Buffer Management and Scheduling for Enhanced Differentiated Services [*]

Technical Report: University of Virginia, CS-2000-24

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

Abstract

A novel framework, called JoBS (Joint Buffer Management and Scheduling), is presented for reasoning about relative and absolute per-class service differentiation in a packet network without information on traffic arrivals. JoBS has two unique capabilities: (1) JoBS makes scheduling and buffer management decisions in a single step, and (2) JoBS supports both relative and absolute QoS requirements of classes. JoBS is presented in terms of the solution to an optimization problem. Numerical simulation examples, including results for a heuristic approximation of JoBS, are presented to illustrate the effectiveness of the approach and to compare JoBS to existing methods for loss and delay differentiation.

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-9903001.

1 Introduction

There are two important criteria for classifying Quality-of-Service (QoS) guarantees in packet networks. 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 if 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 IntServ model [4], 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. An important argument in favor of relative per-class QoS is that it does not require admission control or traffic policing. Since late 1997, the *Differentiated Services* (DiffServ) [3] working group has discussed several proposals for per-class relative QoS guarantees [6, 23, 24].

Most proposals for relative per-class QoS discussed within the DiffServ context define 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 relative QoS notions [9, 10, 21, 22]. Probably the best known such effort is the *proportional service differentiation* model [8], proposed by Dovrolis, Stiliadis, and Ramanathan, which enforces that the ratios of delays [10] or loss rates [9] of successive priority classes are 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 relative per-class QoS within a formal framework inspired by Cruz's service curves [7]. Using this approach, we present a scheduling/dropping algorithm, 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 operates as follows. After each arrival, JoBS predicts delays of the currently backlogged traffic, and then adjusts the service rate allocation to classes and the amount of traffic to be dropped. A unique feature of JoBS is that it considers scheduling and buffer management (dropping) together in a single step.

This paper is organized as follows. In Section 2 we give an overview of the state-of-the-art of relative per-class QoS guarantees. Then, in Sections 3 and 4, we specify the JoBS framework. In Section 5 we present a heuristic approximation of JoBS. In Section 6 we present simulation scenarios to evaluate the effectiveness of JoBS. In Section 7 we present brief conclusions.

2 Related Work

Due to the amount of literature on service differentiation, we limit our discussion to the relevant work on scheduling and buffer management algorithms on per-class relative guarantees. This discussion focuses on techniques devised to improve Best Effort or DiffServ services.

With one notable exception [9], 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 achieving loss differentiation.

2.1 Scheduling Algorithms

The majority of work on per-class relative QoS suggests to use existing fixed-priority, e.g., [24], or rate-based schedulers, e.g., [13]. The number of new scheduling algorithms that have been specifically designed for relative delay differentiation is small.

The Proportional Queue Control Mechanism (PQCM) [21] and Backlog-Proportional Rate (BPR) [10] are variations of the GPS algorithm [26]. 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. JoBS bears similarity to PQCM and BPR in that JoBS dynamically calculates the service rate allocation. However, the rate allocation in JoBS takes into consideration many more parameters, in addition to the current backlog.

Waiting-Time Priority (WTP) [10] implements a well-known scheduling algorithm with dynamic time-dependent priorities ([15], Ch. 3.7). Each packet is assigned a time-dependent priority as follows. Consider a tagged packet from class i, which arrives at time τ. 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 [15]. WTP packets are transmitted in the order of their priorities. In [10], 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 the following delay differentiation under high loads: Class- $(i+1)$ Delay $\approx k$·Class- i Delay.

The Mean-Delay Proportional scheduler (MDP, [22]) 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 $\overline{d_i}(t) \cdot s_i$, where $\overline{d_i}(t)$ is the estimated average delay for class- i, averaged over the entire up-time of the link. As in WTP, the coefficients s_i are set to $s_1 = k \cdot s_2 = k^2 \cdot s_3 = \ldots = k^Q \cdot s_Q$ to ensure Class- $(i+1)$ Delay $\approx k$·Class- i Delay under high loads.

An important difference of JoBS to the PQCM, BPR, WTP, and MDP schedulers discussed above is that the rate allocation in JoBS is not limited to the current state and past history of the link. In addition, JoBS makes predictions on future delays, thereby strengthening the guarantees provided by the scheduler.

Finally, we briefly discuss the relation of our work to the recently proposed Scalable-Core (SCORE) approach [30]. 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 values of the experienced delays in the packet headers. The stored information is used for adjusting the priority of packets so that end-to-end requirements are met. The CoreLite architecture [22] is an extension of this work which couples per-hop proportional delay differentiation with end-to-end delay guarantees. Different from the above schedulers, JoBS does not store any state information in packets.

2.2 Buffer Management Algorithms

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 briefly discuss the related work on these two mechanisms. We refer to a recent survey article [17] for an extensive discussion of buffer management algorithms proposed for IP and ATM networks.

Backlog Controllers Initial proposals for buffer management (also called active queue management) in IP networks [11, 12] were motivated by the need to improve TCP performance, without considering service differentiation. More recent research efforts [6, 20, 25, 28] enhance these initial proposals in order to provide service differentiation.

Among backlog controllers for IP networks, Random Early Detection (RED, [12]) 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., [2, 6, 11, 20, 25, 28]. For example, BLUE [11] 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 [6] and multiclass RED [28] are extensions to RED specifically targeted for class-based service differentiation. Both schemes have different dropping thresholds for different classes, in order to ensure differentiation at the packet loss level. A similar approach, in an IntServ context, is pursued for Flow-RED (FRED, [20]), which uses different threshold values for the different flows sharing a link, identified by their source-destination address pairs.

Random Early Marking (REM, [2]) 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 [2], 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.

Droppers The simplest and most widely used dropping scheme is Drop-Tail, which drops incoming traffic when the buffer is full. Recent implementation studies [31] 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 [18]. 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, [16, 19]), Complete Buffer Partitioning (CBP, [19]), and Partial Buffer Sharing (PBS, [16]). LPF always drops packets from the lowest backlogged priority queue. As shown in [9], LPF does not provide any mechanism for proportional loss differentiation. Likewise, CBP assigns a dedicated amount of buffer space to each class, and drops traffic when this dedicated buffer is full. PBS is 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 to support relative per-class service differentiation, since no *a priori* knowledge of the incoming traffic is available.

Early Packet Discard (EPD, [27]) and Partial Packet Discard (PPD, [1]) are dropping algorithms which have been developed specifically for ATM networks. Here, whenever a cell from a certain (AAL5) packet is dropped, these algorithms ensure that all remaining cells from that packet will be dropped as well.

The Proportional Loss Rate (PLR) dropper [9] has been specifically designed to support relative per-class QoS with a proportional service differentiation [8]. PLR enforces that the ratio of the loss rates of two successive classes remains roughly constant at an assigned value. There are two variants of this scheme. PLR(M) uses only the last M packets for estimating the loss rates of a class, whereas PLR(∞) has no such memory constraints.

The majority of related work regards delay and loss differentiation as orthogonal issues. A notable exception is the most recent revision of the *Proportional Differentiated Services* model [9], which provides mechanisms for both proportional loss and delay differentiation. However, in [9], the scheduling and dropping decisions are made independently, by separate algorithms. In the remainder of this paper, we will show that performing scheduling and dropping in a single algorithm, we can provide stronger service guarantees without sacrificing the scalability benefits of a DiffServ architecture.

