TCP, RED, and PLC as a feedback control system.

While described PLC mechanism seems to be simple, choosing parameter σ is not so straightforward. Before we present an algorithm for setting the PLC parameter σ, we describe the elements of the system shown in Fig. 20.

We model the relationship between TCP throughput T, round-trip time R, and packet loss probability p as:

$$T(R, p_{tcp}) \sim \frac{1}{R^a p_{tcp}^b}, \tag{30}$$

where a and b are model parameters. A procedure used for identification of these parameters is presented in Appendix B. Random Early Detection (RED) algorithm tends to randomize packet losses and spreads them uniformly over a time period ("Method 2"

described by Floyd and Jacobson [29]). Hence, in Appendix B we assume uniformly distributed random packet losses when deriving parameters *a* and *b*.

Why do we introduce a new model when (28) proved to be fairly successful in modelling the steady-state throughput of TCP flows? In order to identify the parameter σ that compensates for the effect of increased round-trip time, we need mathematical relationships among various elements and variables of the system shown in Fig. 20. Finding a relationship between steady-state TCP throughput and packet loss probability requires a TCP model in terms of *packet loss probability*. However, the well-known TCP model (28) is given in terms of *probability of congestion events*. Parameter *p* in (28) is not a packet loss probability, but rather a number of congestion events per acknowledged packet [34], [35]. A group of subsequent packet losses that causes a reduction of the window size is considered to be a congestion event. Finding the relationship between packet loss probability and the probability of congestion events is not a trivial problem. Hence, instead of (28), we use an empirical model of TCP throughput. Variable p_{tcp} (30) corresponds to the packet loss probability observed in the system. Appendix B provides details how to derive (30). We briefly describe here remaining elements shown in Fig. 20.

As mentioned in Section 5.1, BPR assigns class service rates (r_{tcp} and r_{udp}) proportionally to queue lengths (q_{tcp} and q_{udp}) and a desired delay differentiation parameter $\delta \geq 1$:

$$\frac{r_{tcp}}{r_{udp}} = \frac{1}{\delta}\frac{q_{tcp}}{q_{udp}}. \tag{31}$$

Since $C=r_{tcp}+ r_{udp}$ is the link capacity, service rates r_{tcp} and r_{udp} can be calculated from (31). The ratio of delays experienced by TCP and UDP packets approaches δ, as shown in [12]:

$$\frac{d_{tcp}}{d_{udp}} \to \delta . \tag{32}$$

The total queue size q indicated in Fig. 20. is the sum of TCP and UDP queue sizes. Average total queue size $\overline{q}_k = A(q_k)$ is a weighted moving average of the total queue size:

$$A(q_k) = (1-w)\overline{q}_{k-1} + wq_k , \tag{33}$$

where w is the queue weight and k is packet number. RED calculates drop probability $p = H(\overline{q})$ as:

$$p = \begin{cases} 0 & if\ \overline{q} \le q_{\min} \\ \dfrac{\overline{q} - q_{\min}}{q_{\max} - q_{\min}} p_{\max} & if\ q_{\min} < \overline{q} < q_{\max} \\ 1 & if\ \overline{q} \ge q_{\max} \end{cases}, \tag{34}$$

where q_{min}, q_{max}, and p_{max} are control parameters of the RED algorithm. The drop probability p is modified by PLC and the loss probability experienced by TCP flows becomes $p_{tcp}=\sigma p$.

In order to provide at least the same throughput for TCP traffic as in a best-effort network, σ should satisfy:

$$\frac{1}{R^a(\sigma p)^b} \ge \frac{1}{R'^a p'^b}, \tag{35}$$

where *R'* and *p'* are round-trip time and loss probability, respectively, in the absence of service differentiation. Rearranging (35) gives:

$$\sigma \le \frac{p'}{p}\left(\frac{R'}{R}\right)^{x}, \tag{36}$$

where $x=a/b$. Hence, the relationship between p, p', R, and R' defines a range of σ that satisfy our service differentiation requirements.

The round-trip time $R=P(q,\delta)$ of a TCP flow is a sum of transmission, propagation, and queuing delays along the path. We introduce two assumptions: we consider mixed TCP and UDP flows sharing a single bottleneck link and we assume that transmission and propagation delay are negligible compared to the queuing delay. This is a realistic assumption in a high-speed network during congestion periods. Service differentiation is not an issue if the network is underutilized. As a result of these assumptions, round-trip times of TCP flows approximated by the queuing delay are:

$$R \approx \frac{q_{tcp}}{r_{tcp}}, \tag{37}$$

where q_{tcp} is the length of the TCP queue and r_{tcp} is the service rate assigned to TCP flows by BPR. Since the service rate allocation in BPR satisfies (31) and $q=q_{tcp}+q_{udp}$, the round-trip time R is:

$$R \approx \frac{q}{r_{tcp}+\frac{r_{udp}}{\delta}}. \tag{38}$$

Similarly, the round-trip time of TCP flows in a best-effort network that employs FIFO scheduling is:

$$R' \approx \frac{q'}{C}, \tag{39}$$

where q' is the length of RED queue in the absence of service differentiation. Substituting (38) and (39) in (36) gives:

$$\sigma \le \frac{p'}{p}\left(\frac{q'}{q}\right)^x \eta^x, \tag{40}$$

where:

$$\eta = \frac{r_{tcp} + \frac{r_{udp}}{\delta}}{C}. \tag{41}$$

Let σ_0 be the value of σ that provides identical throughput to TCP flows as in a best-effort network:

$$\sigma_0 = \frac{p'}{p}\left(\frac{q'}{q}\right)^x \eta^x. \tag{42}$$

Next step in deriving σ_0 is finding the relationship between p and p'. If λ is the aggregate sending rate of UDP flows, p_{udp} loss probability of UDP packets, and T throughput of TCP flows, then:

$$T + (1 - p_{udp})\lambda = C. \tag{43}$$

Similarly, in a best-effort network:

$$T' + (1 - p')\lambda = C. \tag{44}$$

Since $\sigma=\sigma_0$ ensures that $T=T'$, it follows from (43) and (44) that $p_{udp}=p'$. In order to maintain the same overall loss probability as calculated by RED, PLC drops one UDP packet for each "rescued" TCP packet. Hence:

$$(p - p_{tcp})\frac{r_{tcp}}{1 - p_{tcp}} = (p_{udp} - p)\frac{r_{udp}}{1 - p_{udp}}. \tag{45}$$

Taking into account that $p'=p_{udp}$ gives:

$$p' = p\frac{r_{udp}(1-\sigma_0\ p) + r_{tcp}(1-\sigma_0)}{r_{udp}(1-\sigma_0\ p) + r_{tcp}(1-\sigma_0)p}. \tag{46}$$

In order to simplify (46), we consider three cases based on the average total queue size $\overline{q}$.

6.1. No-loss region: $\overline{q} \le q_{\min}$

In this case p=0, which makes value of σ irrelevant based on (29).

6.2. Early-drop region: $q_{\min} < \overline{q} < q_{\max}$

Average queue sizes between q_{min} and q_{max} indicate that RED operates in a "early drop" region. In that case, drop probability p satisfies $p<p_{max}<<1$. Hence, (46) can be simplified to:

$$p' = p\left(1 + \frac{r_{tcp}}{r_{udp}}(1-\sigma_0)\right). \tag{47}$$

However, in order to solve (42), we need to find the relation between q and q'. Therefore, we introduce an additional approximation: we assume that the ratio of instantaneous

queue sizes can be approximated by the ratio of average queue sizes, which is true for small values of the queue weight *w*:

$$\frac{q'}{q} \approx \frac{\overline{q}'}{\overline{q}}. \tag{48}$$

The relation between the average queue sizes can be derived from (34):

$$\frac{q'}{q} \approx \frac{p' + \xi}{p + \xi}, \tag{49}$$

where:

$$\xi = \frac{p_{\max} q_{\min}}{q_{\max} - q_{\min}}. \tag{50}$$

After substituting (47) and (49) into (42), σ_0 becomes:

$$\sigma_0 = \left(1 + \frac{r_{tcp}}{r_{udp}}(1 - \sigma_0)\right)\left(1 + \gamma \frac{r_{tcp}}{r_{udp}}(1 - \sigma_0)\right)^x \eta^x, \tag{51}$$

where $\gamma = p/(p + \xi)$. Equation (51) can be reduced to a first order equation because several of its parameters are constrained. Rule of thumb for setting RED thresholds is $q_{\max} / q_{\min} = 3$ [41]. In that case, (50) reduces to $\xi = p_{\max} / 2$. Hence:

$$0 \le \gamma \le \frac{2}{3}. \tag{52}$$

We prove in Appendix C that:

$$0 \le \frac{r_{tcp}}{r_{udp}}(1 - \sigma_0) < \frac{2}{3}. \tag{53}$$

Therefore, using Taylor's theorem we obtain:

$$\left(1+\gamma\frac{r_{tcp}}{r_{udp}}(1-\sigma_0)\right)^x \approx 1+\gamma x\frac{r_{tcp}}{r_{udp}}(1-\sigma_0)\,. \tag{54}$$

Based on (51) - (54), σ_0 becomes:

$$\sigma_0 \approx \left(1+\frac{r_{tcp}}{r_{udp}}(1+\gamma x)(1-\sigma_0)\right)\eta^x\,. \tag{55}$$

Finally:

$$\sigma_0 \approx \frac{1+(1+\gamma x)\,\dfrac{r_{tcp}}{r_{udp}}}{\dfrac{1}{\eta^x}+(1+\gamma x)\,\dfrac{r_{tcp}}{r_{udp}}}\,. \tag{56}$$

6.3. Forced-drop region: $\overline{q} \geq q_{\max}$

In this case, the drop probability is p=1 (34). By substituting p=1 in (46), we obtain:

$$p' = p\,. \tag{57}$$

Therefore, the average queue size in the absence of service differentiation is $\overline{q}' \geq q_{\max}$. This interval of average queue sizes is called "forced-drop" region. It is not a steady-state operating region of RED because drop probability p=1 (p'=1) causes a rapid decrease of the average queue size $\overline{q}$ ($\overline{q}'$) below $q_{\max}$. Hence:

$$q_{\max} \leq \overline{q}' \approx \overline{q} < q_{\max} + \varepsilon\,, \tag{58}$$

where $\varepsilon \to 0$. From (42), (57), and (58) it follows that:

$$\sigma_0 \approx \eta^x\,. \tag{59}$$

In summary, PLC parameter σ that provides at least the same throughput to TCP flows in the presence of proportional delay differentiation as in a best-effort network is set by the router based on the operating region of RED: σ is given by (56) in the early-drop region and (59) in the forced-drop region.

