Rate Allocation and Buffer Management for Differentiated Services *

Jörg Liebeherr *Nicolas Christin*
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

To appear in *Computer Networks, Special Issue on the New Internet Architecture*, May 2002.

Abstract

A novel algorithm for buffer management and rate allocation is presented for providing loss and delay differentiation for traffic classes at a network router. The algorithm, called JoBS, provides delay and loss differentiation independently at each node, without assuming admission control or policing. Contrary to most existing algorithms, scheduling and buffer management decisions are performed in a single step. Both relative and (whenever possible) absolute QoS requirements of classes are supported. Simulation examples, including results for a heuristic approximation, are presented to illustrate the effectiveness of the approach, to compare the new algorithm to existing methods for loss and delay differentiation. [1]

Key Words: Buffer Management, Scheduling, Service Curves, Quality-of-Service, Service Differentiation.

*This work is supported in part by the National Science Foundation through grants NCR-9624106 (CAREER), ANI-9730103, and ANI-0085955.

[1] An earlier version of this paper was presented at the Ninth International Workshop on Quality-of-Service (IWQoS 2001) [28].

1 Introduction

Quality-of-Service (QoS) guarantees in packet networks are often classified according to two criteria. The first criterion is whether guarantees are expressed for individual end-to-end traffic flows (*per-flow QoS*) or for groups of flows with the same QoS requirements (*per-class QoS*). The second criterion is whether guarantees are expressed with reference to guarantees given to other flows/flow classes (*relative QoS*), or whether guarantees are expressed as absolute bounds (*absolute QoS*).

Efforts to provision for QoS in the Internet in the early and mid-1990s, which resulted in the *Integrated Services* (IntServ) service model [7], focused on per-flow absolute QoS guarantees. However, due to scalability issues and a lagging demand for per-flow absolute QoS, the interest in Internet QoS eventually shifted to relative per-class guarantees. Since late 1997, the *Differentiated Services* (DiffServ) [5] working group has discussed several proposals for per-class relative QoS guarantees, e.g., [11, 33].

With the exception of the Expedited Forwarding service [22] and its current revisions [3, 16], proposals for relative per-class QoS discussed within the DiffServ context define the service differentiation qualitatively, in the sense that some classes receive lower delays and a lower loss rate than others, but without quantifying the differentiation. Recently, research studies have tried to strengthen the guarantees of relative per-class QoS, and have proposed new buffer management and scheduling algorithms which can support stronger notions of relative QoS [6, 13, 14, 31, 32, 39]. Probably the best known such effort is the *proportional service differentiation* model [13, 14], which attempts to enforce that the ratios of delays or loss rates of successive priority classes be roughly constant. For two priority classes such a service could specify that the delays of packets from the higher-priority class be half of the delays from the lower-priority class, but without specifying an upper bound on the delays.

In this paper, we express the provisioning of per-class QoS within a formalism inspired by the network calculus [12]. We present a rate allocation and dropping algorithm for a single output link, called *Joint Buffer Management and Scheduling (JoBS)*, which is capable of supporting a wide range of relative, as well as absolute, per-class guarantees for loss and delay, without assuming admission control or traffic policing.

JoBS has two unique capabilities: (1) Rate allocation for link scheduling and buffer management are approached together in a single step, and (2) JoBS supports both relative and absolute QoS requirements of classes.

The JoBS algorithm operates as follows. Upon each arrival, a prediction is made on the delays of the currently backlogged traffic. Then, the service rates allocation to classes are adjusted to meet delay requirements. If necessary, traffic from certain classes is selectively dropped. The algorithm provides delay and loss differentiation independently at each node. End-to-end delays and end-to-end loss rates are thus dependent on the per-node guarantees of traffic and on the number of nodes traversed.

This paper is organized as follows. In Section 2 we give an overview of the current work on relative per-class QoS guarantees. In Sections 3 and 4, we specify our algorithm for buffer management and rate allocation. In Section 5 we propose a heuristic approximation of the algorithm. In Section 6 we evaluate the effectiveness of our algorithm via simulation. In Section 7 we present brief conclusions.

2 Related Work

The significant amount of literature on service differentiation prevents us from presentive a comprehensive survey of the related work on scheduling and buffer management algorithms on per-class relative guarantees. The discussion in this section thus focuses on techniques devised to improve Best Effort or DiffServ services, which are the closest to our proposed scheme.

With two exceptions [21, 39], the related work on relative per-class service differentiation treats delay and loss differentiation as orthogonal issues. The common approach is to use scheduling algorithms for

achieving delay differentiation, and to use buffer management algorithms for performing loss differentiation.

2.1 Scheduling

The majority of work on per-class relative service differentiation suggests to use well-known fixed-priority, e.g., [33], or rate-based scheduling algorithms, e.g., [19]. Only a few scheduling algorithms have been specifically designed for relative delay differentiation.

The Proportional Queue Control Mechanism (PQCM, [31]) and Backlog-Proportional Rate (BPR,[15]) are variations of the GPS algorithm [35]. Both schedulers dynamically adjust service rate allocations of classes to meet relative QoS requirements. The service rate allocation is based upon the backlog of classes at the scheduler. The main difference between PQCM and BPR is the specific method used to calculate the service rates. Both schemes bear similarity to the scheduling component of JoBS, in the sense that they dynamically adjust service rate allocations to meet relative QoS requirements.

Different from the rate-based schedulers discussed above, Waiting-Time Priority (WTP, [15]) implements a well-known scheduling algorithm with dynamic time-dependent priorities ([23], Ch. 3.7). A class-i packet, which arrives at time τ, is assigned a time-dependent priority as follows. If the packet is backlogged at time $t > \tau$, then WTP assigns this packet a priority of $(t - \tau) \cdot s_i$, where s_i is a class-dependent priority coefficient [23]. WTP packets are transmitted in the order of their priorities. In [15], the coefficients s_i are chosen so that $s_1 = k \cdot s_2 = k^2 \cdot s_3 = \ldots = k^Q \cdot s_Q$, resulting in a delay differentiation under high loads, where Class- $(i + 1)$ Delay $\approx k \cdot$Class- i Delay. The Mean-Delay Proportional scheduler (MDP, [32]) has a dynamic priority mechanism similar to WTP, but uses estimates of the average delay of a class to determine the priority of that class. Thus, the priority of a class-i packet is set to $s_i \cdot \overline{d_i}(t)$, where $\overline{d_i}(t)$ is the estimated average delay for class- i, averaged over the entire up-time of the link. The coefficients s_i, are as in WTP, i.e., $s_1 = k \cdot s_2 = k^2 \cdot s_3 = \ldots = k^Q \cdot s_Q$. Finally, the Hybrid Proportional Delay scheduler (HPD, [13]) uses a combination of waiting-time and average experienced delay to determine the priority of a given packet. Therefore, the priority of a given class is set to $s_i(g(t - \tau) + (1 - g)\overline{d_i}(t))$ with $0 < g < 1$. ([13] recommends to set $g = 0.875$.)

In contrast to the schedulers presented in this section, the rate allocation algorithms presented in this paper not only consider the current state and past history of the link, but, in addition, use this information to make predictions on future delays in order to improve the performance of the scheduling decisions.

We conclude by briefly discussing the relation of our work to the recently proposed Scalable-Core (SCORE) approach [38]. SCORE provides end-to-end delay guarantees to flows without requiring per-flow state information at network routers. The basic idea for meeting end-to-end delay requirements is to keep track of the delays experienced by packets along the path from the source to the destination, by storing the experienced delays in the packet headers. The stored information is used for adjusting the priority of packets so that end-to-end requirements are met. The CoreLite architecture [32] is an extension of this work which couples per-hop proportional delay differentiation with end-to-end delay guarantees. Different from the above schedulers, the algorithms presented in this paper do not store any state information in packets.

2.2 Buffer Management

The key mechanisms of a buffer management algorithm are the *backlog controller*, which specifies the time instances when traffic should be dropped, and the *dropper*, which specifies the traffic to be dropped. We refer to a recent survey article [25] for an extensive discussion of buffer management algorithms.

Backlog Controllers Initial proposals for buffer management (or active queue management) in IP networks [17, 18] were motivated by the need to improve TCP performance, without considering service dif-

ferentiation. More recent research efforts [11, 30, 34, 36] enhance these initial proposals in order to provide service differentiation.

Among backlog controllers for IP networks, Random Early Detection (RED, [18]) is probably the best known algorithm. RED was motivated by the goal to improve TCP throughput in highly loaded networks. RED operates by probabilistically dropping traffic arrivals, when the backlog at a node grows large. RED has two threshold parameters for the backlog at a node, denoted as TH_{small} and TH_{large}. RED estimates the average queue size, $\overline{Q}_{est}$ and compares the estimate to the two thresholds. If $\overline{Q}_{est} < TH_{small}$, RED does not drop any arrival. If $\overline{Q}_{est} > TH_{large}$, RED drops all incoming traffic. If $TH_{small} \leq \overline{Q}_{est} \leq TH_{large}$, RED will drop an arrival with probability $P(\overline{Q}_{est})$, where $0 \leq P(\overline{Q}_{est}) \leq 1$ is a function which increases in $\overline{Q}_{est}$.

Several algorithms that attempt to improve or extend RED have been proposed, e.g., [4, 11, 17, 20, 30, 34, 36]. For example, BLUE [17] uses different metrics to characterize the probability of dropping an arrival. Instead of the backlog, this algorithm uses the current loss ratio and link utilization as input parameters.

RIO [11] and multiclass RED [36] are extensions to RED which aim at class-based service differentiation. Both schemes have different dropping thresholds for different classes, in order to ensure loss differentiation. In an IntServ context, the idea of using different threshold values is pursued for Flow-RED (FRED, [30]), which uses per-flow thresholds. Flows are discriminated by their source-destination address pairs.

Random Early Marking (REM, [4]) is close in spirit to the dropping algorithm in JoBS, since it treats the problem of marking (or dropping) arrivals as an optimization problem. The objective is to maximize a utility function subject to the constraint that the output link has a finite capacity. In [4], this problem is reduced to the REM algorithm, which marks packets with a probability exponentially dependent on the cost of a link. The cost is directly proportional to the buffer occupancy. [20] proposes to replace RED with a proportional-integral controller to achieve faster convergence to the desired queue length and to increase robustness of the system.