3 The Joint Buffer Management and Scheduling Framework

In this section, we introduce our framework of *Joint Buffer Management and Scheduling* (JoBS), for scheduling and buffer management at the output link of a router. We will first give an informal discussion of the operations of JoBS and then provide a detailed description.

3.1 Overview of JoBS

JoBS assumes per-class buffering of arriving traffic and serves traffic from the same class in a First-Come-First-Served order. JoBS allocates to each traffic class a guaranteed service rate. The service rate guarantees are adjusted over time and may be changed as often as after each traffic arrival. Within the context of JoBS, there is no admission control and no policing of traffic.

The set of relative or absolute performance requirements are given to the JoBS algorithms as a set of per-class QoS constraints. As an example, for three classes, the QoS constraints could be of the form:

- Class-2 Delay $\approx 2 \cdot$ Class-1 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. The set of constraints given to JoBS can be any mix of relative and absolute constraints. Note that absolute constraints may render a system of constraints infeasible. Then, some constraints may need to be relaxed. We assume that JoBS is provided with an order in which constraints are to be relaxed in case of an infeasible state.

The JoBS algorithm operates as follows. For every arrival, JoBS makes a prediction on the delays of the backlogged traffic, and modifies the service rates so that all QoS and system constraints will be met. If changing the service rates is not sufficient for meeting all constraints, JoBS will drop either the arrival or it will drop queued traffic.

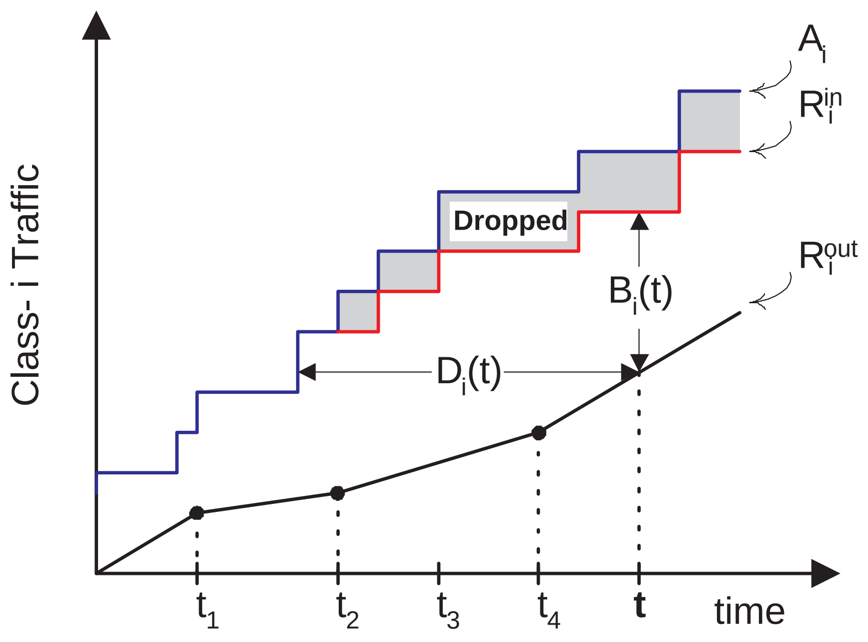


Figure 1: **Delay and backlog.**

We find it convenient to view the operation of JoBS in terms of an optimization problem. The constraints of the optimization problem are relative or absolute bounds on the loss and delay as given in the example above (*QoS constraints*) and constraints on the link and buffer capacity (*system constraints*). The objective function of the optimization is such that the amount of dropped traffic and changes to the current service rate allocation are minimized. The first objective prevents traffic from being dropped unnecessarily, and the second objective tries to avoid frequent fluctuations of the service rate allocation. Clearly, the solution of the optimization problem yields a service rate allocation of classes and determines how much traffic must be dropped. The optimization is performed for each arrival to the link.

The computational complexity of JoBS is determined by the number and the type of constraints and by the frequency of running the above optimization. To explore the principal properties of JoBS, we will, for now, assume that infinite computing resources are available. In a later section, we will approximate JoBS with a heuristic which incurs less computational overhead.

3.2 Formal Description of JoBS

Next we describe the basic operations of the algorithms for service rate adjustment and dropping algorithms at a JoBS 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. Let $a_i(t)$ and $\ell_i(t)$ denote the traffic arrivals and amount of dropped ('lost') traffic from class i at time t.

Let $r_i(t)$ denote the service rate allocated to class i at time t. We assume that $r_i(t)$ is nonzero only if there is a backlog of class-i traffic in the buffer, and we assume that scheduling in JoBS is workconserving, that is, $\sum_i r_i(t) = C$, if the backlog at time t is nonzero.

Remark: Throughout this paper, we take a fluid-flow interpretation of traffic, 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. On the other hand, packet-level scheduling algorithms that closely approximate fluid-flow schedulers with rate-guarantees are readily available [26].

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

$$A_i(t) = \int_0^t a_i(x)dx \ , \tag{1}$$

$$R_i^{in}(t) = A_i(t) - \int_0^t \ell_i(x)dx \ . \tag{2}$$

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

$$R_i^{out}(t) = \int_0^t r_i(x)dx \ . \tag{3}$$

We refer to Figure 1 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 for time t. The delay D_i at time t is the delay of an arrival which is transmitted at time t. Backlog and delay at time t are defined as

$$B_i(t) = R_i^{in}(t) - R_i^{out}(t) \ , \tag{4}$$

$$D_i(t) = \max_{x<t}\{x \mid R_i^{out}(t) \geq R_i^{in}(t-x)\} \ . \tag{5}$$

Upon a traffic arrival, say at time s, JoBS sets new service rates $r_i(s)$ and the amount of traffic to be dropped $\ell_i(s)$ for all classes, such that all QoS and system constraints can be met at times $t > s$. To determine the rates, JoBS projects the delays of all queued traffic. For the projections, JoBS assumes that the current state of the link will not change after time s. Specifically, JoBS makes the following assumptions on the service, the arrival, and the drops (we indicate projected values by a “tilde”) for times $t > s$:

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

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

3. There are no further packet drops: $\tilde{\ell}_i(t) = 0$.

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:

$$\tilde{R}_{i,s}^{in}(t) = R_i^{in}(s) \ , \tag{6}$$

$$\tilde{R}_{i,s}^{out}(t) = R_i^{out}(s) + (t-s)r_i(s) \ , \tag{7}$$

$$\tilde{B}_{i,s}(t) = \tilde{R}_{i,s}^{in}(t) - \tilde{R}_{i,s}^{out}(t) \ . \tag{8}$$