Implementation complexity introduced by the proposed PLC mechanism, compared to the case of BPR scheduling without PLC, is in frequent recalculations of the parameter σ. However, this may not be an issue with today's RISC processors with clock rates of ~ 300 MHz.

7. Simulation results

We evaluated the performance of the proposed architecture for service differentiation in Internet by using the ns-2 network simulator [31]. Simulated topology is shown in Fig. 21. The number of TCP and UDP sources is N_{TCP}=10 and N_{UDP}=10, respectively. Packet sizes for all sources are M=500 bytes. We consider TCP New Reno, the most common flavour of TCP in today's Internet. UDP traffic consists of Pareto ON/OFF traffic generators with ON and OFF periods set to 50 ms and shape parameter 1.5. Parameters of RED deployed in routers R1 and R2 are (unless stated otherwise): buffer size B=250 packets, minimum threshold q_{min}=60 packets, maximum threshold q_{max}=180 packets, queue weight w=0.002, and maximum drop probability p_{max}=0.1. The capacities and propagation delays of links are indicated in Fig. 21.

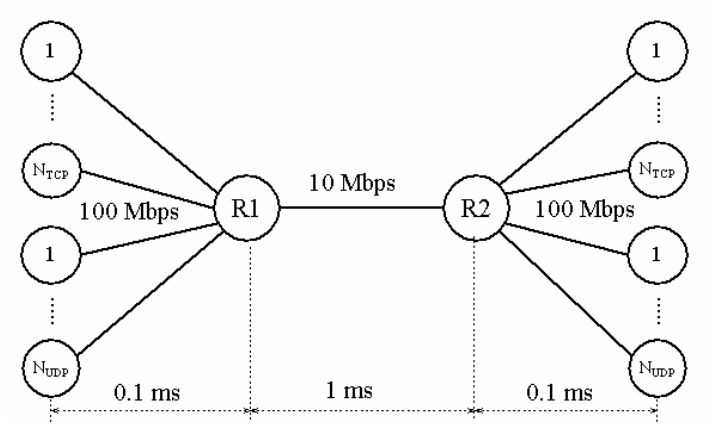

Fig. 21. Simulated network topology used for performance evaluation of the proposed service differentiation mechanism.

In the first simulation scenario, the aggregate UDP load is 5 Mb/s. BPR delay differentiation parameter is δ=8. We consider three cases based on scheduling mechanism

employed in routers: FIFO scheduling (best-effort), BPR scheduling, and the proposed BPR scheduling with PLC.

As shown in Fig. 22, in the case of FIFO scheduling, average delays of TCP and UDP packets are identical (43 ms). In the case of BPR scheduling, the average queuing delays of TCP and UDP packets are 65 ms and 8 ms, respectively. Hence, achieved delay ratio is close to the specified parameter δ=8. PLC controller does not affect packet delay, as shown in the case of BPR scheduling with PLC. Low delay service for real-time UDP flows is achieved at the expense of increasing the delay for TCP packets from 43 ms to 65 ms. Consequently, round-trip times of TCP packets are increased as well. The effect of proportional delay differentiation on aggregate throughput of TCP and UDP flows is shown in Fig. 23.

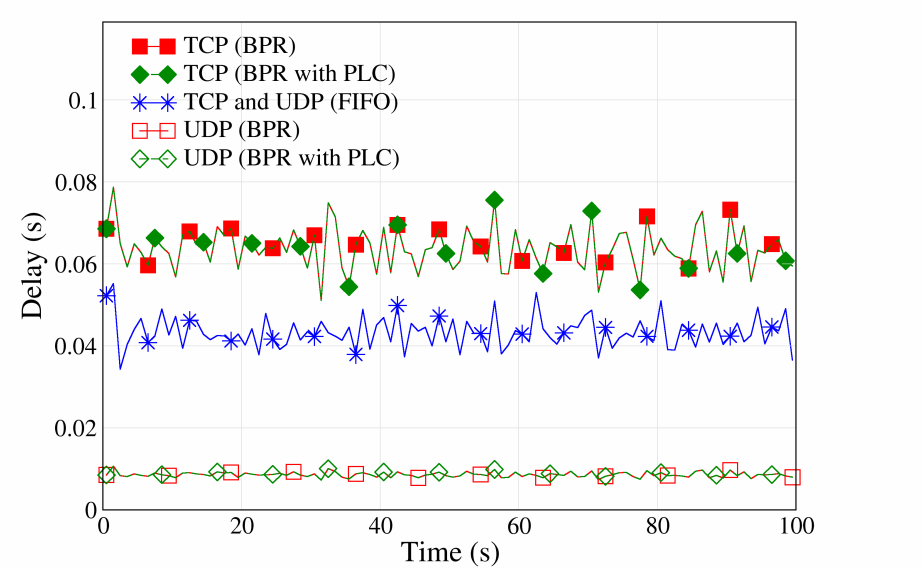

Fig. 22. Average queuing delay for TCP and UDP packets in the presence of service differentiation and in the best-effort (FIFO) case.

Proportional delay differentiation increases the average queuing delay and round-trip time of TCP packets, as shown in Fig. 22. Hence, according to (28), in the case of

BPR scheduling without PLC, TCP flows achieve lower throughput then in the best-effort (FIFO) case. Average TCP throughput is 5.13 Mb/s in the FIFO case, compared to 5.08 Mb/s in the case of BPR without PLC. Not only that UDP flows experience lower delay, but they also consume more bandwidth. The average UDP throughput is 4.87 Mb/s in the best-effort, compared to 4.93 Mb/s in the case of BPR without PLC. This behaviour contradicts the basic concept of non-elevated services, aiming to provide "different but equal" service to both traffic classes. However, in the presence of PLC, the throughput of TCP flows is approximately equal to the throughput achieved in the case of FIFO scheduling. Hence, BPR with PLC provides low delay service to UDP flows and "protects" the throughput of TCP flows.

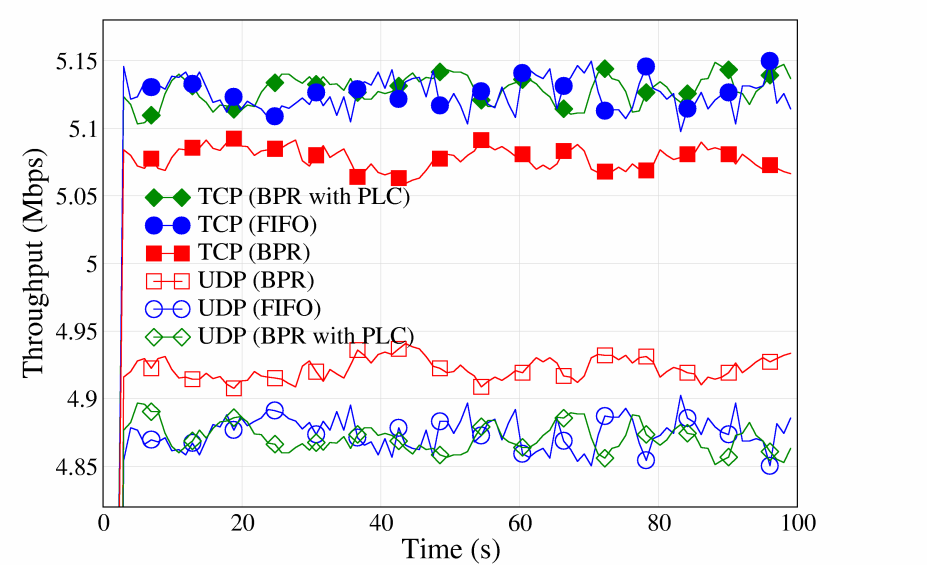

Fig. 23.Throughput of TCP and UDP flows in the presence of delay differentiation (with and without PLC) and in the best-effort (FIFO) case.

7.1. Influence of delay differentiation parameter δ

In the second simulation scenario, we evaluate the performance of PLC controller for various delay differentiation parameters δ. Achieved average delay ratio for various

specified δ is shown in Fig. 24. BPR maintains the specified ratio of delays for δ<8 (approximately). The slope of the curve starts to decrease for δ>8, indicating that BPR is unable to accurately provide desired delay ratio. For instance, achieved average delay ratio is 12.4 for δ=14. However, this discrepancy is acceptable because non-elevated service differentiation model is a lightweight approach that does not offer strict guarantees. Also, feasibility of the proportional delay differentiation becomes an issue for large values of δ. Minimal possible delay for UDP packets would be achieved if UDP packets were given strict priority over TCP packets. It has been shown [42] that average delay ratio δ between two traffic classes is feasible if the corresponding average delay ratio provided by a strict priority scheduler is not larger than the δ.