Droppers The simplest and most widely used dropping scheme is Drop-Tail, which discards arrivals to a full buffer. For a long time, discarding arrivals was thought to be the only viable solution for high-speed routers. Recent implementation studies [40] demonstrated that other, more complex, dropping schemes, which discard packets that are already present in the buffer (push-out), are viable design choices even at high data rates.

The simplest push-out technique is called Drop-from-Front [26]. Here, the oldest packet in the transmission queue is discarded. In comparison to Drop-Tail, Drop-from-Front lowers the queueing delays of all packets waiting in the system. Note that with Drop-Tail, dropping of a packet has no influence on the delay of currently queued packets.

Other push-out techniques include Lower Priority First (LPF, [24, 29]), Complete Buffer Partitioning (CBP, [29]), and Partial Buffer Sharing (PBS, [24]). LPF always drops packets from the lowest backlogged priority queue. As shown in [14], LPF does not provide any mechanism for proportional loss differentiation. CBP assigns a dedicated amount of buffer space to each class, and drops traffic when this dedicated buffer is full. PBS uses a partitioning scheme similar to CBP, but the decision to drop is made after having looked at the aggregated backlog of all classes. The static partitioning of buffers in LPF, CBP, and PBS is not suitable for relative per-class service differentiation, since no *a priori* knowledge of the incoming traffic is available [14].

The Proportional Loss Rate (PLR) dropper [14] is specifically designed to support proportional differentiated services. PLR enforces that the ratio of the loss rates of two successive classes remains roughly constant at a given value. There are two variants of PLR. PLR(M) uses only the last M arrivals for estimating the loss rate of a class, whereas PLR(∞) has no such memory constraints. Average Drop Distance (ADD, [6]) is a variant of PLR(M) aiming at providing loss differentiation regardless of the timescale chosen for

computing the loss rates.

There are two recent exceptions to the idea that delay and loss differentiation are orthogonal issues which are handled by separate algorithms. The Duplicate Scheduler with Deadlines used in the Alternative Best-Effort service (ABE, [21]) provides service differentiation for two traffic classes. The first class has an absolute delay bound, while the second class experienes a lower loss rate. The delay guarantees for the first class are enforced by dropping all traffic that has exceeded a given delay bound. The recently proposed Class-Distance-Based-Priority-Delay-Loss scheduler (C-DBP-Delay-Loss, [39]) tries to provide proportional delay and proportional loss differentiation in a single algorithm. At any time, C-DBP-Loss-Delay keeps state information as a set of (m_i, k) pairs, where, for each class i, m_i represents the number of packets that have been successfully transmitted in the last k packets. For each packet, C-DBP-Loss-Delay computes the distance between the current state of the system, and a "failure state", defined as any state where m_i is less than a desired value $\hat{m}_i$. The lower the distance, the higher the priority assigned to the packet. The authors of [39] conjecture that setting $\hat{m}_i = k - s_i$, where s_i is selected as in WTP and MDP, can be used to provide proportional loss and delay differentiation.

3 An Approach to Joint Buffer Management and Rate Allocation

In this section, we introduce the key concepts of the JoBS algorithm. Before we provide a detailed description, we first give an informal overview of the operations.

3.1 Overview

We assume that each output link performs per-class buffering of arriving traffic and that traffic is transmitted from the buffers using a rate-based scheduling algorithm [41] with a dynamic, time-dependent service rate allocation for classes. Traffic from the same class is transmitted in a First-Come-First-Served order. There is no admission control and no policing of traffic. The set of performance requirements are specified to the algorithm as a set of per-class QoS constraints. As an example, for three classes, the QoS constraints could be of the form:

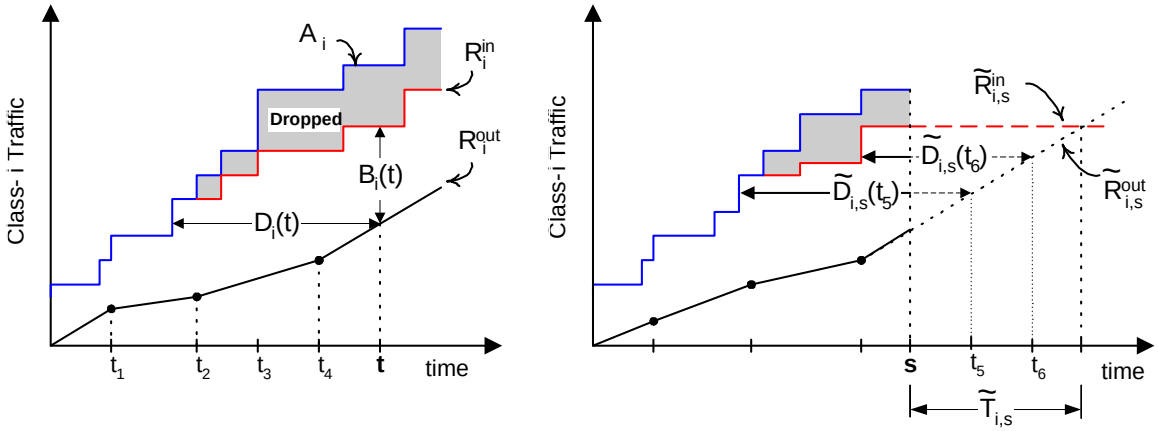
- Class-1 Delay $\approx 2 \cdot$ Class-2 Delay,
- Class-2 Loss Rate $\approx 10^{-1} \cdot$ Class-3 Loss Rate, or
- Class-3 Delay $\leq 5\ ms$.

Here, the first two constraints are relative constraints and the last one is an absolute constraint. [2]The set of constraints can be any mix of relative and absolute constraints. Since absolute constraints may render a system of constraints infeasible, some constraints may need to be relaxed. We assume that all QoS constraints are given a precedence order, which is used to determine which constraints are relaxed in case of an unfeasible system.

The service rate allocation operates as follows. For each arrival, a prediction is made on the delays of all backlogged traffic. Then, the service rate allocation to traffic classes is modified so that all QoS constraints will be met. If there exists no feasible rate allocation that meets all constraints, traffic is dropped, either from a new arrival or from the current backlog.

We will view the service rate allocation in terms of an optimization problem. The constraints are relative or absolute bounds on the loss and delay as given in the example above (*QoS constraints*) and constraints

[2]Throughout this paper, we only consider *deterministic* QoS guarantees. Statistical guarantees (see for instance [8]), which, for instance provide a delay bound D_3 to Class-3 packets with probability $1 - \varepsilon$ where $\varepsilon > 0$ are outside the scope of this paper.



(a) Delay and backlog.

(b) Projected input curve, projected output curve, and projected delays.

Figure 1: **Delay, Backlog and Projections.** In Figure 1(b), the projection is performed at time s for the time interval $[s, s + \tilde{T}_{i,s}]$.

on the link and buffer capacity (*system constraints*). The objective function of the optimization aims at minimizing the amount of traffic to be dropped, and, as a secondary objective, aims at maintaining the current service rate allocation. The first objective prevents traffic from being dropped unnecessarily, and the second objective tries to avoid frequent fluctuations of the service rate allocation. The solution of the optimization problem yields a service rate allocation of classes and determines how much traffic must be dropped.

To explore the principal properties of the optimization, we will, at first, assume that sufficient computing resources are available to solve the optimization problem for each arrival to the link. In a later section, we will approximate the optimization with a heuristic which incurs less computational overhead.

3.2 Formal Description

Next we describe the basic operations of the service rate allocation and the dropping algorithms at a link with capacity C and total buffer space B. We assume that all traffic is marked to belong to one of Q traffic classes. In general, we expect Q to be small, e.g., $Q = 4$. Classes are marked by an index. We use a convention, whereby a class with a smaller index requires a better level of QoS. We use $a_i(t)$ and $l_i(t)$ to denote the traffic arrivals and amount of dropped traffic from class i at time t. We use $r_i(t)$ to denote the service rate allocated to class i at time t. We assume that $r_i(t) > 0$ only if there is a backlog of class-i traffic in the buffer (and $r_i(t) = 0$ otherwise), and we assume that scheduling is work-conserving, that is, $\sum_i r_i(t) = C$, if there is at least one backlogged class at time t.

Remark. With the exception of the evaluation section, we take a fluid-flow interpretation of traffic throughout this paper. That is, the output link is regarded as serving simultaneously traffic from several classes. Since actual traffic is sent in discrete-sized packets, a fluid-flow interpretation of traffic is idealistic. However, scheduling algorithms that closely approximate fluid-flow schedulers with rate guarantees are available [35, 41].

We now introduce the notions of *arrival curve*, *input curve*, and *output curve* for a traffic class i in the time interval $[0, t]$. The arrival curve A_i and the input curve R_i^{in} of class i are defined as

$$A_i(t) = \int_0^t a_i(x)dx \ , \quad R_i^{in}(t) = A_i(t) - \int_0^t l_i(x)dx \ . \tag{1}$$

So, the difference between the arrival and input curve is the amount of dropped traffic. The output curve R_i^{out} of class-i is the transmitted traffic in the interval $[0, t]$, given by

$$R_i^{out}(t) = \int_0^t r_i(x)dx \ . \tag{2}$$

We refer to Figure 1(a) for an illustration. In the figure, the service rate is adjusted at times t_1, t_2, and t_4, and packet drops occur at times t_2 and t_3.

The vertical and the horizontal distance between the input and output curves from class i, respectively, are the backlog B_i and the delay D_i. This is illustrated in Figure 1(a) for time t. The delay D_i at time t is the delay of an arrival which is transmitted at time t. Backlog and delay at time t are defined as

$$B_i(t) = R_i^{in}(t) - R_i^{out}(t) \ , \quad D_i(t) = \max_{x<t}\{x \mid R_i^{out}(t) \geq R_i^{in}(t-x)\} \ . \tag{3}$$

Upon a traffic arrival, say at time s, the new service rates $r_i(s)$ and the amount of traffic to be dropped $l_i(s)$ for all classes are set such that all QoS and system constraints can be met at times greater than s. If all constraints cannot be satisfied at the same time, then some QoS constraints are relaxed in a predetermined order.