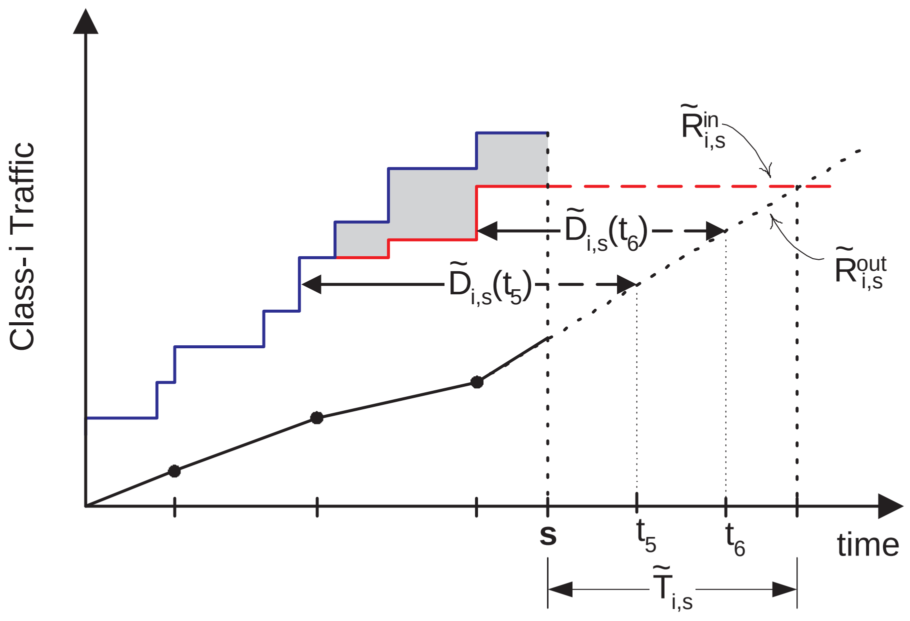


Figure 2: **Projected input curve, projected output curve, and projected delays.** The projection is performed at time s for the time interval $[s, s + \tilde{T}_{i,s}]$.

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.,

$$\tilde{T}_{i,s} = \min_{x>0}\{x \mid \tilde{B}_{i,s}(s+x) = 0\} \ . \tag{9}$$

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

$$\tilde{D}_{i,s}(t) = \max_{t-s<x<t}\{x \mid \tilde{R}^{out}_{i,s}(t) \geq R^{in}_i(t-x)\} \ . \tag{10}$$

If there are no arrivals after time s, the delay projections are correct.

In Figure 2, 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.

3.3 Drop-Tail vs. Drop-from-Front

JoBS only specifies the amount of traffic which should be dropped from a particular class, but JoBS does not select the position in the queue from which to drop traffic. For example, drops may occur from the head of the queue (Drop-from-Front, [18]) or from the tail (Drop-Tail). Note that such a policy has an impact on the shape of the input curve. Drop-from-Front drops traffic that was admitted in the past, and has therefore an influence on *past* values of R^{in}_i. More formally, without the assumption that there will not be any future drops, and a Drop-from-Front policy, Eqn. (10) does not hold anymore and shall be replaced by

$$\tilde{D}_{i,s}(t) = \max_{t-s<x<t} \{x \mid \tilde{R}_{i,s}^{out}(t) \geq R_i^{in}(t-x) - \int_s^t \ell(\tau)d\tau\} \ . \tag{11}$$

In the case of a Drop-Tail policy, Eqn. (10) holds even if some drops are performed in the future. In the case of a Drop-from-Front policy, Eqn. (11) imposes knowledge of future drops in order to be able to make a meaningful projection. Since it is practically difficult to evaluate how much traffic will be dropped in the future, we will consider a Drop-Tail policy in the present report.

3.4 Per-Class Delay and Loss Metrics

When defining relative QoS guarantees as in Subsection 3.1, e.g., Class-2 Delay $\approx 2 \cdot$ Class-1 Delay or Class-2 Loss Rate $\approx 10^{-1} \cdot$ Class-3 Loss Rate, we have assumed that a single metric is available to specify the 'delay' or the 'loss' of a class. In general, since there are several packets backlogged from a class, each likely to have a different delay, the notion of 'delay of class i' needs to be further specified. Likewise, the notion of 'loss rate of class i' requires further clarification.

3.4.1 Delay Metrics

Beginning with the delay metric $D_i(s)$ from Eqn. (5), we provide the rationale for our choices of per-class delay metrics.

Instantaneous Delay. The measure $D_i(s)$, given by Eqn. (5) describes the delay of the class-i packet that is in transmission at time s. $D_i(s)$ is a good measure for the delay of class-i traffic, only if $D_i(s)$ is roughly constant.

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

$$D_{i,s}^{avg}(\tau) = \frac{1}{\tau} \int_{s-\tau}^{s} D_i(x)dx \ . \tag{12}$$

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

$$D_{i,s}^{ewma} = D_{i,s-\tau}^{ewma} + w \cdot (D_i(s) - D_{i,s-\tau}^{ewma}) \ , \tag{13}$$

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

Average delay metrics as defined above only take into consideration the history of delays. Since the recent history of delays may not be a good indicator for the delays to be experienced by currently backlogged traffic, using Eqs. (12) and (13) may lead to poor predictions of delay guarantee violations. Note that Eqs. (12) and (13) may be appropriate metrics if the service rate allocation is formulated in terms of a closed-loop control problem, i.e., if the service rate allocation to classes is regarded as taking corrective actions to an 'error' in the current rate allocation.

Different from the above, the per-class delay metrics used in this paper attempt to measure the delay for the currently backlogged traffic. Our per-class delay metrics take advantage of the notion of predicted delay $\tilde{D}_{i,s}(t)$ as defined in Eqn. (10). Under the assumption that there are no arrivals and no losses after time s, and using the service rate allocation from time s, the predicted delay $\tilde{D}_{i,s}(t)$ provides the delay of the packet in transmission at time t. We define two delay metrics for the backlog from class i at time s, one for the worst-case delay and one for the 'typical' delay.

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

$$\tilde{D}_{i,s}^{max} = \max_{s<t<s+\tilde{T}_{i,s}} \tilde{D}_{i,s}(t) \ . \tag{14}$$

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

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

$$\overline{D}_{i,s} = \frac{1}{\tilde{T}_{i,s}} \int_{s}^{s+\tilde{T}_{i,s}} \tilde{D}_{i,s}(x)dx \ . \tag{15}$$

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

3.4.2 Loss Metrics

Similar to delays, there are several sensible choices for defining 'loss'. We present here two possible choices. We will use the first metric in the present paper. The second metric could be used in order to express algorithms such as RED [12] or RIO [6] within our framework.

Current Loss Ratio with Finite Memory. For this paper, we select one specific 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.[1] So, $p_{i,s}$ expresses the fraction of traffic that has been dropped in the time interval $[t_0, s]$, that is, [2]

$$\begin{aligned} p_{i,s} &= \frac{\int_{t_0}^{s} \ell_i(x)dx}{\int_{t_0}^{s} a_i(x)dx} && (16)\\ &= 1 - \frac{R_i^{in}(s-t_0, s^-) + a_i(s) - \ell_i(s)}{A_i(s-t_0, s)} \ . && (17)\end{aligned}$$

[1]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 $\max_{y<x}\{\sum_i B_i(y) = 0\}$.

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

Drop Probability. As detailed in Section 2, backlog controllers inspired by or derived from RED [12] use a probabilistic factor for determining whether or not an arrival shall be dropped. Assuming that each class of traffic has a time-dependent probabilistic factor $P_i(s)$ characterizing the probability of dropping the next arrival after time s, one could think of using the history of the values of $P_i(.)$ as a loss metric. For instance,

$$p_{i,s} = \frac{1}{s - t_0} \int_{t_0}^{s} P_i(x) dx \ , \tag{18}$$

is the average drop probability over the current busy period. Similarly, one could define the worst-case drop probability since the beginning of the current busy period as

$$p_{i,s} = \max_{t_0 < x < s} P_i(x) \ . \tag{19}$$

In the present paper, we will only use Eqn. (17) as our definition for the 'loss rate of class i'.