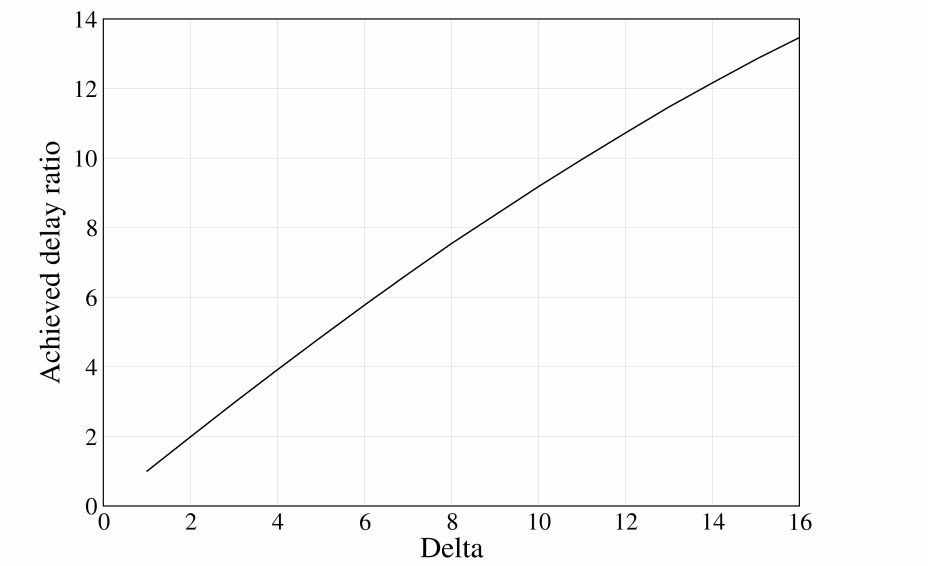

Fig. 24. Average delay ratio for TCP and UDP traffic as a function of specified delay ratio δ.

Throughput of TCP and UDP flows for various δ is shown in Fig. 25. TCP flows achieve lower throughput in the case of BPR scheduling than in the best-effort (FIFO) case. Hence, the throughput of TCP flows is affected by proportional delay differentiation. They receive lower throughput with higher delay ratio δ because the round-trip

time of TCP packets continually increases with δ. The negative effect of delay differentiation is eliminated in the case of BPR with PLC. PLC consistently provides approximately identical throughput to TCP flows as in the best-effort case, as shown in Fig. 25. This result confirms the effectiveness of the PLC controller for various δ.

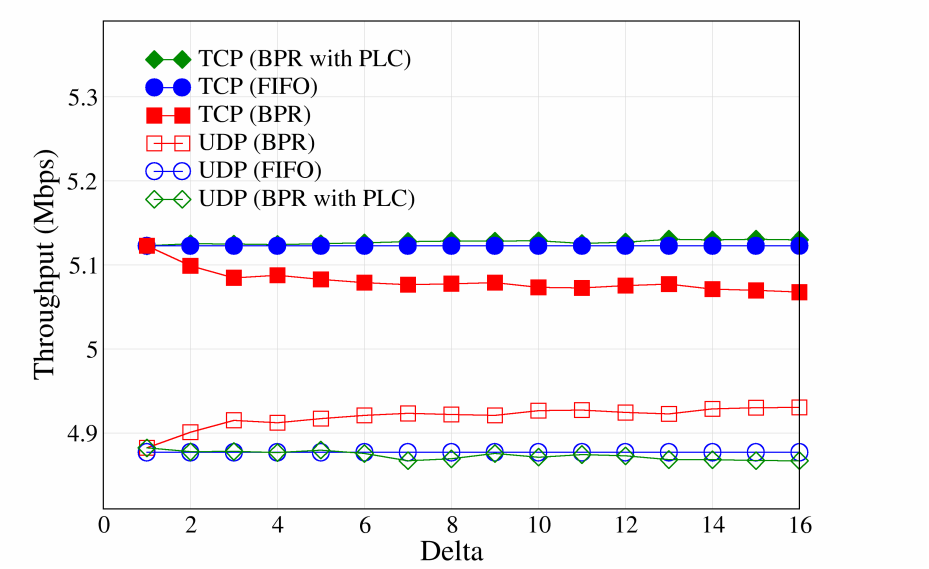

Fig. 25. Influence of delay differentiation parameter δ on TCP and UDP throughput in the presence of delay differentiation (with and without PLC) and in the best-effort (FIFO) case.

7.2. Influence of UDP load

In this simulation scenario, we varied the sending rate of UDP sources from 1 Mb/s to 12 Mb/s. BPR delay differentiation parameter is δ=8. As shown in Fig. 26, BPR approximates the specified ratio more closely for heavier UDP loads. Since BPR assigns service rates to traffic classes based on their buffer occupancy (10), it is not surprising that BPR performs better in a congested network when UDP load is heavy and packets are backlogged in queues. When a network is underutilized, delay differentiation is not necessary because queues are empty most of the time and queuing delay is zero.

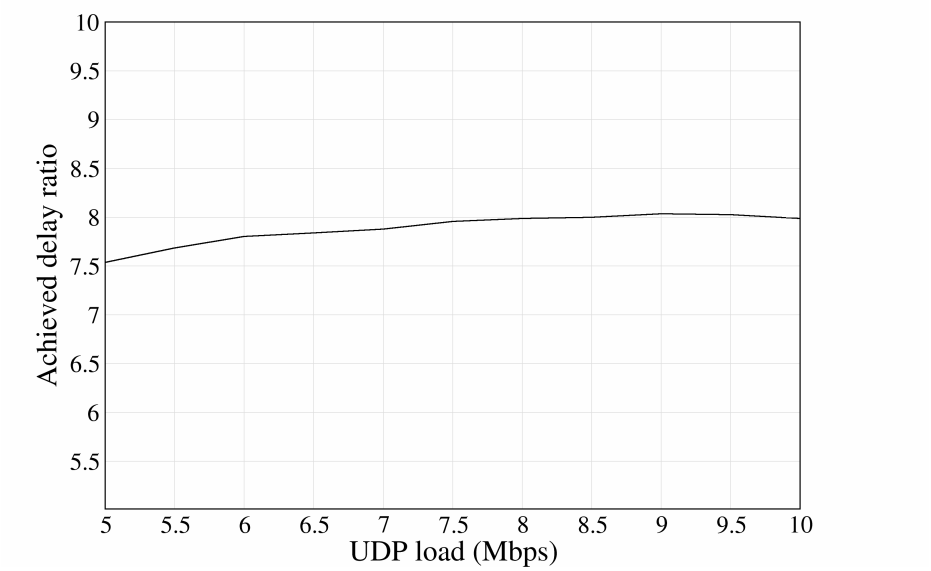

Fig. 26. Average delay ratio for TCP and UDP traffic for various UDP loads. Delay differentiation parameter δ=8.

Throughput of TCP flows for various UDP loads is shown in Fig. 27. BPR with PLC consistently provides at least the same throughput to TCP flows as in the FIFO case. PLC controller eliminates the negative effect of delay differentiation on TCP throughput that is present in case of BPR without PLC. We also observed the well-known problem of unfair bandwidth sharing between responsive TCP and unresponsive UDP flows. Since UDP flows do not decrease their sending rate in response to congestion, they tend to monopolize the link capacity as their load increases. In order to solve this problem, additional mechanisms are needed [43]-[50]. Even though not designed to deal with the unfairness problem, PLC controller attenuates the starving effect on TCP flows. As shown in Fig. 27, when UDP load is 12 Mb/s, TCP flows still achieve approximately 1 Mb/s in the case of BPR with PLC. In the case of BPR without PLC, TCP flows receive no throughput.

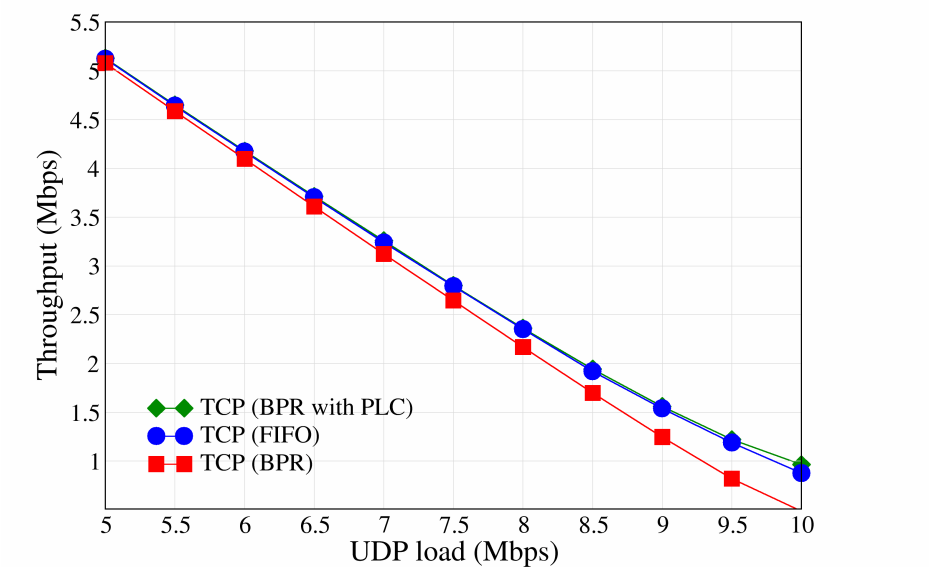

Fig. 27. Influence of UDP load on TCP throughput in the presence of delay differentiation (with and without PLC) and in the best-effort (FIFO) case.

7.3. Multi-hop topology scenario

The proposed architecture is designed to provide local (per-hop) service differentiation. However, from the user's perspective, end-to-end performance of an application is the only relevant indicator of the QoS received from the network. In this scenario, we evaluate the end-to-end performance of the proposed architecture.