To determine the rate allocation, the scheduler makes a projection of the delays of all backlogged traffic. For the purpose of the projection, it is assumed that the current state of the link will not change after time s. Specifically, indicating projected values by a tilde ($\tilde{}$), for times $t > s$, we assume that (1) service rates remain as they are (i.e., $\tilde{r}_i(t) = r_i(s)$), (2) there are no further arrivals (i.e., $\tilde{a}_i(t) = 0$), and (3) there are no further packet drops (i.e., $\tilde{l}_i(t) = 0$).

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

$$\tilde{R}_{i,s}^{in}(t) = R_i^{in}(s) \ , \quad \tilde{R}_{i,s}^{out}(t) = R_i^{out}(s) + (t-s)r_i(s) \ , \quad \tilde{B}_{i,s}(t) = \tilde{R}_{i,s}^{in}(t) - \tilde{R}_{i,s}^{out}(t) \ . \tag{4}$$

We refer to the *projected horizon* for class i at time s, denoted as $\tilde{T}_{i,s}$, as the time when the projected backlog becomes zero, i.e., $\tilde{T}_{i,s} = \min_{x>0}\{x \mid \tilde{B}_{i,s}(s+x) = 0\}$. With this notation, we can make predictions for delays in the time interval $[s, s + \tilde{T}_{i,s}]$. We define the projected delay $\tilde{D}_{i,s}$ as

$$\tilde{D}_{i,s}(t) = \max_{t-s<x<t}\{x \mid \tilde{R}_{i,s}^{out}(t) \geq R_i^{in}(t-x)\} \ . \tag{5}$$

If there are no arrivals after time s, the delay projections are correct, in the sense that the projected delay at time s is the delay that will be encountered when the traffic element departs the system at time t. In Figure 1(b), we illustrate the projected input curve, projected output curve, and projected delays for projections made at time s. In the figure, all values for $t > s$ are projections and are indicated by dashed lines. The figure includes the projected delays for times t_5 and t_6.

4 Service Rate Adaptation and Drop Algorithm

In this section we discuss an algorithm to perform the service rates allocation to classes and the decision to drop traffic in terms of an optimization problem.

Each time s at which an arrival occurs, a new optimization is performed. The optimization variable is a time-dependent vector $\mathbf{x}_s = (r_1(s) \ldots r_Q(s)\ l_1(s) \ldots l_Q(s))^T$, which contains the service rates $r_i(s)$ and the amount of traffic to be dropped $l_i(s)$. The optimization problem has the form

$$\begin{array}{ll} \textbf{Minimize} & F(\mathbf{x}_s) \\ \textbf{Subject to} & g_j(\mathbf{x}_s) = 0, \quad j = 1, \ldots, M \\ & h_j(\mathbf{x}_s) \geq 0, \quad j = M+1, \ldots, N, \end{array} \tag{6}$$

where $F(.)$ is an objective function, and the g_j's and h_j's are constraints. The objective function, which will be presented in Subsection 4.2, will be chosen so that the amount of dropped traffic and the changes to the current service rate allocation are minimized. The constraints of the optimization problem are QoS constraints and system constraints. The optimization at time s is done with knowledge of the system state before time s, that is the optimizer knows R_i^{in} and R_i^{out} for all times $t < s$, and A_i for all times $t \leq s$.

In the remainder of this section we present the constraints and the optimization function. The optimization can be used as a reference system against which practical scheduling and dropping algorithms can be compared.

4.1 System and QoS Constraints

There are two types of constraints. *System constraints* describe constraints and properties of the output link, and *QoS constraints* define the desired service differentiation.

System Constraints. The system constraints specify physical limitations and properties at the output link. The first such constraint states that the total backlog cannot exceed the buffer size B, that is, $\sum_i B_i(t) \leq B$ for all times t. The second system constraint enforces that scheduling at the output link is work-conserving. At a work-conserving link, $\sum_i r_i(t) = C$ holds for all times t where $\sum_i B_i(t) > 0$. Note that systems that are not work-conserving, i.e., where the link may be idle even if there is a positive backlog, are undesirable for networks that need to achieve a high resource utilization. Other system constraints enforce that transmission rates and loss rates are non-negative. Also, the amount of traffic that can be dropped is bounded by the current backlog. So we obtain $r_i(t) \geq 0$ and $0 \leq l_i(t) \leq B_i(t)$ for all times t.

QoS Constraints. We consider two types of QoS constraints, relative constraints and absolute constraints. QoS constraints are either constraints on delays or constraints on the loss rate. The number and type of QoS constraints is not limited. Since absolute QoS constraints may result in an infeasible system of constraints, one or more constraints may need to be relaxed at certain times. We assume that the set of QoS constraints is assigned some total order, and that constraints are relaxed in the given order until the system of constraints becomes feasible. In addition, QoS constraints for classes which are not backlogged are simply ignored.

Absolute delay constraints (ADC) enforce that the projected delays of class i satisfy a worst-case bound d_i. That is,

$$\max_{s<t<s+\tilde{T}_{i,s}} \tilde{D}_{i,s}(t) \leq d_i \ , \tag{7}$$

for all $t \in [s, s + \tilde{T}_{i,s}]$. If this condition holds for all s, the delay bound d_i is never violated.

Relative delay constraints (RDC) specify the proportional delay differentiation between classes. As an example, for two classes 1 and 2, the RDC enforces a relationship

$$\frac{\text{Delay of Class 2}}{\text{Delay of Class 1}} \approx \text{constant} \ .$$

While absolute delay constraints are needed by applications which present stringent QoS requirements, relative differentiation is useful for differentiating traffic with less stringent QoS requirements in times of resource contention [13]. Since, in general, there are several packets backlogged from a class, each likely to have a different delay, the notion of 'delay of class i' needs to be further specified. For example, the delay of class i could be specified as the delay of the packet at the head of the class-i queue, the maximum projected delay as in Eqn. (7), or via other measures. We choose a measure, called *average projected delay* $\overline{D}_{i,s}$, which is the time average of the projected delays from a class, averaged over the horizon $\tilde{T}_{i,s}$. That is,

$$\overline{D}_{i,s} = \frac{1}{\tilde{T}_{i,s}} \int_s^{s+\tilde{T}_{i,s}} \tilde{D}_{i,s}(x)dx \ . \tag{8}$$

To provide some flexibility in the scheduling decision, we do not enforce relative delay constraints strictly, but allow for some slack. Using the metric defined in Eqn. (8), and translating the notion of slack into a tolerance level, we can write the relative delay constraints as

$$k_i(1-\varepsilon) \leq \frac{\overline{D}_{i+1,s}}{\overline{D}_{i,s}} \leq k_i(1+\varepsilon) \ , \tag{9}$$

where $k_i > 1$ is a constant defining the proportional differentiation desired, and ε ($0 \leq \varepsilon \leq 1$) indicates a tolerance level. If relative constraints are not specified for some classes, the constraints are adjusted accordingly. Note that in the delay constraints in Eqs. (7) and (9), all values with exception of the components of the optimization variable $\mathbf{x}_s$ are known at time s.

Next we discuss constraints on the loss rate. Similar to delays, there are several sensible choices for defining 'loss'. Here, we select a loss measure, denoted by $p_{i,s}$, which expresses the fraction of lost traffic since the beginning of the current busy period at time t_0.[3] So, $p_{i,s}$ expresses the fraction of traffic that has been dropped in the time interval $[t_0, s]$, that is,[4]

$$p_{i,s} = \frac{\int_{t_0}^{s} l_i(x)dx}{\int_{t_0}^{s} a_i(x)dx} = 1 - \frac{R_i^{in}(s^-) + (a_i(s) - l_i(s)) - R_i^{in}(t_0)}{A_i(s) - A_i(t_0)} \ . \tag{10}$$

In the last equation, all values except $l_i(s)$ are known at time s. With this definition we now specify absolute and relative constraints on the loss rates.

An *absolute loss constraint (ALC)* specifies that the loss rate of class i, as defined above, never exceeds a limit L_i, that is,

$$p_{i,s} \leq L_i \ . \tag{11}$$

Relative loss constraints (RLC) specify the desired proportional loss differentiation between classes. Similar to the RDCs, we provide a certain slack within these constraints. The RLC for classes $(i+1)$ and i has the form

$$k_i'(1-\varepsilon') \leq \frac{p_{i+1,s}}{p_{i,s}} \leq k_i'(1+\varepsilon') \ , \tag{12}$$

where $k_i' > 1$ is the target differentiation factor, and ε' ($0 \leq \varepsilon' \leq 1$) indicates a level of tolerance. Note that the constraints defined by Eqs. (7), (9), (11) and (12) are expressed in terms of delays and loss ratios, but the only parameters the system can control at time s are the components of the optimization variable $\mathbf{x}_s$, that is, the service rates $r_i(s)$ and the packet drops $l_i(s)$. We refer to [27] for the expression of the constraints defined by Eqs. (7), (9), (11) and (12) as functions of the service rates and the packet drops.

4.2 Objective Function

Provided that the QoS and system constraints can be satisfied, the objective function will select a solution for $\mathbf{x}_s$. Even though the choice of the objective function is a policy decision, we select two specific objectives, which, we believe, have general validity: (1) *avoid dropping traffic* , and (2) *avoid changes to the current service rate allocation*. The first objective ensures that traffic is dropped only if there is no alternative way to satisfy the constraints. The second objective tries to hold on to a feasible service rate allocation as long as possible. We give the first objective priority over the second objective.

[3] A busy period is a time interval with a positive backlog of traffic. For time x with $\sum_i B_i(x) > 0$, the beginning of the busy period is given by $\sup_{y<x}\{\sum_i B_i(y) = 0\}$.

[4] $s^- = s - h$, where $h > 0$ is infinitesimally small.