4 Service Rate Adaptation and Drop Algorithm in JoBS

In this section we discuss how JoBS adjusts the service rates of classes and decides if and how much traffic to drop. The algorithm will be expressed in terms of an optimization problem.

Each time s, when an arrival occurs, a new optimization is performed. The optimization variable is a time-dependent vector $\mathbf{x}_s$ which contains the service rates $r_i(s)$ and the amount of traffic to be dropped $\ell_i(s)$,

$$\mathbf{x}_s = (r_1(s) \ldots r_Q(s) \ \ell_1(s) \ldots \ell_Q(s))^T \ . \tag{20}$$

The optimization problem has the form

$$\begin{array}{ll} \textbf{Minimize} & F(\mathbf{x}_s) \\ \textbf{Subject to} & g_j(\mathbf{x}_s) = 0, \quad j = 1, \ldots, M \\ & h_j(\mathbf{x}_s) \geq 0, \quad j = M+1, \ldots, N. \end{array} \tag{21}$$

where $F(.)$ is an objective function, and the g_j's and h_j's are constraints.

The objective function of JoBS will be stated such that JoBS minimizes the amount of dropped traffic, and keeps the changes to the current service rate allocation small. The constraints are QoS constraints and system constraints. The optimization at time s is done with knowledge of the system state before times 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 discuss the constraints and the optimization function. The optimization can be used as a reference system that provides a benchmark, against which practical scheduling and dropping algorithms can be compared.

4.1 System and QoS Constraints

Next, we discuss the constraints in our system. There are two types of constraints: *system constraints* describe constraints and properties of the output link, and *QoS constraints* define the desired service differentiation.

4.1.1 System Constraints

The system constraints specify physical limitations and properties at the output link.

- *Buffer size*: The total backlog cannot exceed the buffer size B, that is, $\sum_i B_i(t) \leq B$ for all times t.
- *Workconserving property:* At a workconserving link $\sum_i r_i(t) = C$, holds for all times t where $\sum_i B_i(t) > 0$. This constraint is stronger than the limit given by the link capacity C, i.e., $\sum_i r_i(t) \leq C$.
- *Other bounds:* Rates and packet drops 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 $0 \leq \ell_i(t) \leq B_i(t)$ for all times t.

4.1.2 QoS Constraints

JoBS allows two types of QoS constraints, relative constraints and absolute constraints. QoS constraints can be expressed for delays and for loss rates. One could also think of providing constraints on the rates assigned to each class of traffic. Since JoBS directly controls the variable $\mathbf{x}_s$, and thus the rates $r_i(s)$, JoBS could handle such rate guarantees if desired, but we will not address this issue in the present paper.

The number and type of QoS constraints is not limited by JoBS. However, absolute QoS constraints may result in an infeasible system of constraints. In such a situation, one or more constraints must be relaxed or eliminated. 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. QoS constraints for classes which are not backlogged are simply ignored.

- *Absolute Delay Constraints (ADC):* These constraints enforce that the projected delays of class i satisfy a worst-case bound d_i. We obtain

$$\tilde{D}_{i,s}^{max} \leq d_i \ , \tag{22}$$

where $\tilde{D}_{i,s}^{max}$ is the worst-case metric defined in Eqn. (14). If this condition holds for all s, the delay bound d_i is never violated.

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

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

We are interested in meeting this constraint for most of the packets present in the queue at time s, thus, we will use the average metric defined in Eqn. (15) for characterizing the 'delay of class i'.

To provide some flexibility in the scheduling decision, we do not enforce relative delay constraints strictly, but allow for some slack. So, the relative delay constraints are of the form

$$k_i(1-\varepsilon) \leq \frac{\overline{D}_{i+1,s}}{\overline{D}_{i,s}} \leq k_i(1+\varepsilon) \ , \tag{23}$$

where $k_i > 1$ is the target differentiation factor and ε ($0 \le \varepsilon \le 1$) indicates a tolerance level, and $\overline{D}_{i,s}$ is as defined in Eqn. (15). If relative constraints are not specified for some classes, the constraints are adjusted accordingly.

Next we discuss constraints on the loss rate. Being interested in the current loss ratio, we will use the metric defined in Eqn. (17). Note that, in Eqn. (17), all values except $\ell_i(s)$ are known at time s.

- *Absolute Loss Constraints (ALC):* An ALC specifies that the loss ratio of class i, as defined above, never exceeds a limit L_i, that is,

$$p_{i,s} \le L_i \ , \tag{24}$$

  where $p_{i,s}$ is as defined in Eqn. (17).

- *Relative Loss Constraints (RLC):* The RLCs 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

$$k'_i(1-\varepsilon') \le \frac{p_{i+1,s}}{p_{i,s}} \le k'_i(1+\varepsilon') \ , \tag{25}$$

  where $k'_i > 1$ is the target differentiation factor, and ε'_i ($0 \le \varepsilon' \le 1$) indicates a level of tolerance.

The constraints defined by Eqs. (22), (23), (24), and (25) 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 $r_i(t)$ and the packet drops $\ell_i(t)$. Therefore, we need to express the constraints defined by Eqs. (22), (23), (24), and (25) as functions of the service rates and the packet drops. We present this derivation in Appendix A.

4.2 Objective Function

Provided that the QoS and system constraints can be satisfied, the objective function of JoBS selects 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:

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

The first objective ensures that traffic is dropped only if there is no alternative 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.

The formulation of the objective function expresses the above objectives in terms of a cost function.

$$F(\mathbf{x}_s) = \sum_{i=1}^{Q}(r_i(s) - r_i(s^-))^2 + C^2 \sum_{i=1}^{Q} \ell_i(s) \ , \tag{26}$$

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. (26) we need to use the quadratic form $(r_i(s) - r_i(s^-))^2$, since $\sum_i (r_i(s) - r_i(s^-)) = 0$ for a workconserving link with a backlog at time s. Given that $(r_i(s) - r_i(s^-))^2 \leq C^2$, the scaling factor C^2 in front of the second sum of Eqn. (26) ensures that traffic drops are the dominating term in the objective function.

This concludes the description of the optimization problem in JoBS. The discussion in Appendix A will show that this is a *non-linear optimization problem*, which can be solved with available numerical algorithms [29]. It is worth noting that the only non-linear constraints are the RDCs. Therefore, by linearizing the RDCs using a first-order approximation, it is feasible to reduce the computational overhead of the optimization. Such a linearization of the RDCs will be exploited in Section 5, where we provide a heuristic algorithm which emulates the reference model with a low computational overhead.

5 Heuristic Approximation of JoBS

We next present a heuristic that approximates the JoBS algorithm, yet, which has significantly lower computational complexity. Our goal is not to present an algorithm that is readily implementable and can operate at line speeds. Instead, we want to demonstrate that it is feasible to find relatively simple algorithms that can closely approximate the idealized JoBS system. The presented heuristic should be regarded as a first step towards a router implementation. We first describe how we translate the reference fluid-flow model used by the optimization into a packet-level architecture, then we detail how rate allocation and packet drops are performed in this heuristic.