Simulation topology is shown in Fig. 28. Traffic load consists of four groups of TCP flows and four groups of UDP flows, with 25 flows in each group. The UDP flows follow the Pareto distribution with the same parameters as in the previous simulation scenarios (Sections 7.1 and 7.2) , except for the average sending rate during the ON period, which is 100 Kb/s. Capacities and propagation delays of all access links are 100 Mb/s and 0.1 ms, respectively. Propagation delays of all backbone links are 1 ms, while their capacities are indicated in the Fig. 28. The BPR delay differentiation parameter δ=4

in all routers. We measured the average end-to-end delay and throughput of TCP flows in each group. Results are shown in Table 2.

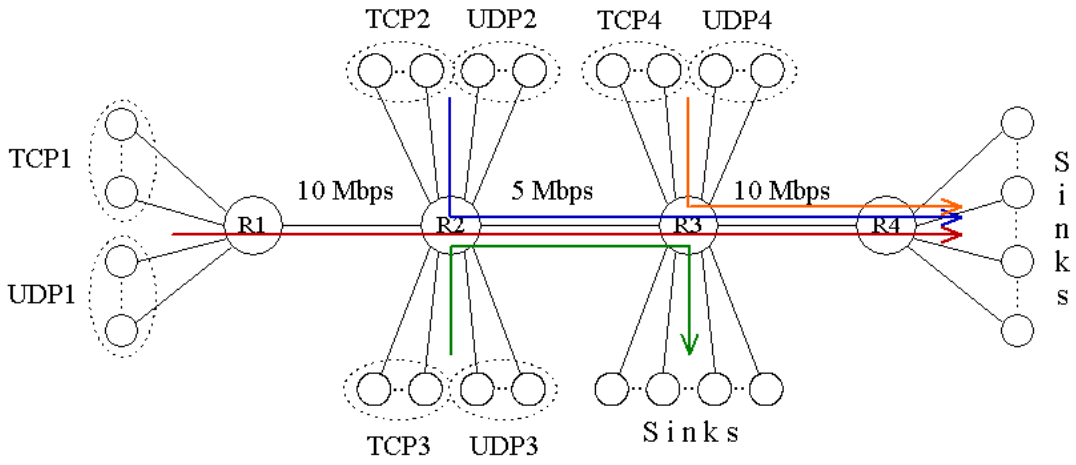

Fig. 28. Simulation topology used for end-to-end performance evaluation of the proposed architecture for service differentiation.

The achieved delay ratio is very close to the specified δ in all groups of flows. This result illustrates the fact that the per-hop proportional delay differentiation directly translates to end-to-end proportional delay differentiation, even in the case of non-homogeneous link capacities and cross traffic along the path. Similar observations have been reported in the case of Waiting-Time Priority (WTP) scheduler [12], [23]. In [12], authors noticed that the achieved end-to-end delay ratio becomes even closer to the specified ratio δ as the number of hops increases because the deviations from different hops tend to cancel-out.

The presented results also confirm that PLC is able to protect the throughput of TCP flows in presence of proportional delay differentiation. Although we derived the expression for PLC's parameter σ under the assumption of a single bottleneck in the network, our simulation results indicate that PLC provides desired service differentiation

even in the case of multiple bottlenecks. For instance, the first group of flows traverses three bottlenecks, as shown in Fig. 28. The link between routers *R1* and *R2* is the first bottleneck for these flows because its capacity is smaller than the capacity of access links. The link between routers *R2* and *R3* is the second bottleneck because its capacity is only one half of the previous link's capacity. Finally, these flows are additionally constrained by the capacity of the link between routers *R3* and *R4* because of the cross traffic from the fourth group of flows (Fig. 28). As shown in Table 2, throughput of the first group of TCP flows is higher in the case of BPR scheduling with PLC than in the FIFO case. However, performance analysis of the proposed architecture in case of multiple bottlenecks is a rather challenging problem. Analytically proving that the PLC provides consistent and predictable result in case of multiple bottlenecks is still an open issue.

Table 2. Achieved delay ratios and throughput for four groups of TCP flows in the case of FIFO scheduling and BPR scheduling with and without PLC.

	FIFO		BPR			BPR with PLC		
	d_{tcp}/d_{udp}	T (Mb/s)	d_{tcp}/d_{udp}	T (Mb/s)	Diff. (%)	d_{tcp}/d_{udp}	T (Mb/s)	Diff. (%)
TCP_1	0.9961	0.3594	3.7970	0.3312	-7.8464	3.9255	0.3748	+4.2849
TCP_2	0.9957	0.4166	3.8391	0.3753	-9.9136	3.9682	0.4331	+3.9606
TCP_3	0.9937	0.5979	3.8683	0.5801	-2.9771	4.0134	0.6159	+3.0105
TCP_4	0.9992	5.6269	3.7073	5.5189	-1.9194	3.7402	5.6940	+1.1925

8. Conclusions and open issues

Non-elevated service models emerged recently as new proposals for introducing service differentiation in Internet. These architectures aim to provide "different but equal" service to traffic classes and usually employ a trade-off between delay and loss probability to achieve desired service differentiation. However, existing mechanisms for delay and loss differentiation are not directly applicable to TCP, whose performance depends on both packet delay and loss rate.

In this thesis, we presented a new non-elevated service differentiation architecture that alleviates this problem and provides predictable and controllable performance of TCP flows in the presence of low-delay UDP traffic. Objectives of the proposed architecture are to provide low delay to delay-sensitive UDP applications and at least the same throughput to throughput-sensitive TCP applications as they would receive in a best-effort network. Hence, both throughput-sensitive (TCP) and delay-sensitive (UDP) classes benefit from the proposed mechanism. The new mechanism employs two basic elements: Backlog Proportional Rate (BPR) scheduler and Packet Loss Controller (PLC). BPR scheduler provides proportional delay differentiation between throughput-sensitive and delay-sensitive classes. PLC controller interacts with TCP's congestion control algorithm in order to protect the throughput of throughput-sensitive flows. We presented a procedure for setting the PLC's drop differentiation parameter σ in case of mixed TCP and UDP flows sharing a single bottleneck link. In order to derive the parameter σ that guarantees that the throughput of TCP flows will not be decreased in the presence of proportional delay differentiation, we considered BPR scheduler, RED queue

management, PLC controller and TCP's congestion control algorithm as a feedback control system. Using ns-2 simulation with various delay differentiation parameters δ, UDP loads, and topologies, we confirmed that the proposed differentiation architecture achieves the specified objectives. Since the traffic classes receive "different but equal" quality of service, proposed architecture can be classified as non-elevated. The proposed architecture is simple because it does not require admission control, traffic policing, and substantial operational changes in current Internet. Moreover, flat pricing currently used in Internet can be preserved.

In summary, contribution of this thesis has three aspects. First, it promotes non-elevated approach to service differentiation as a viable solution for QoS provisioning in Internet. Low implementation complexity, possibility of incremental deployment and flat pricing make non-elevated architectures especially appealing. Second, this thesis contributes to better understanding of interaction between delay and loss differentiation mechanisms and TCP's congestion control algorithm. Since Internet traffic is dominated by TCP, any solution that does not take into account TCP performance degradation is not likely to be accepted. Finally, this thesis presents a new architecture that enables predictable and controllable service differentiation between delay-sensitive (UDP) and throughput-sensitive (TCP) flows. The proposed architecture provides new services that are expected to replace current best-effort Internet service.

A number of open issues and areas for future work arise from this thesis. We particularly note the following:

The first and the most important concern is the ability of the proposed architecture to provide desired service differentiation in case of multiple bottlenecks in a network.

Although our simulation results are encouraging, further research is needed to confirm these results. Performance analysis of the proposed architecture in the case of multiple bottlenecks is a rather challenging problem.

A network administrator's strategy for setting the BPR's delay differentiation parameter δ is an open issue that requires knowledge of delay requirements of real-time applications used by a network. In addition to users' requirements, network administrator's strategy has to consider that not all values of δ are feasible. A feasibility study for BPR scheduler is necessary, similar to the study presented in [23] for the case of Time-Dependent Priorities (TDP) scheduler.

Finally, exploring other rate assignment strategies for proportional delay differentiation may eliminate some shortcomings of BPR, such as dependence on traffic load distribution between classes and inferior performance on short timescales. Furthermore, other AQM algorithms may be considered in addition to RED, especially those algorithms that aim to eliminate unfairness problem in Internet. In that case, the algorithm for setting the PLC's parameter σ should be revised.

Appendix A

The pseudocode for the heuristic approximation of fluid BPR$^+$ server [38] presented in Section 5 includes two functions. The first function is executed for each packet accepted to the buffer. It appends the current timestamp to the packet and places it to a corresponding queue. The second function is executed for each packet leaving the buffer and it decides which packet will be next serviced.

```
For each packet accepted to the buffer {
        TIME = current time;
        setTimestamp(pkt);
        if the packet belongs to the TS class {
                put packet in the TS queue;
                D_TS += TIME; Q_TS += length(pkt);
        if the packet belongs to the DS class {
                put packet in the DS queue;
                D_DS += TIME; Q_DS += length(pkt);
        }
}

For each packet leaving the buffer {
        TIME = current time;
        if the TS queue is empty and the DS queue is not empty {
                pkt = get a packet from the DS queue;
                D_DS -= getTimestamp(pkt); Q_DS -= N_DS * length(pkt);
                V_DS = 0; V_TS = 0;
        }
        if the TS queue is not empty and the DS queue is empty {
                pkt = get a packet from the TS queue;
                D_TS -= getTimestamp(pkt); Q_TS  -= N_TS * length(pkt);
                V_TS = 0; V_DS = 0;
        }
        if none of the queues is empty {
                C = capacity of the output link;
                N_TS = length of the TS queue;
                N_DS = length of the DS queue;
                Q_TS = Q_TS / N_TS; Q_DS = Q_DS / N_DS;
                D_TS = TIME – D_TS / N_TS; D_DS = TIME – D_DS / N_DS;
                // a, b, and c are the coefficients in (22)
                a= D_TS  - δ *D_DS;
                b = Q_TS + δ *Q_DS – C*a;
```

```
                c = - Q_TS*C;
                if (a != 0) {
                        r_TS = (- b + sqrt (b^2 - 4*a*c)) / (2*a);
                } else r_TS  = - c / b;
                r_DS = C – r_TS;
                if (timestamp of the HOL packet in the DS queue >= t^d) {
                        V_DS = 0;
                } else V_DS += r_DS * (TIME - t^d);
                if (timestamp of the HOL packet in the TS queue >= t^d) {
                        V_TS = 0;
                } else V_TS + = r_TS*(TIME - t^d);
                if (L_DS - V_DS  > L_TS - V_TS) {
                        pkt = get a packet from the TS queue;
                        D_TS -= getTimestamp(pkt); Q_TS -= N_TS * length(pkt);
                        V_TS = V_TS  - length(pkt);
                } else {
                        pkt = get a packet from the DS queue;
                        D_DS -= getTimestamp(pkt); Q_DS  -= N_DS * length(pkt);
                        V_DS = V_DS  - length(pkt);
                }
        }
        t^d = TIME;
}
```