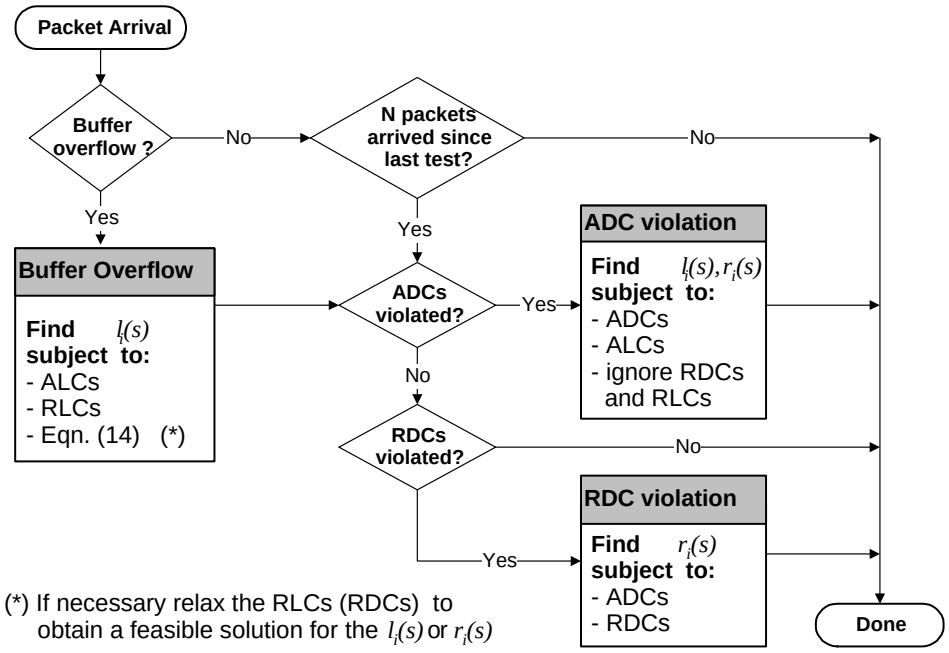


Figure 2: **Outline of the Heuristic Algorithm**.

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

$$F(\mathbf{x}_s) = \sum_{i=1}^{Q} (r_i(s) - r_i(s^-))^2 + C^2 \sum_{i=1}^{Q} l_i(s) \ , \tag{13}$$

where C is the link capacity. The first term expresses the changes to the service rate allocation and the second term expresses the losses at time s. Note that, at time s, $r_i(s)$ is part of the optimization variable, while $r_i(s^-)$ is a known value. In Eqn. (13) we use the quadratic form $(r_i(s) - r_i(s^-))^2$, since $\sum_i (r_i(s) - r_i(s^-)) = 0$ for a work-conserving link with a backlog at time s. The scaling factor C^2 in front of the second sum of Eqn. (13) ensures that traffic drops are the dominating term in the objective function.

This concludes the description of the optimization process in JoBS. The structure of constraints and objective function makes this a *non-linear optimization problem*, which can be solved with available numerical algorithms [37].

5 Heuristic Approximation

We next present a heuristic that approximates the optimization presented in the previous section, with significantly lower computational complexity. The presented heuristic should be regarded as a first step towards a router implementation.

Approximating a non-linear optimization problem such as the one presented in Section 4 can be performed by well-known techniques such as fuzzy systems, or neural networks. However, these techniques can be computationally expensive. Therefore, we choose a different approach, which decomposes the optimization problem into several computationally less intensive problems. The heuristic algorithm presented here maintains a feasible rate allocation until a buffer overflow occurs or a delay violation is predicted. At that time, the heuristic picks a new feasible rate allocation and/or drops traffic. Unless there is a buffer overflow, the tests for violations of ADCs and RDCs are not performed for every packet arrival, but only periodically.

A set of constraints, which contains absolute constraints (ALCs or ADCs), may be infeasible at certain times. Then, some constraints need to be relaxed. In our heuristic algorithm, the constraints are prioritized in the following order: system constraints have priority over absolute constraints, which in turn have priority over relative constraints. If the system of constraints becomes infeasible, the heuristic relaxes the relative constraints (RLCs or RDCs). If this does not yield a feasible solution, the heuristic relaxes one or more absolute constraints.

A high-level overview of the heuristic algorithm is presented in Figure 2. The algorithm consists of a number of small computations, one for each situation which requires to adjust the service rate allocation and/or to drop packets. We next present each of these situations and the associated computation.

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

$$\sum_i l_i(s) = \sum_i a_i(s) \; . \tag{14}$$

The heuristic picks a solution for the $l_i(s)$ which satisfies Eqn. (14) and the RLCs in Eqn. (12), where we set $\varepsilon' = 0$ to obtain a unique solution. If the solution violates an ALC, the RLCs are relaxed until all ALCs are satisfied. Once the $l_i(s)$'s are determined the algorithm continues with a test for delay constraint violations, as shown in Figure 2. The algorithm only specifies the amount of traffic which should be dropped from a particular class, however, the algorithm does not select the position in the queue from which to drop traffic. As we did for the optimization-based algorithm, we assume a Drop-Tail dropping policy for the heuristic presented in this section.

If there are no buffer overflows, the algorithm makes projections for delay violations only once for every N packet arrivals. The selection of N represents a tradeoff between the runtime complexity of the algorithm and performance of the scheduling with respect to satisfying the constraints. Simulation experiments, as described in Section 6, were performed with the value $N = 100$ and exhibit good performance.

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

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

Case 2: RDC violation. If some RDC (but no ADC) is violated, the heuristic algorithm determines new rate values. Here, the RDCs as defined in Eqn. (9) are transformed into equations by setting $\varepsilon = 0$. Together with the work-conserving property, one obtains a system of equations, for which the algorithm picks a solution. If the solution violates an ADC, the RDCs are relaxed until the ADCs are satisfied.

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

The heuristic starts with a conservative estimate of the worst-case delay for the class-i backlog at time s. For this, the heuristic uses the fact that for all $x \in [s, s + \tilde{T}_{i,s}]$, $\tilde{D}_{i,s}(x) \leq D_i(s) + \frac{B_i(s)}{r_i(s)}$, which can be verified by referring to Figures 1(a) and 1(b). Then, using $B_i(s) = B_i(s^-) + a_i(s) - l_i(s)$, we can write a sufficient condition for satisfying the ADC of class i with delay bound d_i at time s,

$$\underbrace{\frac{1}{r_i(s)} \frac{B_i(s^-) + a_i(s) - l_i(s)}{d_i - D_i(s)}}_{\rho_i} \leq 1 \; . \tag{15}$$

The heuristic algorithm will select the $r_i(s)$ and $l_i(s)$ such that Eqn. (15) is satisfied for all i. Initially, rates and traffic drops are set to $r_i(s) = r_i(s^-)$ and $l_i(s) = 0$. Since at least one ADC is violated, there is at least one class with $\rho_i > 1$, where ρ_i is defined in Eqn.(15). Now, we apply a greedy method which tries to

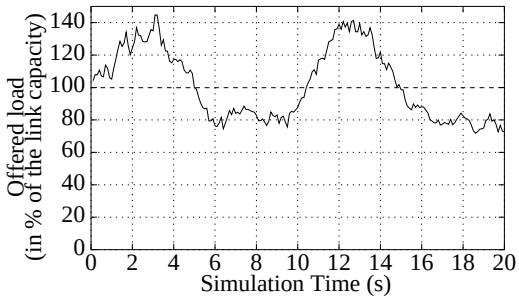


Figure 3: **Offered Load.**

redistribute the rate allocations until $\rho_i \leq 1$ for all classes. This is done by reducing $r_i(s)$ for classes with $\rho_i < 1$, and increasing $r_i(s)$ for classes with $\rho_i > 1$. If it is not feasible to achieve $\rho_i \leq 1$ for all classes by adjusting the $r_i(s)$'s, the $l_i(s)$'s are increased until $\rho_i \leq 1$ for all i. To minimize the number of dropped packets, $l_i(s)$ is never increased to a point where an ALC is violated.