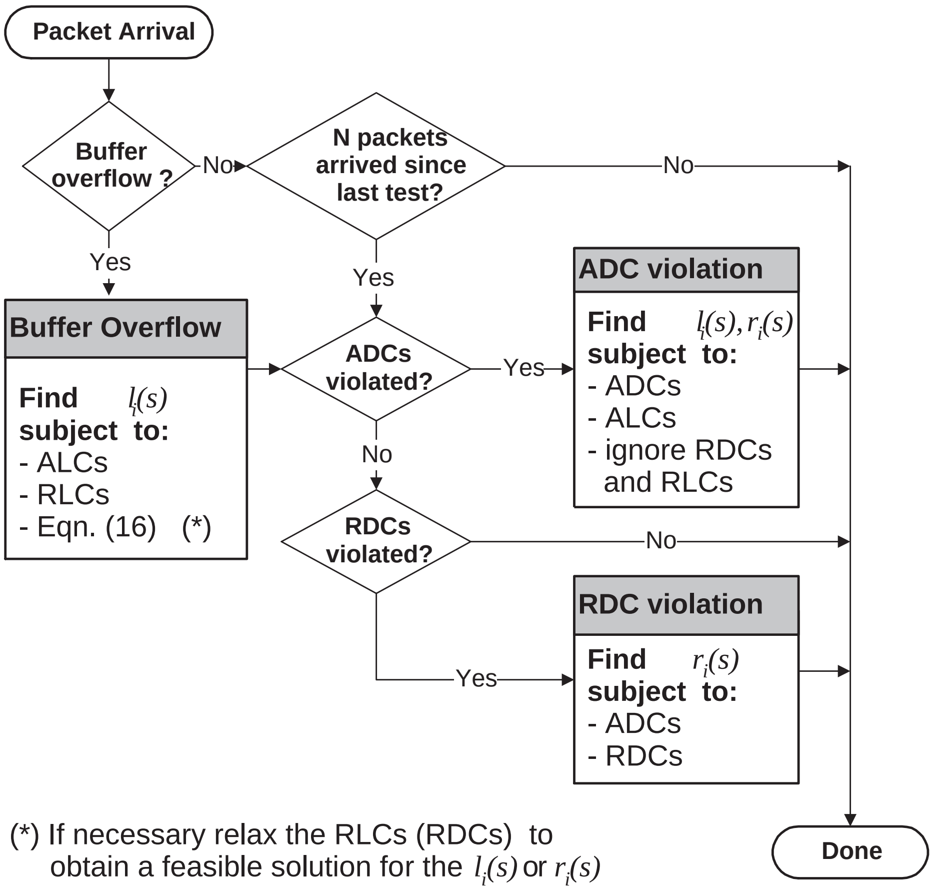
5.1 The Packetized Model

The translation of the fluid-flow service model of JoBS into a packet-level architecture is done using well-known techniques of assigning virtual deadlines to packets which are computed from a rate allocation [26, 32].

The algorithm used in JoBS (heuristic) is similar to Virtual Clock [32]. Each class of traffic has its own virtual clock VC_i, used to determine the virtual deadline VD_i of a packet from the same class. After the beginning of the current busy period, when the first packet from class i enters the system, VC_i is set equal to the current time. From then on, every time t a class-i packet of size z arrives at the node, the virtual clock and the virtual deadline for class i are updated as follows:

$$\begin{array}{lrcl} 1. & \text{VD}_i & \leftarrow & \max\{t, \text{VC}_i\}\ , \\ 2. & \text{VC}_i & \leftarrow & \text{VC}_i + \frac{z}{r_i(t)}\ , \\ 3. & \text{VD}_i & \leftarrow & \text{VD}_i + \frac{z}{r_i(t)}\ . \end{array}$$

Each class-i packet entering the system at time t is then stamped with the virtual deadline VD_i. The packets are transmitted in the order of increasing virtual deadlines. A change to the service rate of a class may require to update the virtual deadlines of already queued packets. Finally, at fixed time intervals, also called checkpoints, the virtual clocks VC_i are reset to the current time t if $\text{VC}_i < t$.



Packet Arrival
Buffer overflow ?
No
N packets arrived since last test?
No
Yes
Yes
ADC violation
Find $l_i(s), r_i(s)$ subject to:
- ADCs
- ALCs
- ignore RDCs and RLCs
Buffer Overflow
Find $l_i(s)$ subject to:
- ALCs
- RLCs
- Eqn. (16) (*)
ADCs violated?
Yes
No
RDCs violated?
No
RDC violation
Find $r_i(s)$ subject to:
- ADCs
- RDCs
Yes
(*) If necessary relax the RLCs (RDCs) to obtain a feasible solution for the $l_i(s)$ or $r_i(s)$
Done

Figure 3: **Outline of the Heuristic algorithm**.

5.2 Rate Allocation and Packet Drops

Our heuristic algorithm completely avoids running the optimization from Section 4. Instead, the heuristic maintains the current rate allocation until a buffer overflow occurs or a delay violation is predicted. At that time, the heuristic picks a new feasible rate allocation. Unless there is a buffer overflow, the tests for violations of ADCs and RDCs[3] 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 3. The algorithm is broken up into a number of smaller computations. We first separate the particular case when there is a buffer overflow.

[3]Recall: *ADC* = absolute delay constraint, *RDC* = relative delay constraint, *ALC* = absolute loss constraint, and *RLC* = relative loss constraint.

5.2.1 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

$$\sum_i \ell_i(s) = \text{Size of arriving packet} . \tag{27}$$

The heuristic picks a solution for the $\ell_i(s)$ which satisfies Eqn. (27) and the RLCs in Eqn. (25), where ε' is set to zero to simplify the search for a solution. If the solution violates an ALC, the RLCs are relaxed until all ALCs are satisfied. Once the $\ell_i(s)$'s are determined the algorithm continues with a test for delay constraint violations, as shown in Figure 3.

In Appendix B we discuss in detail how a solution is obtained. In the appendix, we also show that the complexity of finding the solution is $O(Q^2)$, where Q is the number of classes. If the number of classes is small (e.g., $Q = 4$), the process is reasonably fast.

5.2.2 Meeting the Delay Constraints

If there are no buffer overflows, the algorithm makes delay projections to test for delay violations (ADCs and RDCs). The test for delay violations is not performed for every arrival, but only once for every N packet arrivals. [4]

The tests use the current service rate allocation to predict future violations. The heuristic performs two tests. The first test checks whether the ADC will be met. In order to make this test, the heuristic starts with a conservative estimate of the worst-case delay given by Eqn. (14) for the class-i backlog at time s. For this, the heuristic uses the following bound, which is easily verified by referring to Figures 1 and 2.

$$\tilde{D}_{i,s}^{max} \leq D_i(s) + \frac{B_i(s)}{r_i(s)} . \tag{28}$$

This conservative bound is then used as a worst-case delay estimate in Eqn. (22) for testing potential ADC violations.