Appendix B

We describe here a procedure used to derive an empirical model of steady-state throughput for long-lived TCP flows in presence of random independent packet losses. We assume a nonlinear dependence of TCP throughput on round-trip time (R) and loss probability (p):

$$T(R, p) \sim \frac{1}{R^a p^b}, \tag{60}$$

where a and b are model parameters. Although other functions may be used to describe relationship between $T(R, p)$, R, and p, (60) has been chosen because of its simplicity and consistency with previous TCP models. Mathis et al., [35] used a procedure similar to the one that we present to fit the model parameter k:

$$T(R, p) \sim \frac{1}{R p^k}, \tag{61}$$

based on ns-2 simulation results. Hence, linear dependence between $T(R, p)$ and R has been assumed and only one-dimensional fitting has been performed. In order to better match empirical results, we use two-dimensional fitting to determine parameters a and b of the empirical model (60). This model does not capture timeout events in TCP. We consider light to moderate loss probabilities ($p < 5\%$). Hence, it is realistic to assume that packet losses will be detected by triple duplicate ACKs, especially since we consider uncorrelated packet losses (i.e., low possibility of multiple packet losses from a same congestion window).

In order to identify model parameters (60), we consider ns-2 scenario presented in Fig. 29. The scenario includes four modules: sender, receiver, delay, and loss module. Sender module consists of 10 identical TCP NewReno sources. Delay module provides controllable round-trip time of TCP packets by varying propagation delays of links. Loss module is a random packet dropper with adjustable loss probability p. TCP flows are not restricted by capacity of the links or window size advertised by receiver. Therefore, there is no queuing in the system. Purpose of this simulation scenario is to provide completely controllable round-trip time and loss probability.

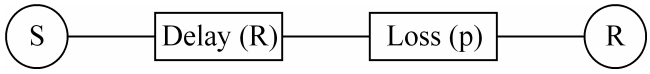

Fig. 29. Simulation scenario used for identification of parameters *a* and *b*.

Results obtained by varying round-trip time from 10 ms to 200 ms and loss probability from 0.1 % to 5 % are shown in Fig. 30. These results have been averaged over 15 simulation runs. Based on Levenburg-Marquardt algorithm for minimum least-squares fitting, values of parameters a and b and their 95% confidence intervals are:

$$\begin{aligned} a &= 0.9007, \quad a \in [0.8874, 0.9140] \\ b &= 0.7260, \quad b \in [0.7145, 0.7375] \end{aligned} .$$

Table 3 indicates that discrepancy between simulation results and the TCP throughput predicted by empirical model (60) is within ±15%. We consider these results good enough for the purpose of designing PLC controller described in Section 6. Model (60) is an ad-hoc solution driven by the lack of appropriate analytical TCP model that could be otherwise used..

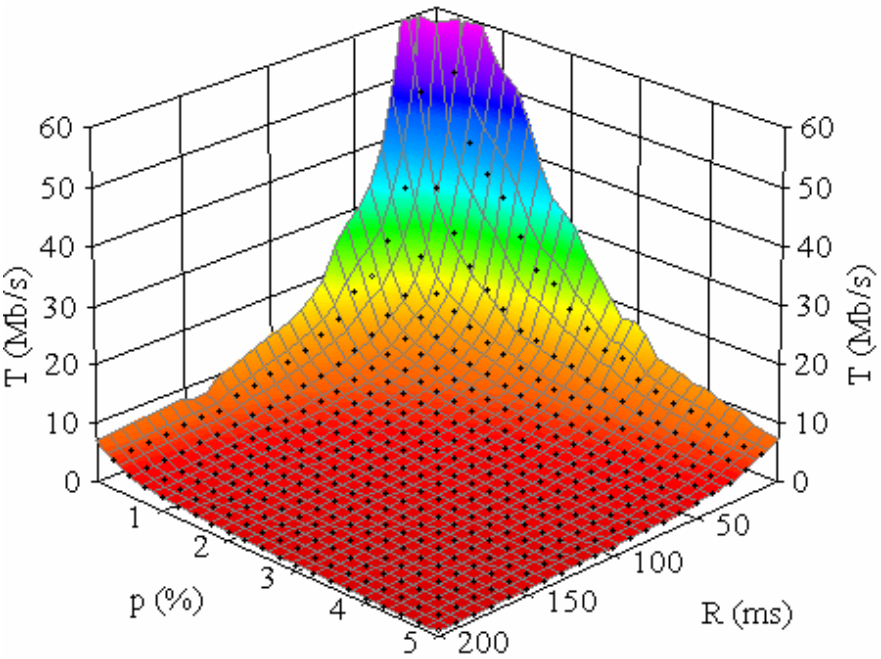

Fig. 30. TCP throughput as a function of round-trip time *R* and loss probability *p*.

Table 3. Goodness of fit: ns-2 simulation results vs. predicted throughput for selected round-trip times and loss probabilities.

R (ms)	p (%)	ns-2	Empirical model	
		T (Mb/s)	T (Mb/s)	Diff. (%)
20	0.50	32.369	29.682	-8.301
20	1.00	20.974	18.945	-9.671
20	2.00	12.109	10.849	-10.404
20	4.00	6.062	6.559	8.210
50	0.50	13.216	13.004	-1.600
50	1.00	8.863	7.862	-11.290
50	2.00	5.703	4.853	-14.895
50	4.00	3.029	2.874	-5.114
100	0.50	6.702	6.966	3.932
100	1.00	4.511	4.211	-6.636
100	2.00	2.911	2.546	-12.549
100	4.00	1.771	1.589	-10.254
150	0.50	4.472	4.835	8.097
150	1.00	3.000	2.923	-2.564
150	2.00	1.968	1.767	-10.201
150	4.00	1.414	1.268	-10.308

Appendix C

In this Appendix we prove that relation (53) introduced in Section 6.2 holds.

Substituting:

$$\frac{r_{tcp}}{r_{udp}}(1-\sigma_0) = z \tag{62}$$

in (51), gives:

$$1-\frac{r_{udp}}{r_{tcp}}z = (1+z)(1+\gamma\, z)^2\eta^2 . \tag{63}$$

Its polynomial form is:

$$f(z) := \gamma^2 z^3 + \gamma(\gamma+2)z^2 + \left(1+2\gamma+\frac{r_{udp}}{r_{tcp}}\frac{1}{\eta^2}\right)z + 1 - \frac{1}{\eta^2} = 0 . \tag{64}$$

It can be shown by using Descartes' rule of signs or by inspecting its determinant that $f(z)$ has one positive real root and a pair of complex conjugate. In order to prove (53), we show that the positive real solution of $f(z)$ belongs to the interval [0, 2/3).

First, we show that $f(0) \le 0$. It follows from (64) that:

$$f(0) = 1 - \frac{1}{\eta^2} . \tag{65}$$

Since η is defined by (41) and $r_{tcp}+r_{udp}=C$, it follows that $1/\delta \le \eta \le 1$. Hence, $f(0) \le 0$.

Second, we prove that $f(2/3) > 0$. From (64):

$$f\left(\frac{2}{3}\right)=\frac{5}{3}\left(1+\frac{2}{3}\gamma\right)^2-\frac{1}{\eta^2}\left(1-\frac{2}{3}\frac{r_{udp}}{r_{tcp}}\right). \tag{66}$$

Based on (52), the first term on the right hand side of (66) satisfies:

$$\frac{5}{3}\left(1+\frac{2}{3}\gamma\right)^2 \geq \frac{5}{3}. \tag{67}$$

After substituting $r_{udp} / r_{tcp} = y$ and using (41), the second term on the right hand side of (66) becomes:

$$\frac{1}{\eta^2}\left(1-\frac{2}{3}\frac{r_{udp}}{r_{tcp}}\right)=\frac{(1+y)^2}{(1+y/\delta)^2}\left(1-\frac{2}{3}y\right). \tag{68}$$

Since $0 \leq y < \infty$ and $1 \leq \delta < \infty$:

$$\frac{(1+y)^2}{(1+y/\delta)^2}\left(1-\frac{2}{3}y\right)<(1+y)^2\left(1-\frac{2}{3}y\right)<\frac{5}{3}. \tag{69}$$

From (66), (67), and (69), we conclude that $f(2/3) > 0$.

Since $f(0) \leq 0$ and $f(2/3) > 0$, real positive root of *f*(*z*) must be in the interval [0, 2/3). Hence, (53) holds.□

Appendix D

Structure of ns-2 source-code directories and location of the code that implements the proposed service differentiation architecture are shown in Fig. 31. The code includes four files: *ns-default.tcl* that defines default parameters (all inherited from RED, except for the delay differentiation parameter δ); *probe.h* and *probe.cc* that implement the BPR scheduler and the PLC controller; and *definitions.h* that contains certain auxiliary functions used in *probe.cc*.