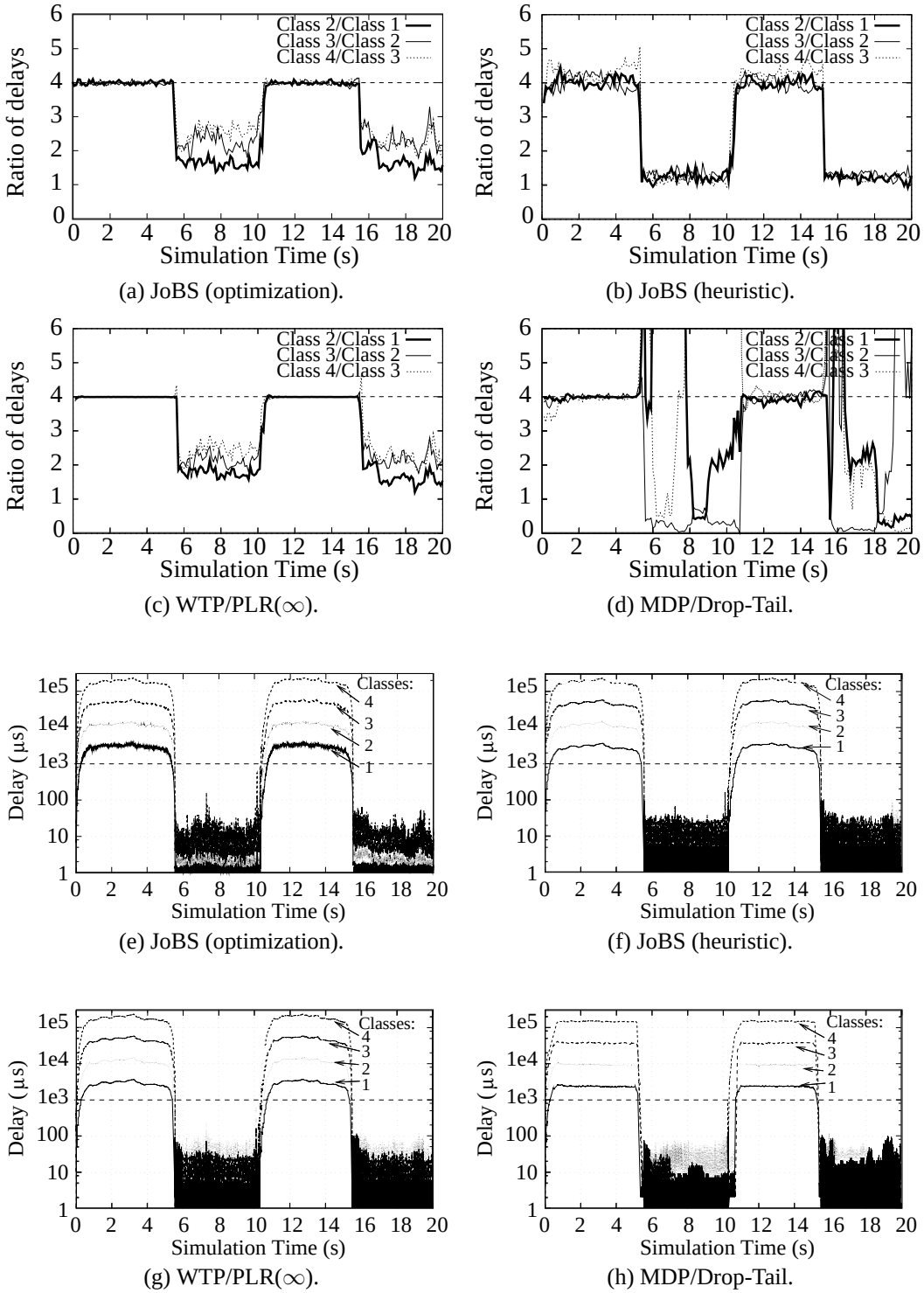
6 Evaluation

We present an evaluation of the algorithms developed in this paper via simulation. Our goals are (1) to determine if and how well the desired service differentiation is achieved; (2) to determine how well the heuristic algorithm from Section 5 approximates the optimization from Section 4; (3) to compare our algorithm with existing proposals for proportional differentiated services; and (4) to examine the effect of JoBS on end-to-end flows.

In the simulations, we evaluate the following four schemes.

- **JoBS (optimization)** refers to the optimization algorithm described in Section 4,
- **JoBS (heuristic)** is the heuristic algorithm discussed in Section 5. Unless there is a buffer overflow, tests for delay violations are performed once for every $N = 100$ packet arrivals.
- **WTP/PLR(∞) [14]:** The dropping and scheduling algorithm WTP/PLR(M) and WTP/PLR(∞) from [14] are discussed in Section 2. Since WTP/PLR(∞) provides a better service differentiation, we only include results for WTP/PLR(∞). Note that WTP/PLR(∞) does not support absolute guarantees to traffic classes.
- **MDP[32]/Drop-Tail:** The MDP scheduler presented in [32] was discussed in Section 2. Since MDP does not provide mechanisms for loss differentiation, we assume a simple Drop-Tail algorithm for discarding packets. As WTP/PLR(∞), MDP does not support absolute QoS guarantees.

We present three simulation experiments. In the first experiment, we compare the relative differentiation provided by JoBS (optimization), JoBS (heuristic), WTP/PLR(∞), and MDP/Drop-Tail without specifying absolute constraints, at a single node. In the second experiment, we augment the set of constraints by absolute loss and delay constraints on the highest priority class, and show that JoBS can effectively provide both relative and absolute differentiation at a single node. These first two experiments do not take into account the possible effect of the transport protocol being used, nor the effect of JoBS on end-to-end traffic. In the third experiment, we examine the effect of JoBS on end-to-end traffic in a multi-node network, with a mix of TCP and UDP traffic.



(a) JoBS (optimization). (b) JoBS (heuristic).

(c) WTP/PLR(∞). (d) MDP/Drop-Tail.

(e) JoBS (optimization). (f) JoBS (heuristic).

(g) WTP/PLR(∞). (h) MDP/Drop-Tail.

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

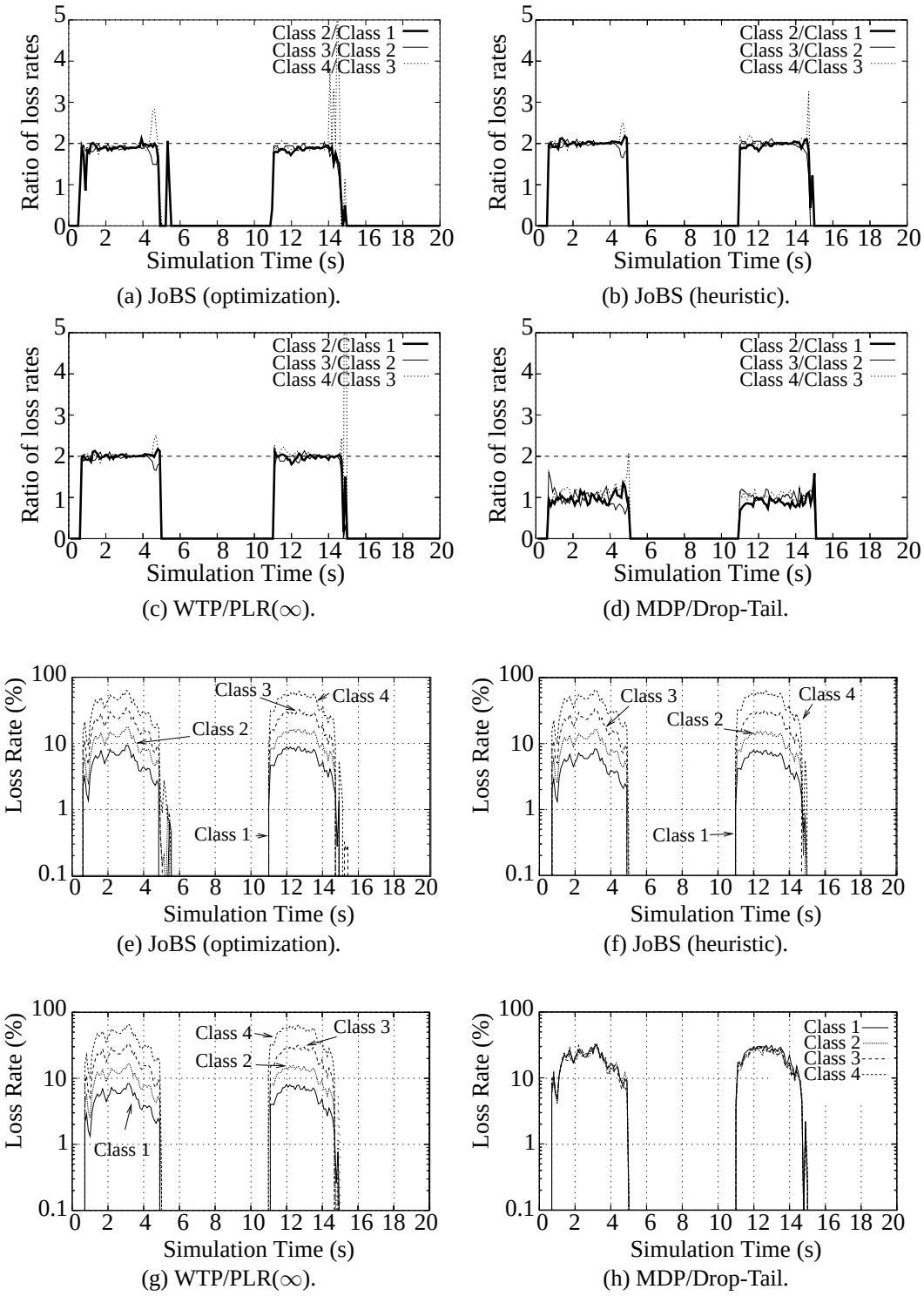


Figure 5: **Experiment 1: Relative Loss Differentiation.** The graphs show the ratios of loss rates for successive classes (a)-(d) and the loss rates (e)-(h). The target value for the ratios, indicated by a dashed line, is $k' = 2$.

6.1 Simulation Experiment 1: Relative Differentiation Only

The first experiment focuses on relative service differentiation, and does not include absolute constraints. [5] We consider a single output link with capacity $C = 1$ Gbps and a buffer size of 6.25 MB. We assume $Q = 4$ classes. The length of each experiment is 20 seconds of simulated time, starting with an empty system. In all experiments, the incoming traffic is composed of a superposition of Pareto sources with $\alpha = 1.2$ and average interarrival time of 300 μs. The number of sources active at a given time oscillates between 200 and 550, following a sinusoidal pattern. All sources generate packets with a fixed size of 125 bytes. [6] The resulting offered load is plotted in Figure 3. At any time, each class contributes 25% of the aggregate load, yielding a symmetric load. In a realistic environment, one would expect to have "less" high priority traffic than low priority traffic. Therefore, a symmetric load can be regarded as a realistic worst-case that can occur during bursts of high-priority traffic.

The objectives for the relative differentiation are so that we want to have a ratio of four between the delays of two successive classes, and a ratio of two between the loss rates of two successive classes. Thus, for JoBS, we set $k_i = 4$ and $k'_i = 2$ for all i. The tolerance levels are set to $(\varepsilon, \varepsilon') = (0.001, 0.05)$ in JoBS (optimization), and to $\varepsilon = 0.01$ in JoBS (heuristic). The results of the experiment are presented in Figures 4 and 5, where we graph the ratios of delays and loss rates, respectively, of successive classes for JoBS (optimization), JoBS (heuristic), WTP/PLR(∞), and MDP/Drop-Tail. The plotted delay and loss values are averages over moving time windows of size 0.1 s.[7]

When the link load is above 90% of the link capacity, that is, in time intervals $[0\ s, 6\ s]$ and $[10\ s, 15\ s]$, all methods provide the desired service differentiation. The oscillations around the target values in JoBS (optimization) and JoBS (heuristic) are mostly due to the tolerance values ε and ε'. The selection of the tolerance values ε and ε' in JoBS presents a tradeoff: smaller values for ε and ε' reduce oscillations, but incur more work for the algorithms. When the system load is low, that is, in time intervals $[6\ s, 10\ s]$ and $[16\ s, 20\ s]$, only JoBS (optimization) and WTP/PLR(∞) manage to achieve some delay differentiation, albeit far from the target values. MDP/Drop-Tail, plotted in Figure 4(d), provides some differentiation, but the system seems unstable, particularly after a transient change in the load. However, at an underloaded link, the absolute values of the delays are very small for all classes, regardless of the scheduling algorithm used, as shown on Figures 4(e)-(h). Figures 4(e)-(h) also show that the absolute values for the delays are comparable in all schemes.