The second test checks if the RDC will be met. This test is only performed if the first test was successful, i.e., no ADC violation was predicted. Instead of using the definition of the average projected delay given by Eqn. (15), the heuristic uses the following estimate of the average delay:

$$\overline{D}_{i,s} \approx \frac{D_i(s) + \max_{x\geq 0}\{x \mid R_i^{in}(s) = R_i^{in}(s-x)\}}{2} + \frac{B_i(s)}{2r_i(s)} . \tag{29}$$

The first term on the righthand side of Eqn. (29) is an estimate for the average time already spent in the queue, averaged over all currently backlogged class-i packets. The second term is an estimate of the average remaining waiting time, averaged over all currently backlogged class-i packets as well. Different from Eqn. (15) which considered the delays of all backlogged class-i packets, the estimate in Eqn. (29) only requires to compute the delay of two packets: the oldest class-i packet still present in the queue, and the last class-i packet to have entered the queue. The estimate in Eqn. (29) is used in Eqn. (23) for testing potential RDC violations.

[4]The parameter N should be properly tuned. Increasing N increases the risk of violating some constraints. Decreasing N increases the overall computational overhead.

Depending on the outcome of the two tests described above (ADC violation and RDC violation), the heuristic distinguishes the following three cases.

1. No violations are predicted. In this case, the service rate allocation remains unchanged.

2. RDC violation predicted. If the violation of some RDC (but no ADC) is predicted, the heuristic algorithm determines new rate values.

   Here, the RDCs as defined in Eqn. (23) are first transformed into equations by setting $\varepsilon = 0$. The heuristic uses then a set of estimates in order to make the RLC linear. Together with the workconserving property, $\sum_i r_i(s) = C$, 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. We give a detailed expression of this system of equations in Appendix B.

3. ADC violation predicted. 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 $\ell_i(s)$'s. To simplify the task, our heuristic simply ignores all relative QoS constraints (RLCs, RDCs) when an ADC violation occurs, and only tries to satisfy ALCs and ADCs.

   The heuristic starts with the conservative estimate of the worst-case delay given by Eqn. (28) for the class-i backlog at time s. Then, using $B_i(s) = B_i(s^-) + a_i(s) - \ell_i(s)$, the following is a sufficient condition for satisfying the ADC of class i with delay bound d_i at time s.

   $$\underbrace{\frac{1}{r_i(s)} \frac{B_i(s^-) + a_i(s) - \ell_i(s)}{d_i - D_i(s)}}_{\rho_i} \leq 1 \ . \qquad (30)$$

   The heuristic algorithm will select the $r_i(s)$ and $\ell_i(s)$ such that Eqn. (30) is satisfied for all i. Initially, rates and traffic drops are set to $r_i(s) = r_i(s^-)$ and $\ell_i(s) = 0$. Since at least one ADC is violated, there is at least one class with $\rho_i > 1$, where ρ_i is defined in Eqn.(30).

   Now, we apply a greedy method which tries to redistribute the rate allocations until $\rho_i \leq 1$ for all classes. This is done by reducing $r_i(s)$ for classes with $\rho_i < 1$, and increasing $r_i(s)$ for classes with $\rho_i > 1$. The algorithm starts by reducing $r_i(s)$ in the lowest priority class that has $\rho_i < 1$ and increases accordingly $r_i(s)$ in the highest priority class with $\rho_i > 1$. The process is then iterated until the redistribution of rates is complete. The computational complexity of this operation is $O(Q)$.

   If, after the redistribution of rates, there are still classes i with $\rho_i > 1$, the $\ell_i(s)$ are increased until $\rho_i < 1$ for those classes. The worst-case complexity of this operation is also $O(Q)$, yielding a worst-case complexity of $O(Q)$ as well for the entire algorithm. To minimize the number of dropped packets, $\ell_i(s)$ is never increased to a point where an ALC is violated.

6 Evaluation

We present an evaluation of the JoBS algorithm via simulation. Our goals are (1) to determine if and how well JoBS provides the desired service differentiation; (2) to determine how well the heuristic algorithm from Section 5 approximates the optimization of JoBS; and (3) to compare JoBS with existing proposals for proportional differentiated services.

In the simulations, we compare the performance of the following four schemes.

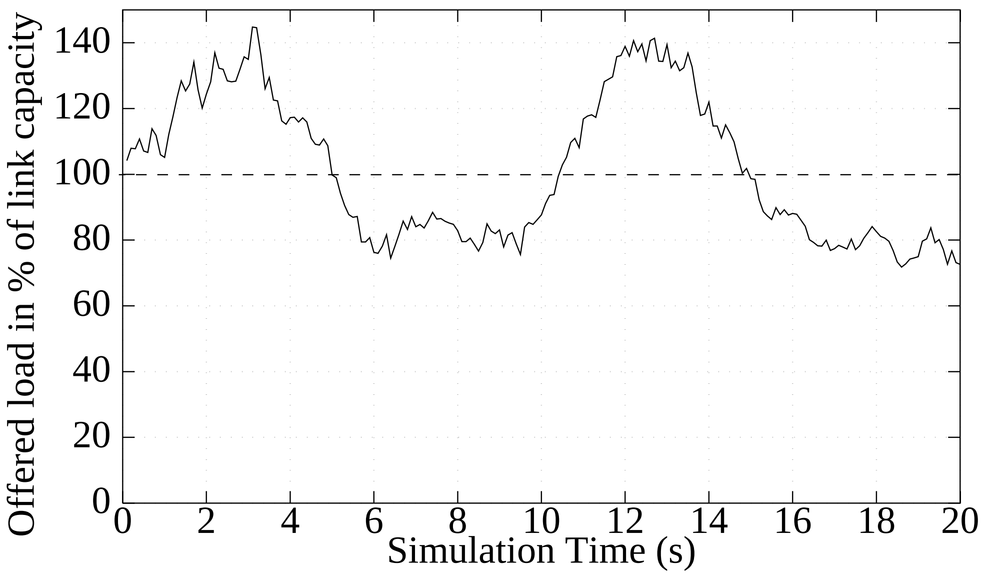


Figure 4: **Offered Load.**

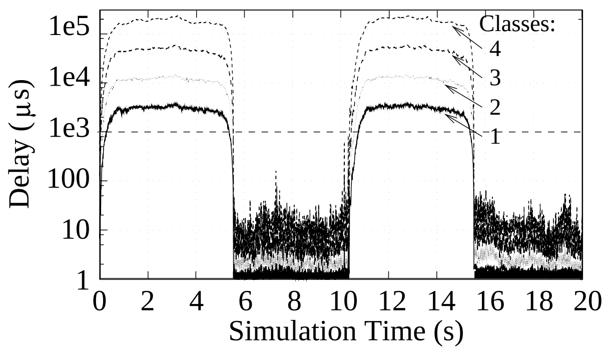
- **JoBS (optimization):** This is the optimization described in Section 4.
- **JoBS (heuristic):** This is the heuristic algorithm discussed in Section 5. Unless there is a buffer overflow, tests for delay violations are performed once for every $N = 100$ packet arrivals.
- **WTP/PLR(∞) [9]:** The dropping and scheduling algorithm WTP/PLR(N) and WTP/PLR(∞) from [9] 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[22]/Drop-Tail:** The MDP scheduler presented in [22] 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 two simulation experiments. In the first experiment, we compare and contrast the relative differentiation provided of JoBS (optimization), JoBS (heuristic), WTP/PLR(∞), and MDP/Drop-Tail without specifying absolute constraints. In the second experiment, we augment the set of constraints by absolute delay constraints on the highest priority class, and show that JoBS can effectively provide both relative and absolute differentiation.