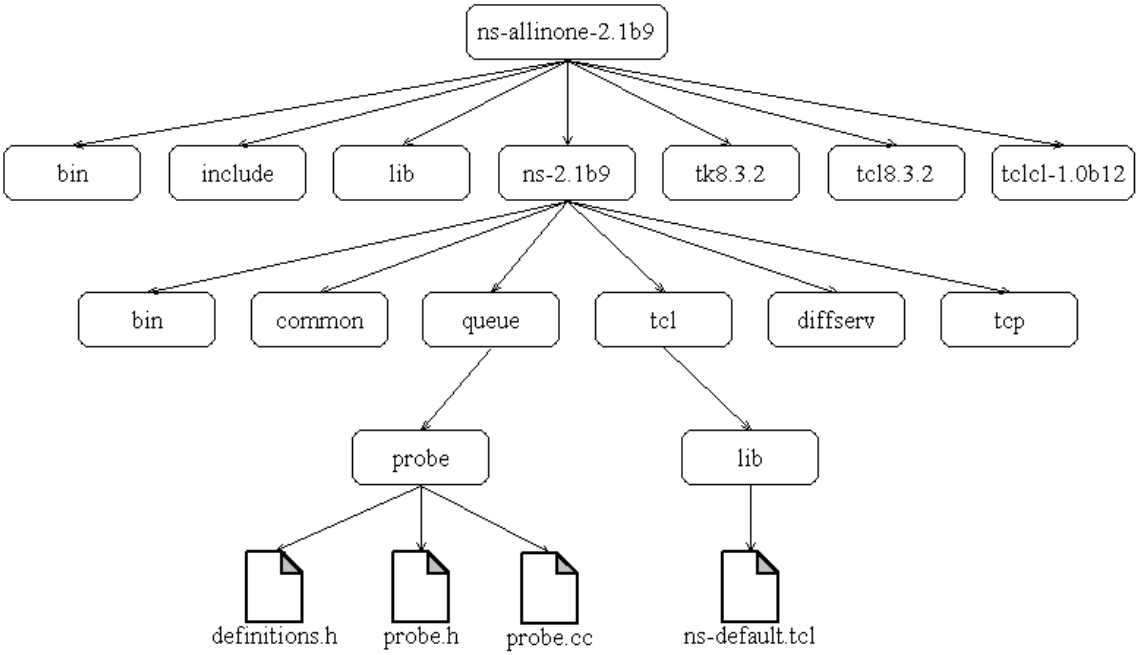

Fig. 31. Structure of ns2 source-code directories and position of the code that implements the proposed service differentiation architecture.

```
*****************************ns-default.tcl*****************************

+Queue/PROBE set bytes_ true
+Queue/PROBE set queue_in_bytes_ true
+Queue/PROBE set thresh_ 0
+Queue/PROBE set maxthresh_ 0
+Queue/PROBE set mean_pktsize_ 500
+Queue/PROBE set idle_pktsize_ 100
+Queue/PROBE set q_weight_ -1
```

```
+Queue/PROBE set wait_ true
+Queue/PROBE set linterm_ 10
+Queue/PROBE set ave_ 0.0
+Queue/PROBE set prob1_ 0.0
+Queue/PROBE set cur_max_p_ 0
+Queue/PROBE set delta_ 1

*****************************definitions.h*****************************

#ifndef ns_definitions_h
#define ns_definitions_h

#define TS        0
#define DS        1

#define HDR_FLAGS(p)    (hdr_flags::access(p))
#define isTS(p)         (HDR_FLAGS(p)->pri_ == TS)
#define isDS(p)         (HDR_FLAGS(p)->pri_ == DS)
#define setTS(p)        (HDR_FLAGS(p)->pri_ = TS)
#define setDS(p)        (HDR_FLAGS(p)->pri_ = DS)
#define pktSize(p)      (HDR_CMN(p)->size())
#define pktClass(p)     (HDR_FLAGS(p)->pri_)
#define TIME            Scheduler::instance().clock()
#define setTimestamp(p) (HDR_CMN(p)->timestamp_ = TIME)
#define getTimestamp(p) (HDR_CMN(p)->timestamp_)

#endif

*******************************probe.h*******************************

#ifndef ns_probe_h
#define ns_probe_h

#include "queue.h"
#include "trace.h"

class LinkDelay;

/* Early drop parameters, supplied by user */

struct edp {

      /* User supplied */
      int mean_pktsize; /* avg packet size, linked into Tcl */
      int idle_pktsize; /* avg packet size used during idle times */
      int bytes;        /* true if queue in bytes, false if packets */
      int wait;         /* true for waiting between dropped packets */
      double th_min;          /* min threshold of average queue size */
      double th_min_pkts;     /* always maintained in packets */
      double th_max;          /* max threshold of average queue size */
      double th_max_pkts;     /* always maintained in packets */
      double max_p_inv;       /* 1/max_p, for max_p = maximum prob.  */
      double q_w;       /* queue weight given to cur. q size sample */

      /* Computed as a function of user supplied parameters. */
      double ptc;       /* packet time constant in packets/second */
      double delay;     /* link delay */
```

```
};

/* Early drop variables, maintained by PROBE */

struct edv {
      TracedDouble v_ave;     /* average queue size */
      TracedDouble v_prob1;   /* prob. of pkt drop before "count". */
      TracedDouble cur_max_p; /* current max_p */
      double v_slope;         /* used in computing avg queue size */
      double v_prob;          /* prob. of packet drop */
      double v_a;             /* v_prob = v_a * v_ave + v_b */
      double v_b;
      int count;              /* # of packets since last drop */
      int count_bytes;        /* # of bytes since last drop */
      int old;                /* 0 when avg queue exceeds thresh */

      edv() : v_ave(0.0), v_prob1(0.0), v_slope(0.0), v_prob(0.0),
            v_a(0.0), v_b(0.0), count(0), count_bytes(0), old(0),
            cur_max_p(1.0) { }
};

class PROBEQueue : public Queue {
public:
      PROBEQueue();

protected:
      void initialize_params();
      void reset();
      void enque(Packet* pkt);
      void enque(int qos_class, Packet* pkt);
      Packet* deque();
      Packet* deque(int qos_class);
      Packet* getDS();
      void setLink(LinkDelay* lk_) {link_=lk_;}
      double estimator(int nq, int m, double ave, double q_w);
      int drop_early(Packet* pkt);
      double modify_p(double p, int count, int count_bytes, int bytes,
            int mean_pktsize, int wait, int size);
      double calculate_p_new(double v_ave, double th_max,
            double v_a, double v_b, double max_p);
      int length();
      int length(int qos_class);
      int byteLength();
      int byteLength(int qos_class);
      int command(int argc, const char*const* argv);

      LinkDelay* link_;             /* outgoing link */
      PacketQueue tsQueue;          /* underlying TS queue */
      PacketQueue dsQueue;          /* underlying DS queue */
      int qib_;                     /* queue measured in bytes? */
      double delta_, plc;
      double credits;
      double r_ts, r_ds;
      double v_ts, v_ds;
      double told;
      edp edp_;               /* early-drop parameters */
      edv edv_;               /* early-drop variables */
```

```
	int idle_;              /* queue is idle? */
	double idletime_;       /* time since the queue is idle */
	int first_reset_;       /* first time reset() is called */
};

#endif

*****************************probe.cc*********************************

#include <math.h>
#include <sys/types.h>
#include "config.h"
#include "template.h"
#include "random.h"
#include "flags.h"
#include "delay.h"
#include "probe.h"
#include "definitions.h"

static class PROBEClass : public TclClass {
public:
	PROBEClass() : TclClass("Queue/PROBE") {}
	TclObject* create(int argc, const char*const* argv) {
		return (new PROBEQueue());
	}
} class_probe;

PROBEQueue::PROBEQueue() : link_(NULL), idle_(1), credits(0){

	bind_bool("bytes_", &edp_.bytes);
	bind_bool("queue_in_bytes_", &qib_);
	bind("thresh_", &edp_.th_min_pkts);
	bind("maxthresh_", &edp_.th_max_pkts);
	bind("mean_pktsize_", &edp_.mean_pktsize);
	bind("idle_pktsize_", &edp_.idle_pktsize);
	bind("q_weight_", &edp_.q_w);
	bind_bool("wait_", &edp_.wait);
	bind("linterm_", &edp_.max_p_inv);
	bind("ave_", &edv_.v_ave);
	bind("prob1_", &edv_.v_prob1);
	bind("cur_max_p_", &edv_.cur_max_p);
	bind("delta_", &delta_);
}

void PROBEQueue::initialize_params(){

/* This function is the same as in RED code. */

	if (edp_.q_w == 0.0) {
		edp_.q_w = 1.0 - exp(-1.0/edp_.ptc);
	} else if (edp_.q_w == -1.0) {
		double rtt = 3.0*(edp_.delay+1.0/edp_.ptc);
		if (rtt < 0.1) rtt = 0.1;
		edp_.q_w = 1.0 - exp(-1.0/(10*rtt*edp_.ptc));
	}
	if (edp_.th_min_pkts == 0) {edp_.th_min_pkts = 5.0;}
```

```
	if (edp_.th_max_pkts == 0) edp_.th_max_pkts = 3.0 *
	edp_.th_min_pkts;
}

void PROBEQueue::reset(){

/* This function is the same as in RED code. */

	if (link_) {
		edp_.ptc = link_->bandwidth()/(8. * edp_.mean_pktsize);
		initialize_params();
	}
	if (edp_.th_max_pkts == 0)
		edp_.th_max_pkts = 3.0 * edp_.th_min_pkts;
	/*
	 * If queue is measured in bytes, scale min/max thresh
	 * by the size of an average packet specified by user.
	 */
	if (qib_) {
		edp_.th_min = edp_.th_min_pkts * edp_.mean_pktsize;
		edp_.th_max = edp_.th_max_pkts * edp_.mean_pktsize;
	} else {
		edp_.th_min = edp_.th_min_pkts;
		edp_.th_max = edp_.th_max_pkts;
	}
	edv_.v_ave = 0.0; edv_.v_slope = 0.0;
	edv_.count = 0; edv_.count_bytes = 0; edv_.old = 0;
	edv_.v_a = 1/(edp_.th_max - edp_.th_min);
	edv_.cur_max_p = 1.0/edp_.max_p_inv;
	edv_.v_b = - edp_.th_min/(edp_.th_max - edp_.th_min);
	idle_ = 1;
	if (&Scheduler::instance() != NULL) {
		idletime_ = TIME;
	} else idletime_ = 0.0; /* scheduler not instantiated yet */
	Queue::reset();
}

/* Compute the average queue size: nq can be in bytes or packets. */

double PROBEQueue::estimator(int nq, int m, double ave, double q_w){

	double new_ave, old_ave;
	new_ave = ave;
	while (--m >= 1) new_ave *= 1.0 - q_w;
	old_ave = new_ave;
	new_ave *= 1.0 - q_w;
	new_ave += q_w * nq;
	return new_ave;
}

Packet* PROBEQueue::deque(){

	Packet* pkt = 0;
	if (dsQueue.head() && !tsQueue.head()) {
		pkt = deque(DS);
		v_ts = 0; v_ds = 0;
	}
```

```
	if (!dsQueue.head() && tsQueue.head()) {
		pkt = deque(TS);
		v_ts = 0; v_ds = 0;
	}
	if (dsQueue.head() && tsQueue.head()) {
		if (getTimestamp(dsQueue.head()) >= told) {
			v_ds = 0;
		} else v_ds += r_ds * (TIME - told);
		if (getTimestamp(tsQueue.head()) >= told) {
			v_ts = 0;
		} else v_ts += r_ts * (TIME - told);
		if ((8*pktSize(dsQueue.head())-v_ds) > (8*pktSize
		(tsQueue.head())-v_ts)) {
			pkt = deque(TS);
		} else pkt = deque(DS);
	}
	if (length(TS)!=0 && length(DS)!=0) {
		double bw = link_->bandwidth();