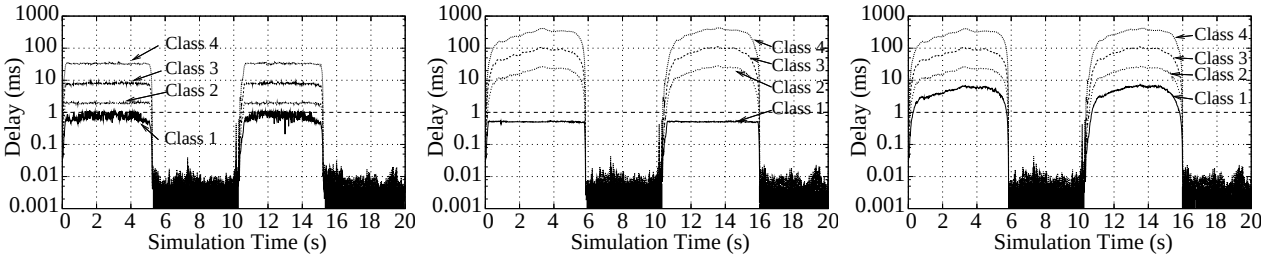
In Figures 5(a) and 5(c), we observe that both WTP/PLR(∞) and JoBS (optimization) show some transient oscillations with respect to loss differentiation when the link changes from an overloaded to an underloaded state, while JoBS (heuristic) does not seem to suffer from this problem as much. Without offering an explanation, we speculate that during a transition between an overloaded and an underloaded system, a perfect relative loss differentiation is not achievable without violating the workconserving property or without dropping packets even if the buffer is not full. MDP/Drop-Tail does not exhibit these transient oscillations since it does not provide any loss differentiation, as shown in Figure 5(d).

Finally, the total loss rate is of interest, as a scheme may provide excellent proportional loss differentiation, but have an overall high loss rate. Figures 5(e)-(g) prove that in the simulations, the loss rates of WTP/PLR(∞) and JoBS (optimization and heuristic) are very similar. With both WTP/PLR(∞) and JoBS (optimization and heuristic), the average of the per-class loss rates is equal to the loss rate obtained with a Drop-Tail policy, plotted in Figure 5(h). This shows that, in this experiment, all schemes only drop packets when a buffer overflow occurs.

[5] The simulator used for Experiments 1 and 2 is described at `http://qosbox.cs.virginia.edu/snooplet.html`.

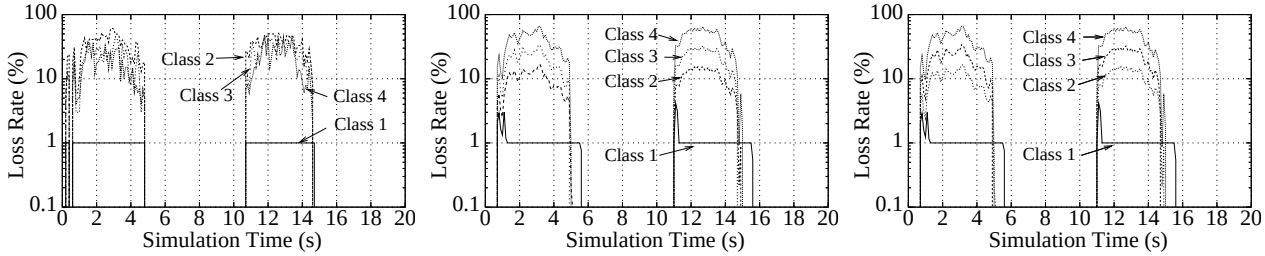
[6] Packet sizes on the Internet are in fact subject to a multimodal distibution [2], and thus, the simulation presented here is only a simplified model.

[7] This measure is adopted from [14].



(a) With ADC, all RDCs. (b) With ADC, one RDC removed. (c) No ADC, all RDCs.

Figure 6: **Experiment 2: Absolute Delay Differentiation.** The graphs show the delays of all packets. All results are for JoBS (heuristic).



(a) With ADC, all RDCs. (b) With ADC, one RDC removed. (c) No ADC, all RDCs.

Figure 7: **Experiment 2: Absolute Loss Differentiation.** The graphs show the loss rates of all classes. All results are for JoBS (heuristic).

6.2 Simulation Experiment 2: Relative and Absolute Differentiation

In this second experiment, we evaluate how well our algorithm can satisfy a mix of absolute and relative constraints on both delays and losses. Here, we only present results for JoBS (heuristic). WTP/PLR(∞) and MDP/Drop-Tail do not support absolute guarantees. The main objective of this second experiment is to show that, compared to schemes that treat dropping and scheduling orthogonally, using a joint buffer management and rate allocation approach such as JoBS allows for satisfying both absolute and relative constraints at the same time by dropping traffic when an absolute delay constraint is on the verge of being violated.

We consider the same simulation setup and the same relative delay constraints as in Experiment 1, but add an absolute delay constraint (ADC) for Class 1 such that $d_1 = 1\ ms$, and we replace the relative loss constraint (RLC) between Classes 1 and 2 by an absolute loss constraint (ALC) for Class 1 such that $L_1 = 1\%$. We call this scenario "with ADC, all RDCs". With the given relative delay constraints from Experiment 1, the other classes have implicit absolute delay constraints, which are approximately[8] 4 ms for Class 2, 16 ms for Class 3, and 64 ms for Class 4. Removing the RDC between Class 1 and Class 2, we avoid the 'implicit' absolute constraints for Classes 2, 3, and 4, and call the resulting constraint set "with ADC, one RDC removed". We also include the results for JoBS (heuristic) from Experiment 1, with the ALC on Class 1 replacing the RLC between Classes 1 and 2, and refer to this constraint set as "no ADC, all RDCs". In Figure 6 we plot the absolute delays of all packets, and in Figure 7 we plot the loss rates of all classes, averaged over time intervals of length 0.1 s. We discuss the results for each of the three constraint sets proposed.

Concerning the experiment "with ADC, all RDCs", Figure 6(a) shows that the heuristic maintains the relative delay differentiation between classes, thus, enforcing the 'implicit' delay constraints for Classes 2, 3, and 4. With a large number of absolute delay constraints, the system of constraints easily becomes

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

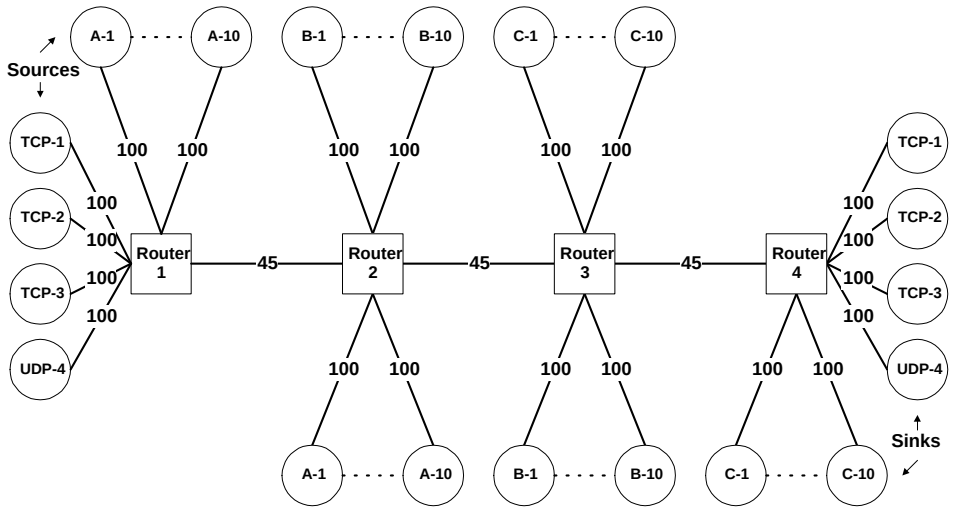


Figure 8: **Experiment 3: Network Topology.** The numbers on the links denote the links capacities in Mb/s.

infeasible, as pointed out by the following two observations. First, Figure 7(a) shows that the loss rates of Classes 2, 3 and 4 are similar. This result illustrates that the heuristic relaxes relative loss constraints to meet the absolute delay constraints. Second, Figure 6(a) shows that the absolute delay constraint d_1 is sometimes violated. However, such violations are rare (over 95% of Class-1 packets have a delay less than 900 μs), and Class-1 packet delays always remain reasonably close to the delay bound d_1. For the experiment "with ADC, one RDC removed", Figure 6(b) shows that, without an RDC between Classes 1 and 2, the ratio of Class-2 delays and Class-1 delays can exceed a factor of 10 at high loads. With this constraint set, the absolute delay constraint d_1 is never violated, and Figure 7(b) shows the RLCs are consistently enforced during periods of packet drops. Finally, for the experiment "no ADC, all RDCs", Figure 6(c) shows that, without the ADC, the delays for Class 1 are as high as 5 ms.[9]

6.3 Experiment 3: Multiple Node Simulation with TCP and UDP Traffic

Finally, we present a multinode simulation, to see if our approach is still able to provide the desired service, in the context of a mix of TCP and UDP traffic, with multiple hops and propagation delays. We also want to examine the level of quality of service an *end-to-end* flow can receive with the proposed per-node guarantees. To that effect, we implemented a variant of JoBS (heuristic) [10] in the *ns-2* network simulator [1].

For this third experiment, we simulate a network with a topology as shown in Figure 8. We have four routers connected by three 45 Mbps links, and sources and sinks connected to the routers by independent 100 Mbps links. Each 45 Mbps link has a propagation delay of 3 ms, and each 100 Mbps link has a propagation delay of 1 ms. There are four classes of traffic. The composition of the traffic mix is given in Table 1 and the service guarantees are given in Table 2.

Cross-traffic flows (denoted by A-1, ..., C-10) start transmitting at time $t = 0$ s. The flows TCP-1, TCP-2, TCP-3 and UDP-4 start transmitting at time $t = 10$ s. All flows consists of packets with a fixed size of 500 Bytes, and the experiment lasts 70 seconds of simulated time.