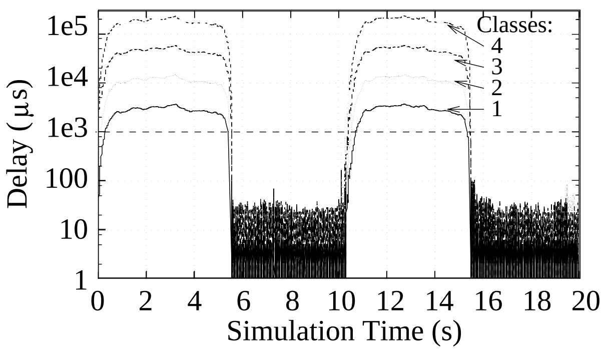
6.1 Experimental Setup

We consider a single output link with capacity $C = 1$ Gbps and a buffer size of 6.25 MByte. We have $Q = 4$ classes. We use the same load curve in all experiments. The length of each experiment is 20 seconds of simulated time, starting at time 0 with an empty system and an idle scheduler.

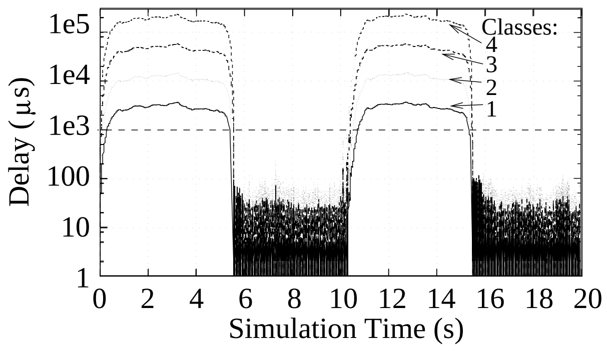
The incoming traffic is composed of a superposition of Pareto sources with a parameter $\alpha = 1.2$ and an average interarrival time of 300 μs. These sources generate packets with a fixed size of 125 Byte. As offered



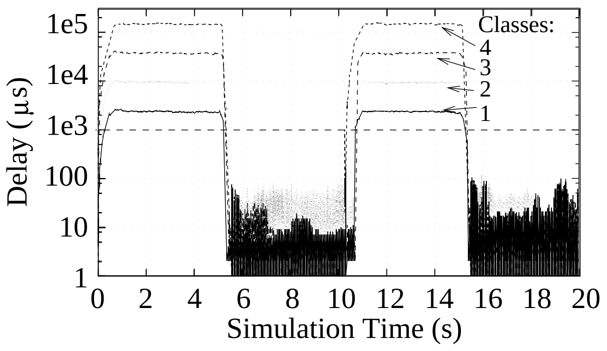
(a) JoBS (optimization) - Packet Delays



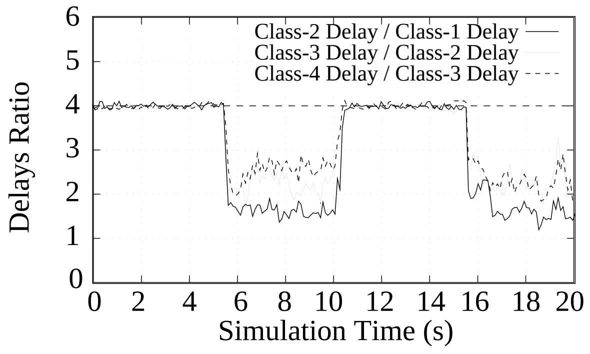
(b) JoBS (heuristic) - Packet Delays



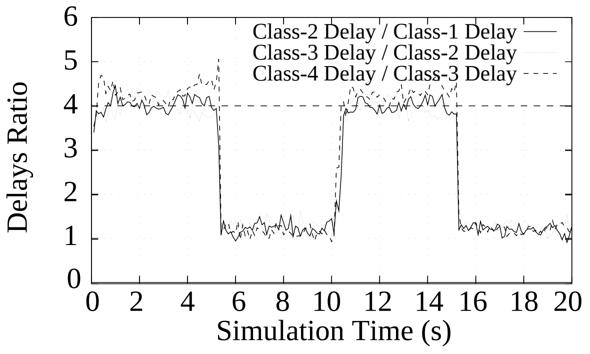
(c) WTP/PLR(∞) - Packet Delays



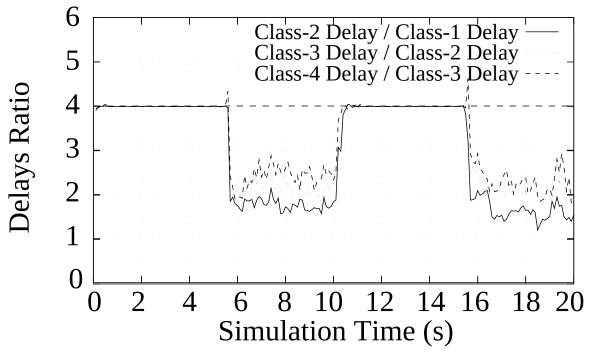
(d) MDP/Drop-Tail - Packet Delays



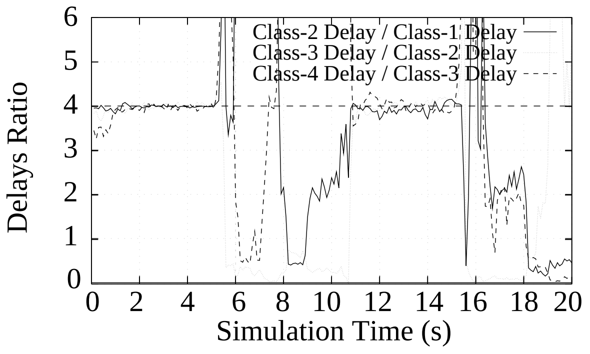
(e) JoBS (optimization) - Ratios of delays



(f) JoBS (heuristic) - Ratios of delays

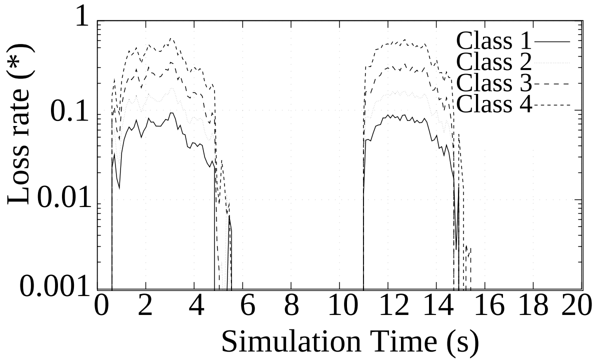


(g) WTP/PLR(∞) - Ratios of delays

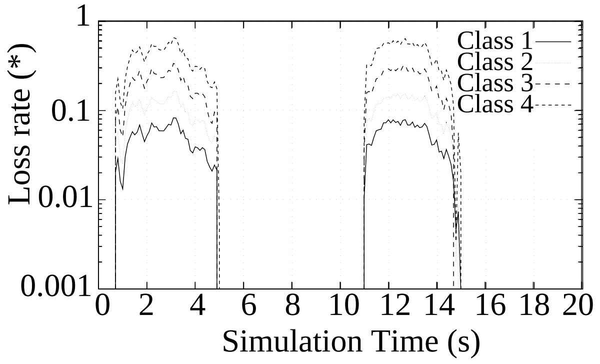


(h) MDP/Drop-Tail - Ratios of delays

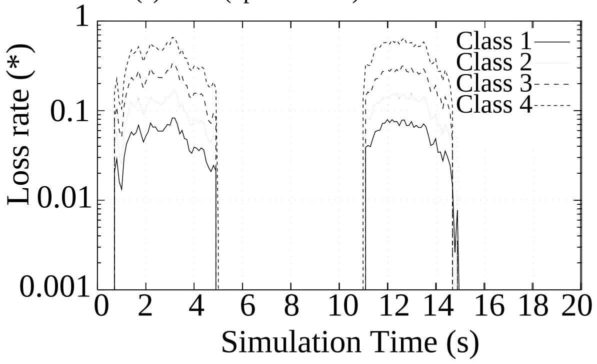
Figure 5: **Experiment 1: Delay Differentiation.** The graphs show the absolute delay values (a)-(d) and the ratios of the delays for successive classes (e)-(h). The target value for the ratios is $k = 4$.