		r_ds = bw/(1 + byteLength(TS)/(byteLength(DS) * delta_));
		r_ts = bw - r_ds;
	}
	if (pkt) {
		told = TIME; idle_ = 0;
	} else {
		idle_ = 1;
		if (&Scheduler::instance() != NULL) {
			idletime_ = TIME;
		} else idletime_ = 0.0;
	}
	return pkt;
}

/* Calculate the drop probability. */

double PROBEQueue::calculate_p_new(double v_ave, double th_max, double
v_a, double v_b, double max_p) {

	double p;
	p = v_a * v_ave + v_b;
	p *= max_p;
	if (p > 1.0) p = 1.0;
	return p;
}

/* Make uniform instead of geometric inter-drop periods. */

double PROBEQueue::modify_p(double p, int count, int count_bytes, int
bytes, int mean_pktsize, int wait, int size) {

	double count1 = (double) count;
	if (bytes)
		count1 = (double) (count_bytes/mean_pktsize);
	if (wait) {
		if (count1 * p < 1.0) {
			p = 0.0;
		} else if (count1 * p < 2.0) {
			p /= (2 - count1 * p);
```

```
            } else p = 1.0;
      } else {
            if (count1 * p < 1.0) {
                  p /= (1.0 - count1 * p);
            } else p = 1.0;
      }
      if (bytes && p < 1.0) p = p * size / mean_pktsize;
      if (p > 1.0) p = 1.0;
      return p;
}

int PROBEQueue::drop_early(Packet* pkt) {

      hdr_cmn* ch = hdr_cmn::access(pkt);
      edv_.v_prob1 = calculate_p_new(edv_.v_ave, edp_.th_max, edv_.v_a,
      edv_.v_b, edv_.cur_max_p);
      edv_.v_prob = modify_p(edv_.v_prob1, edv_.count,edv_.count_bytes,
      edp_.bytes, edp_.mean_pktsize, edp_.wait, ch->size());

      /* Drop probability is computed, pick random number and act. */

      double u = Random::uniform();
      if (u <= edv_.v_prob) {
            /* DROP */
            edv_.count = 0; edv_.count_bytes = 0;
            hdr_flags* hf = hdr_flags::access(pkt);
            return (1);             /* drop */
      }
      return (0);                   /* no drop */
}

#define     DTYPE_NONE        0     /* no drop */
#define     DTYPE_FORCED      1     /* a "forced" drop */
#define     DTYPE_UNFORCED    2     /* an "unforced" (random) drop */

void PROBEQueue::enque(Packet* pkt) {

/* if we were idle, we pretend that m packets arrived during
 * the idle period.  m is set to be the ptc times the amount
 * of time we've been idle for. */

      double now = TIME;
      int m = 0;
      if (idle_) {
            /* To account for the period when the queue was empty. */
            idle_ = 0;
            /* Use idle_pktsize instead of mean_pktsize, for
             * a faster response to idle times. */
            m = int(edp_.ptc * (now - idletime_));
      }

/* Run the estimator with either 1 new packet arrival, or with
 * the scaled version above (scaled by m due to idle time). */

      edv_.v_ave = estimator(qib_ ? byteLength() : length(), m+1,
      edv_.v_ave, edp_.q_w);
```

```
/* count and count_bytes keeps a tally of arriving traffic
 * that has not been dropped (i.e., how long, in terms of traffic,
 * it has been since the last early drop). */

      hdr_cmn* ch = hdr_cmn::access(pkt);
      ++edv_.count;
      edv_.count_bytes += ch->size();

/* DROP LOGIC:
 *    q = current q size, ~q = averaged q size
 *    1) if ~q > maxthresh, this is a FORCED drop
 *    2) if minthresh < ~q < maxthresh, this may be an UNFORCED drop
 *    3) if (q+1) > hard q limit, this is a FORCED drop */

      register double qavg = edv_.v_ave;
      int droptype = DTYPE_NONE;
      int qlen = qib_ ? byteLength() : length();
      int qlim = qib_ ? (qlim_ * edp_.mean_pktsize) : qlim_;
      if (qavg >= edp_.th_min && qlen > 1) {
            if (qavg >= edp_.th_max) {
                  droptype = DTYPE_FORCED;
            } else if (edv_.old == 0) {

            /* The average queue size has just crossed the
             * threshold from below to above "minthresh", or
             * from above "minthresh" with an empty queue to
             * above "minthresh" with a nonempty queue. */

                  edv_.count = 1;
                  edv_.count_bytes = ch->size();
                  edv_.old = 1;
            } else if (drop_early(pkt)) droptype = DTYPE_UNFORCED;
      } else {
            /* No packets are being dropped.  */
            edv_.v_prob = 0.0;
            edv_.old = 0;
      }
      if (qlen >= qlim) droptype = DTYPE_FORCED;
      if (droptype == DTYPE_UNFORCED || droptype == DTYPE_FORCED) {
            if (isTS(pkt)) {
                  if (droptype == DTYPE_FORCED) {
                        edv_.v_prob = 1;
                  } else {
                        edv_.v_prob = calculate_p_new(edv_.v_ave,
                        edp_.th_max, edv_.v_a, edv_.v_b,
                        edv_.cur_max_p);
                  }
                  double bw = link_->bandwidth();
                  double ni = ((r_ts + r_ds/delta_) / bw);
                  double psi = - edv_.cur_max_p * edv_.v_b;
                  double eta = 2*edv_.v_prob / (edv_.v_prob + psi);
                  if (r_ds > 0) {
                        plc = (1+(eta+1)*r_ts/r_ds) /
                        (1/(ni*ni)+(eta+1)*r_ts/r_ds);
                  } else plc = 1;
                  credits += (1-plc);
                  if (credits >= 1) {
```

```
                        int count = 0;
                        while (count < pktSize(pkt)) {
                              Packet* ds_pkt = getDS();
                              if (ds_pkt) {
                                    count += pktSize(ds_pkt);
                                    drop(ds_pkt);
                              } else break;
                        }
                        credits -= count / pktSize(pkt);
                        if (count >= pktSize(pkt)) {
                              enque(TS, pkt);
                        } else drop(pkt);
                  } else drop(pkt);
            } else drop(pkt);
      } else {
            if isDS(pkt) enque(DS, pkt); else enque(TS, pkt);
      }
      return;
}

int PROBEQueue::command(int argc, const char*const* argv) {

      Tcl& tcl = Tcl::instance();
      if (argc == 2) {
            if (strcmp(argv[1], "reset") == 0) {
                  reset();
                  return (TCL_OK);
            }
      }
      else if (argc == 3) {
            if (strcmp(argv[1], "link") == 0) {
                  LinkDelay* del = (LinkDelay*)
                  TclObject::lookup(argv[2]);
                  if (del == 0) {
                        tcl.resultf("PROBE: no LinkDelay object %s",
                        argv[2]);
                        return(TCL_ERROR);
                  }
                  link_ = del;
                  edp_.ptc = link_->bandwidth()/(8.*edp_.mean_pktsize);
                  edp_.delay = link_->delay();
                  if (edp_.q_w <= 0.0 || edp_.th_min_pkts == 0 ||
                  edp_.th_max_pkts == 0) initialize_params();
                  return (TCL_OK);
            }
      }
      return (Queue::command(argc, argv));
}

int PROBEQueue::byteLength(int qos_class) {

      if (qos_class == TS) return tsQueue.byteLength();
      if (qos_class == DS) return dsQueue.byteLength();
}

int PROBEQueue::byteLength() {return byteLength(TS)+byteLength(DS);}
```

```
int PROBEQueue::length(int qos_class) {

      if (qos_class == TS) return tsQueue.length();
      if (qos_class == DS) return dsQueue.length();
}

int PROBEQueue::length() {return length(TS)+length(DS);}

Packet* PROBEQueue::deque(int qos_class) {
      Packet* pkt=0;
      if (qos_class == DS) {
            pkt = dsQueue.deque();
            v_ds = v_ds - 8*pktSize(pkt);
      } else {
            pkt = tsQueue.deque();
            v_ts = v_ts - 8*pktSize(pkt);
      }
      return pkt;
}

Packet* PROBEQueue::getDS() {

      Packet* pkt = dsQueue.deque();
      return pkt;
}

void PROBEQueue::enque(int qos_class, Packet* pkt) {

      setTimestamp(pkt);
      if (qos_class == DS) {
            dsQueue.enque(pkt);
      } else {
            tsQueue.enque(pkt);
      }
}

**********************************************************************
```

References

[1] V. Firoiu, J.-Y. Le Boudec, D. Towsley, and Z.-L. Zhang, "Theories and models for Internet quality of service," *Proc. IEEE*, vol. 90, no. 9, pp. 1565–1591, September 2002.