From Tables 1 and 2, Classes 1, 2 and 3 only consist of TCP traffic, and Class 4 only consists of UDP traffic. The offered load is asymmetric, since initially Class 1 contributes 10% of the aggregate cross-traffic,

[9]The delay values for Classes 2, 3, and 4 in Figures 6(b) and (c) appear similar, especially since we use a log-scale. We emphasize that the values are *not* identical, and that the results are consistent.

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

Table 1: **Experiment 3: Traffic mix.** The traffic mix for flows B-1, . . ., B-10 and C-1, . . ., C10 is identical to the traffic mix described here for flows A-1,. . .,A-10. TCP sources run the *TCP Reno* congestion control algorithms. The "Shape" parameters characterizes the Pareto distributions used.

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

Table 2: **Experiment 3: Service guarantees.** The guarantees are identical at each router.

Class 2 contributes 20%, Class 3 contributes 30% and Class-4 contributes 40 %. Thus, there is more UDP traffic in this simulation experiment than can be expected in an actual network, since UDP traffic accounts for less than 10% of the traffic on the Internet [2]. We made this choice so that we could examine the effects of mixing the two types (UDP and TCP) of traffic more easily.

Per-hop per-class QoS We graph the per-class queueing delays and per-class loss rates at each of the first three routers in Figure 9, starting at time $t = 0$ s. Given that the aggregate arrival rate at Router 4 is always less than the total output capacity of Router 4, there is never any backlog at Router 4, and thus, the queueing delays and loss rates are constantly equal to zero. With the exception of Figure 9(c), (g) and (k), where the individual packet delays are represented, each point on Figure 9 represents an average over a sliding window of size 0.5 s. Figure 9 shows that the proposed algorithm manages to enforce all proposed service guarantees at each node, with only a couple of transient violations of the absolute delay bound on Class 1 at Router 1, and that the algorithm seems to respond appropriately to transient changes such as the introduction of additional traffic at time $t = 10$ s.

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

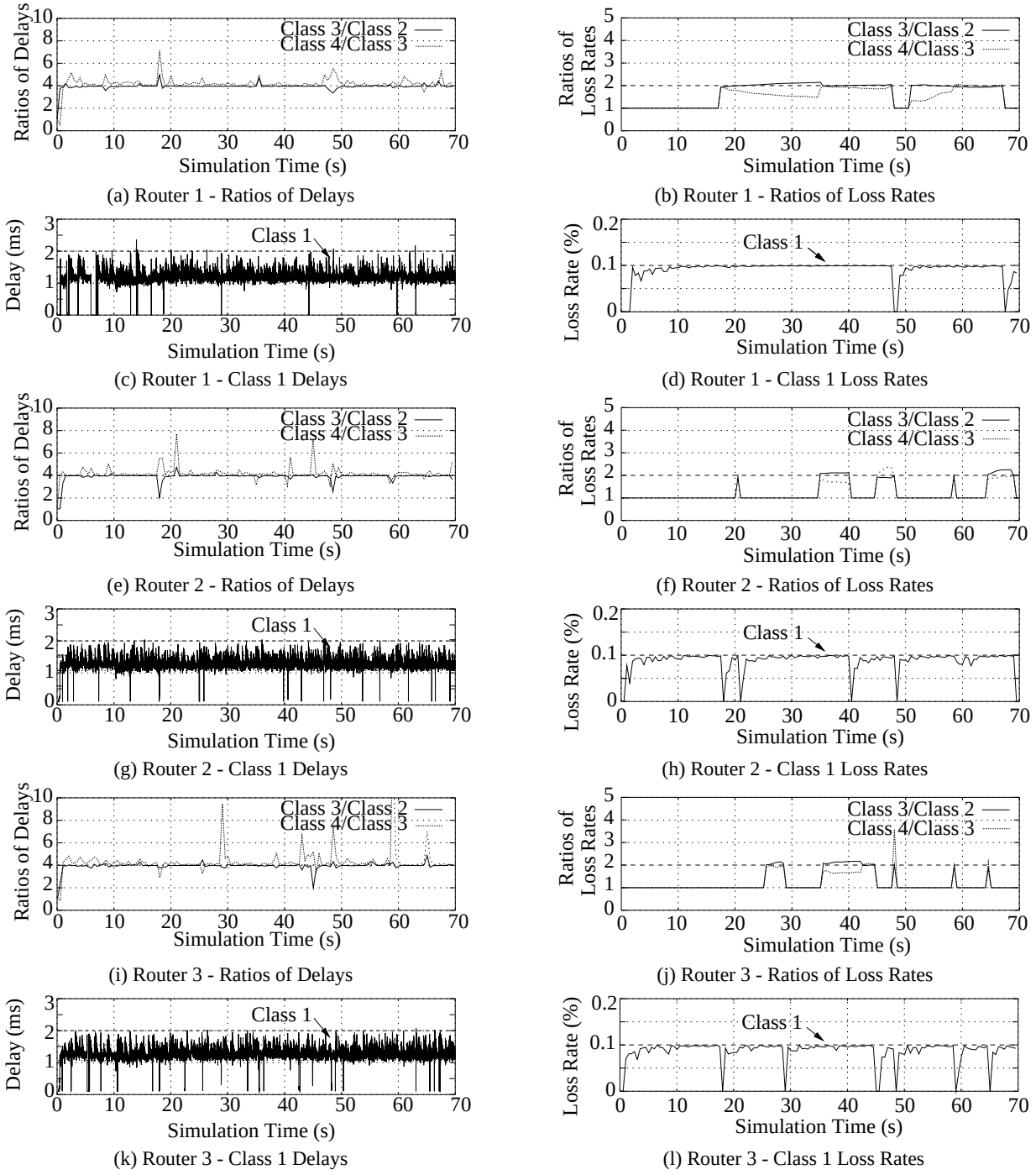
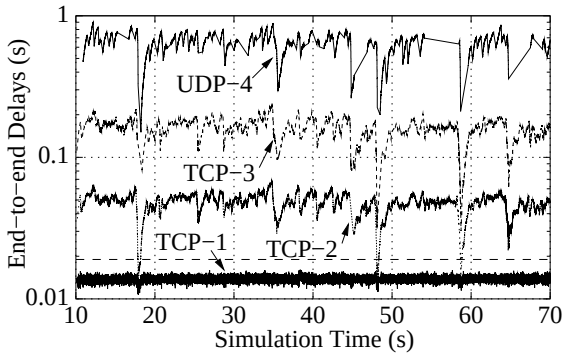
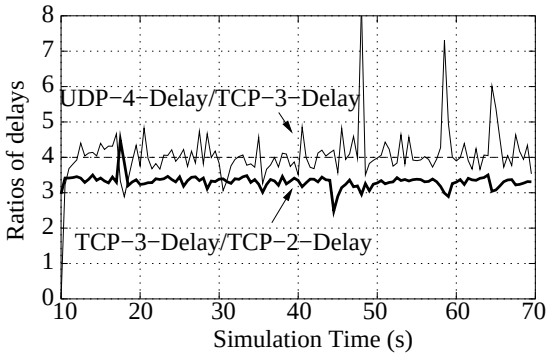


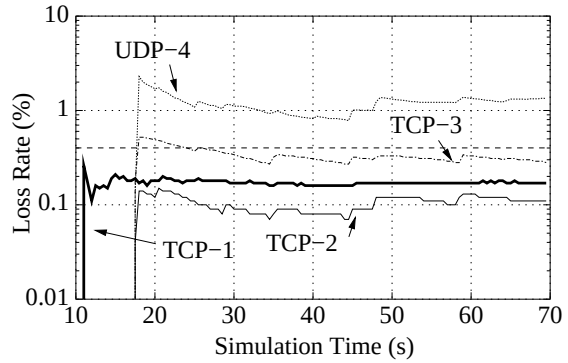
Figure 9: **Experiment 3: Multiple Node simulation with TCP and UDP traffic.** The graphs show the delays and loss rates encountered at each router by Class 1 traffic, and the ratios of delays and the ratios of loss rates for Classes 2, 3 and 4 at each router. The absolute constraints and the target ratios are indicated by straight dashed lines.



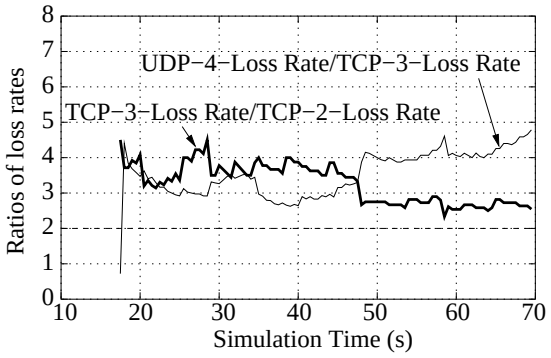
(a) End-to-End Packet Delays



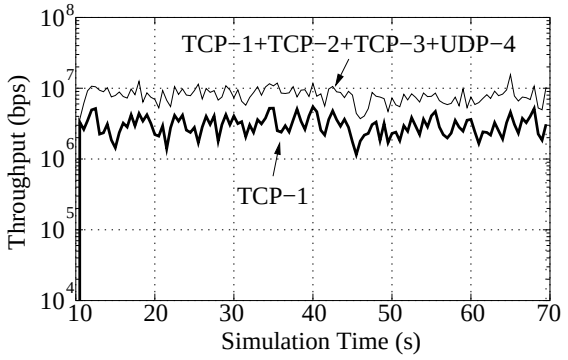
(b) End-to-End Ratios of Delays



(c) End-to-End Packet Loss Rates



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



(e) Class-1 Throughput

Figure 10: **Experiment 3: End-to-end packet delays.** The graphs represent the individual, end-to-end, packet delays encountered by flows TCP 1, TCP-2, TCP-3, UDP-4 (a), the ratios of delays over a sliding window of size 0.5 s for TCP-2, TCP-3, and UDP-4 (b), the loss rates (c), ratio of loss rates (d), and the throughput obtained by TCP-1, as well as the aggregate throughput obtained by all four flows (e). The 'implicit' end-to-end delay guarantee, indicated by a dashed line, on TCP-1 is a delay bound of 19 ms, the 'implicit' end-to-end loss guarantee on TCP-1, also indicated by a dashed line, is 0.4%.