[2] P. Gevros, J. Crowcroft, P. Kirstein, and S. Bhatti, "Congestion control mechanisms and the best-effort service model," *IEEE Network*, vol. 15, no. 3, pp. 16–26, May 2001.

[3] P. Hurley, M. Kara, J. Y. Le Boudec, and P. Thiran, "ABE: providing a low-delay service within best-effort," *IEEE Network*, vol. 15, no. 3, pp. 60–69, May 2001.

[4] V. Firoiu and X. Zhang, "Best-effort Differentiated Services: trade-off service differentiation for elastic applications," in *Proc. IEEE ICT 2001*, Bucharest, Romania, June 2001.

[5] B. Gaidioz and P. Primet, "EDS: a new scalable service differentiation architecture for Internet," in *Proc. IEEE ISCC*, Taormina, Italy, July 2002, pp. 777–782.

[6] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," RFC 2212, Internet Engineering Task Force, September 1997.

[7] J. Wroclawski, "Specification of the controlled-load network element service," RFC 2211, Internet Engineering Task Force, September 1997.

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, Internet Engineering Task Force, December 1998.

[9] V. Jacobson, K. Nichols, and K. Poduri, "An expedited forwarding PHB," RFC 2598, Internet Engineering Task Force, June 1999.

[10] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding PHB group," RFC 2597, Internet Engineering Task Force, June 1999.

[11] B. Teitelbaum, "Future priorities for Internet2 QoS," Internet2 QoS WG, October 2001:http://www.internet2.edu/qos/wg/papers/qosFuture01.pdf.

[12] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional differentiated services: delay differentiation and packet scheduling," in *Proc. ACM SIGCOMM 1999*, Cambridge MA, September 1999, pp. 109–120.

[13] S. Bodamer, "A new scheduling mechanism to provide relative differentiation for real-time IP traffic," in *Proc. IEEE GLOBECOM 2000*, San Francisco, CA, December 2000, pp. 646–650.

[14] 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 *Proc. IEEE International Conference on Computer Communications and Networks (ICCCN 2000)*, October 2000, pp. 418–423.

[15] J.-S. Li and H.-C. Lai, "Providing proportional differentiated services using PLQ," in *Proc. IEEE GLOBECOM 2001*, San Antonio, TX, November 2001, pp. 2280–2284.

[16] Y-C. Lai and W-H. Li, "A novel scheduler for proportional delay differentiation by considering packet transmission time," *IEEE Communications Letters*, vol. 7, no. 4, pp. 189–191, April 2003.

[17] Y. Chen, C. Qiao, M. Hamdi, and D. H. K Tsang, "Proportional differentiation: a scalable QoS approach," *IEEE Communications Magazine*, vol. 41, no. 6, pp.52–58, June 2003.

[18] H. Saito, C. Lukovzski, and I. Moldovan, "Local optimal proportional differentiated services scheduler for relative differentiated services," in *Proc. IEEE International Conference on Computer Communications and Networks (ICCCN 2000)*, October 2000, pp. 544–550

[19] H-T. Ngin and C-K. Tham, "Achieving proportional delay differentiation efficiently," in *Proc. IEEE International Conference on Networks (ICON 2002)*, Singapore, August 2002, pp. 169–174.

[20] T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "Delay differentiation and adaptation in core stateless networks," in *Proc. IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000, pp. 421–430.

[21] S. Sankaran and A. E. Kamal, "A combined delay and throughput proportional scheduling scheme for differentiated services," in *Proc. IEEE Globecom*, Taipei, Taiwan, November 2002, pp. 1910–1914.

[22] Y. Moret and S. Fdida, "A proportional queue control mechanism to provide differentiated services," in *Proc. International Symposium on Computer and Information Systems (ISCIS 1998)*, Belek, Turkey, October 1998, pp. 17–24.

[23] M. K. H. Leung, J. C. S. Lui, and D. K. Y. Yau, "Adaptive proportional delay differentiated services: characterization and performance evaluation," *IEEE/ACM Trans. Networking*, vol. 9, no. 6, December 2001, pp. 801–817.

[24] C. Dovrolis and P. Ramanathan, "Proportional differentiated services, part II: Loss rate differentiation and packet dropping," in *Proc. International Workshop on Quality of Service (IWQoS 2000)*, Pittsburgh, PA, June 2000, pp. 52–61.

[25] W. Wu, Y. Ren, and X. Shan, "Forwarding the balance between absolute and relative: a new differentiated services model for adaptive traffic**,**" in *Proc. IEEE Workshop on High Performance Switching and Routing (HPSR 2001)*, Dallas, TX, May 2001, pp. 250–254.

[26] A. Striegel and G. Manimaran, "Differentiated services: packet scheduling with delay and loss differentiation," in *Computer Communications*, vol. 25, no. 1, pp.21–31, January 2002.

[27] U. Bodin, A. Jonsson, and O. Schelén, "On creating proportional loss-rate differentiation: predictability and performance," in *Proc. International Workshop on Quality of Service (IWQoS 2001)*, Karlsruhe, Germany, June 2001, pp. 372–388.

[28] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM 2000*, Stockholm, Sweden, August 2000, pp. 43–56.

[29] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[30] V. Vukadinović, G. Petrović, and Lj. Trajković, "TCP-friendly packet loss controller for proportional delay differentiation," submitted for publication.

[31] The Network Simulator - ns-2: http://www-mash.cs.berkeley.edu/ns.

[32] H. J. Chao and X. Guo, *Quality of service control in high-speed networks*. New York: John Wiley and Sons, 2002.

[33] S. Jha and M. Hassan, *Engineering Internet QoS*. Boston: Artech House, 2002.

[34] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Trans. Networking*, vol. 8, no. 2, pp. 133–145, April 2000.

[35] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, July 1997.

[36] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," in *Proc. ACM SIGCOMM'89*, Austin, TX, September 1989, pp. 1–12.

[37] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services field (DS field) in the IPv4 and IPv6 headers," IETF RFC 2474, December 1998.

[38] V. Vukadinovic. G. Petrovic, and Lj. Trajkovic, "BPR+: a packet scheduling algorithm for proportional delay differentiation," in *Proc. YUINFO 2004*, Kopaonik, Serbia and Montenegro, March 2004.

[39] N. Christin and J. Liebeherr, "A QoS architecture for quantitative service differentiation," *IEEE Communications Magazine*, vol. 41, no. 6, pp.38–45 June 2003.

[40] J. Liebeherr and N. Christin, "Rate allocation and buffer management for differentiated services," *Computer Networks*, vol. 40, no. 1, pp. 89–110, September 2002.

[41] S. Floyd, "RED: discussions of setting parameters," November 1997: http://www.icir.org/floyd/REDparameters.txt.

[42] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional differentiated services: delay differentiation and packet scheduling," *IEEE/ACM Trans. Networking*, vol. 10, no. 1, pp. 12–26, February 2002.

[43] D. Lin and R. Morris, "Dynamics of random early detection," in *Proc. ACM SIGCOMM '97*, Cannes, France, October 1997, pp. 127–137.

[44] W. Feng, "Stochastic fair BLUE: a queue management algorithm for enforcing fairness," in *Proc. IEEE INFOCOM'01*, Anchorage, AL, April 2001, pp. 1520–1529.

[45] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe: a stateless active queue management scheme for approximating fair bandwidth allocation," in *Proc. IEEE INFOCOM'00*, Tel-Aviv, Israel, April 2000, pp. 942–951.

[46] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," in *ACM Computer Communication Review*, vol. 32, no. 1, January 2002, pp. 72.

[47] J. Bruno, B. Ozden, H. Saran, and A. Silberschatz, "Early fair drop: a new buffer management policy," in *Proc. ACM/SPIE Multimedia Computing and Networking 1999*, San Jose, CA, January 1999, pp. 148–161.

[48] V. Vukadinović and Lj. Trajković, "RED with dynamic thresholds for improved fairness," in *Proc. ACM Symposium on Applied Computing (SAC 2004)*, Nicosia, Cyprus, March 2004, pp. 371–372.

[49] G. Chatranon, M. Labrador, and S. Banerjee, “BLACK: detection and preferential dropping of high bandwidth unresponsive flows,” in *Proc. IEEE ICC 2003*, Anchorage, AK, May 2003, pp. 664–668.

[50] G. Hasegawa and M. Murata, “Dynamic threshold control of RED for fairness among thousands TCP connections," in *Proc. 4th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT 2001)*, Kathmandu, Nepal, November 2001, pp. 213–217.