In Figure 10(a), we present the individual end-to-end packet delays encountered by each flow. Flow 1's end-to-end delays are indeed always below 19 ms, and we see again that the algorithm we propose uses a conservative estimate of the delays for enforcing delay bounds, since most Flow 1 packets encounter a total delay close to 15 ms. [10] Figure 10(b) also suggests that the proportional delay differentiation holds with respect to the end-to-end delays between Classes 3 and 4, even if the relative delay constraints are enforced only on a per-node basis. This result can be explained by the fact that the propagation delays are negligible compared to the queueing delays encountered by TCP-3, and UDP-4. Conversely, the propagation delay cannot be neglected compared to the queueing delays in the case of the flow TCP-2, hence, the proportional differentiation between TCP-2 and TCP-3 is close to a factor of 3.3.

We plot the end-to-end packet loss rates in Figure 10(c). We see that the end-to-end loss rate bound of 0.4% on flow TCP-1 is respected, even though the load is asymetric. [11] However, as shown in Figure 10(d), proportional guarantees on loss rates between classes do not translate into proportional guarantees between end-to-end flows: loss rates ratios between TCP-2, TCP-3, and UDP-4 are consistently above the desired ratios $k'_2 = k'_3 = 2$. This result is mostly due to the fact that the different flows present in the network do not have the same probability of suffering packet drops, since some of them are greedy flows, while others are on-off flows.

Last, in Figure 10, we graph the throughput received by flow TCP-1, as well as the aggregate throughput received by flows TCP-1, TCP-2, TCP-3 and UDP-4. There is no throughput guarantee on any class, but we see that the flow TCP-1 consistently gets an end-to-end throughput greater than 1.5 Mbps, and close to 3 Mbps in general. This result shows that the strong guarantees on the loss and delay of Class 1 are not realized at the expense of a low throughput. For readability purposes, we do not show the throughput plots for the three other flows, which present values close to that of TCP-1.

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

7 Conclusions

We proposed an algorithm, called JoBS (Joint Buffer Management and Scheduling), for relative and absolute per-class QoS guarantees without information on traffic arrivals. At times when not all absolute QoS guarantees can be satisfied simultaneously, JoBS selectively ignores some of the QoS guarantees. The JoBS algorithm reconciles rate allocation and buffer management into a single scheme, thereby acknowledging that scheduling and dropping decisions at an output link are not orthogonal issues, but should be addressed together. JoBS implements the desired service differentiation based on delay predictions of backlogged traffic. The predictions are used to update service rate allocations to classes and the amount of traffic to be dropped. We showed in a set of simulation experiments, that JoBS can provide relative and absolute per-class QoS guarantees for delay and loss, thereby demonstrating the virtue of an approach combining scheduling and dropping, and examined the effect of the algorithm on end-to-end flows.

In future work, we will extend the approach presented in this paper to TCP congestion control. As a point of departure, we will attempt to express existing active queue management schemes, e.g., RED [18],

[10] Even if we do not take into account the queueing delays at Router 4, which are always zero due to the topology, in the computation of the end-to-end delay guarantee, the end-to-end delay guarantee becomes 16 ms, and is respected as well.

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

RIO [11], or the proportional-integral controller of [20] within the formal framework introduced in this paper. Additionally, we are currently working on router implementations of JoBS-style algorithms [9].

References

[1] *ns-2* network simulator. http://www.isi.edu/nsnam/ns/.

[2] Packet sizes and sequencing, May 2001. http://www.caida.org/outreach/resources/learn/packetsizes.

[3] G. Armitage, A. Casati, J. Crowcroft, J. Halpern, B. Kumar, and J. Schnizlein. A delay bound alternative revision of RFC2598, April 2001. IETF draft (informational), draft-ietf-diffserv-efresolve-01.txt.

[4] S. Athuraliya, D. Lapsley, and S. Low. An enhanced random early marking algorithm for internet flow control. In *Proceedings of IEEE INFOCOM 2000*, pages 1425–1434, Tel-Aviv, Israel, April 2000.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. IETF RFC 2475, December 1998.

[6] U. Bodin, A. Jonsson, and O. Schelen. On creating proportional loss differentiation: predictability and performance. In *Proceedings of IWQoS 2001*, pages 372–386, Karlsruhe, Germany, June 2001.

[7] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. IETF RFC 1633, July 1994.

[8] C. S. Chang. *Performance Guarantees in Communication Networks*. Springer Verlag, London, UK, 1999.

[9] N. Christin and J. Liebeherr. The QoSbox: A PC-router for quantitative service differentiation in IP networks. Technical Report CS-2001-28, University of Virginia, November 2001. ftp://ftp.cs.virginia.edu/pub/techreports/CS-2001-28.pdf.

[10] N. Christin, J. Liebeherr, and T. F. Abdelzaher. A quantitative assured forwarding service. In *Proceedings of IEEE INFOCOM 2002*, New York, NY, June 2002. To appear.

[11] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4):362–373, August 1998.

[12] R. Cruz, H. Sariowan, and G. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proceedings of the Fourth IEEE International Conference on Computer Communications and Networks (ICCCN '95)*, pages 512–520, Las Vegas, NV, September 1995.

[13] C. Dovrolis. *Proportional differentiated services for the Internet*. PhD thesis, University of Wisconsin-Madison, December 2000.

[14] C. Dovrolis and P. Ramanathan. Proportional differentiated services, part II: Loss rate differentiation and packet dropping. In *Proceedings of IWQoS 2000*, pages 52–61, Pittsburgh, PA., June 2000.

[15] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *Proceedings of ACM SIGCOMM '99*, pages 109–120, Boston, MA., August 1999.

[16] B. Davie (editor), F. Baker, J. Bennet, K. Benson, J.-Y. Le Boudec, A. Charny, A. Chiu, W. Courtney, S. Davari, V. Firoiu, C. Kalmanek, K.K. Ramakrishnam, and D. Stiliadis. An expedited forwading PHB, September 2001. IETF draft, draft-ietf-diffserv-rfc2598bis-02.txt.

[17] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: A new class of active queue management algorithms. Technical Report CSE-TR-387-99, University of Michigan, April 1999.

[18] S. Floyd and V. Jacobson. Random early detection for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, July 1993.

[19] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[20] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1726–1734, Anchorage, AK, April 2001.

[21] P. Hurley, J.-Y. Le Boudec, P. Thiran, and M. Kara. ABE: providing low delay service within best effort. *IEEE Networks*, 15(3):60–69, May 2001. See also http://www.abeservice.org.

[22] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB. IETF RFC 2598, June 1999.

[23] L. Kleinrock. *Queueing Systems. Volume II: Computer Applications*. John Wiley & Sons, New York, NY, 1976.

[24] H. Kroner, G. Hebuterne, P. Boyer, and A. Gravey. Priority management in ATM switching nodes. *IEEE Journal in Selected Areas in Communications*, 9(3):418–427, April 1991.

[25] M. A. Labrador and S. Banerjee. Packet dropping policies for ATM and IP networks. *IEEE Communications Surveys*, 2(3), 3rd Quarter 1999. http://www.comsoc.org/pubs/surveys.

[26] T.V. Lakshman, A. Neidhardt, and T. Ott. The drop from front strategy in TCP and in TCP over ATM. In *Proceedings of IEEE INFOCOM '96*, pages 1242–1250, San Francisco, CA, March 1996.

[27] J. Liebeherr and N. Christin. Buffer management and scheduling for enhanced differentiated s ervices. Technical Report CS-2000-24, University of Virginia, August 2000.

[28] J. Liebeherr and N. Christin. JoBS: Joint buffer management and scheduling for differentiated services. In *Proceedings of IWQoS 2001*, pages 404–418, Karlsruhe, Germany, June 2001.

[29] A.-M. Lin and J.A. Silvester. Priority queueing strategies and buffer allocation protocols for traffic control at an ATM integrated broadband switching system. *IEEE Journal on Selected Areas in Communications*, 9(9):1524–1536, December 1991.

[30] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM '97*, pages 127–137, Cannes, France, September 1997.

[31] Y. Moret and S. Fdida. A proportional queue control mechanism to provide differentiated services. In *Proceedings of the International Symposium on Computer and Information Systems (ISCIS)*, pages 17–24, Belek, Turkey, October 1998.

[32] T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Barghavan. Delay differentiation and adaptation in core stateless networks. In *Proceedings of IEEE INFOCOM 2000*, pages 421–430, Tel-Aviv, Israel, April 2000.

[33] K. Nichols, V. Jacobson, and L. Zhang. Two-bit differentiated services architecture for the Internet. IETF RFC 2638, July 1999.

[34] R. Pan, B. Prabhakar, and K. Psounis. CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE INFOCOM 2000*, pages 942–951, Tel-Aviv, Israel, April 2000.

[35] A. K. Parekh and R. G. Gallagher. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[36] S. Sahu, P. Nain, D. Towsley, C. Diot, and V. Fioroiu. On achievable service differentiation with token bucket marking for TCP. In *Proceedings of ACM SIGMETRICS 2000*, pages 23–33, Santa Clara, CA, June 2000.

[37] K. Schittkowski. NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5:485–500, 1986. Edited by Clyde L. Monma.

[38] I. Stoica and H. Zhang. Providing guaranteed services without per-flow management. In *Proceedings of ACM SIGCOMM '99*, pages 81–94, Boston, MA, August 1999.

[39] A. Striegel and G. Manimaran. Packet scheduling with delay and loss differentiation. *Computer Communications*, 25(1):21–31, January 2002.

[40] B. Suter, T.V. Lakshman, D. Stiliadis, and A.K. Choudhury. Buffer management schemes for supporting TCP in gigabit routers with per-flow queueing. *IEEE Journal on Selected Areas of Communications*, 17(6):1159–1170, September 1999.

[41] L. Zhang. Virtual clock: A new traffic control algorithm for packet switched networks. *ACM Trans. Comput. Syst.*, 9(2):101–125, May 1991.