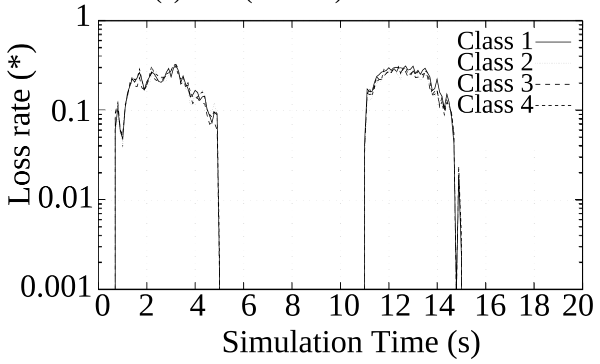
(a) JoBS (optimization) - Loss Rates



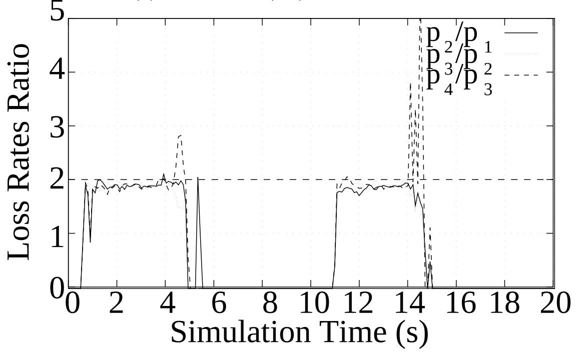
(b) JoBS (heuristic) - Loss Rates



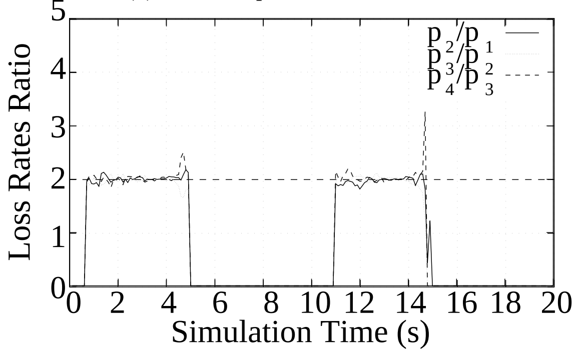
(c) WTP/PLR(∞) - Loss Rates



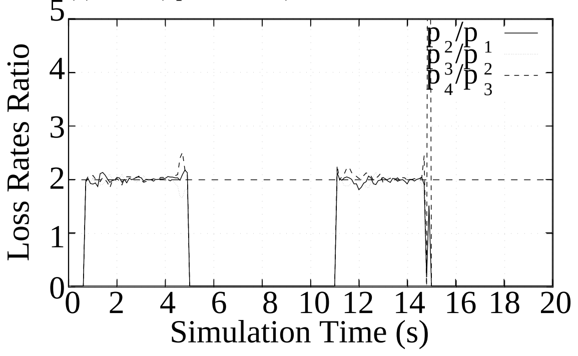
(d) MDP/Drop-Tail - Loss Rates



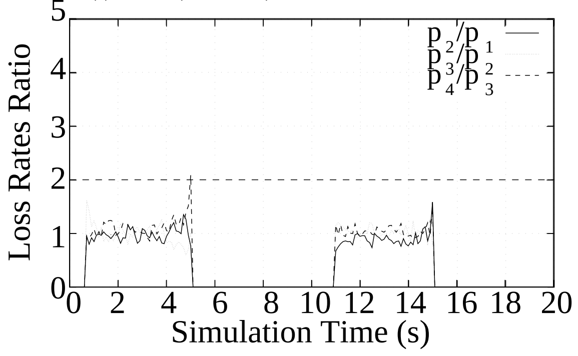
(e) JoBS (optimization) - Ratios of loss rates



(f) JoBS (heuristic) - Ratios of loss rates



(g) WTP/PLR(∞) - Ratios of loss rates



(h) MDP/Drop-Tail - Ratios of loss rates

Figure 6: **Experiment 1: Loss Differentiation.** The graphs show the loss rates (a)-(d) and the ratios of loss rates for successive classes (e)-(h). The target value for the ratios $k' = 2$. (*) The loss rate of class i is expressed as a fraction of the arrival rate of class i.

load, we generate a time-varying load curve, where the number of active sources follows a sinusoidal pattern with period $T = 10s$. The offered load used in our experiments is plotted in Figure 4. Between 200 and 550 sources are active at the same time, resulting in an offered load comprised between 75% and 145% of the link capacity. The load from the classes is symmetric, that is, at each time, each class generates 25% of the aggregate load.

6.2 Experiment 1: Relative Differentiation Only

The first experiment focuses on relative service differentiation, and does not include absolute constraints. The objective for the relative differentiation are

$$\text{Delay of Class } (i+1)/\text{Delay of Class } i \approx 4 \ ,$$
$$\text{Class-}(i+1) \text{ Loss Rate}/\text{Class-}i \text{ Loss Rate} \approx 2 \ .$$

Thus, for JoBS, the parameters in the RDCs and RLCs are set to $k_i = 4$ and $k'_i = 2$ for all i. The tolerance levels are set to $\varepsilon = 0.001$ and $\varepsilon' = 0.05$ in JoBS (optimization), and to $\varepsilon = 0.01$ and $\varepsilon' = 0.05$ in JoBS (heuristic). The results of the experiment are presented in Figures 5 and 6, where we graph the delays of packets on Figures 5(a)-(d), the ratios of delays of successive classes on Figures 5(e)-(h), the loss rates on Figures 6(a)-(d), and the ratios of loss rates of successive classes for JoBS (optimization), JoBS (heuristic), WTP/PLR(∞) and MDP/Drop-Tail on Figures 6(e)-(h). The plotted ratios of delays and ratios of loss rates present averages over moving time windows of size $0.1\ s$ [5], while the absolute delay values are plotted for each packet traversing the system.

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'. Note that 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 algorithm.

When the system load is low, that is, in time intervals $[6\ s, 10\ s]$ and $[16\ s, 20\ s]$, JoBS (heuristic) is not effective for providing a delay differentiation. Here, JoBS (optimization) and WTP/PLR(∞) still manage to achieve some delay differentiation, albeit far from the target values. MDP/Drop-Tail, plotted in Figure 5(h), provides some differentiation, but the system seems unstable, particularly after a transient change in the load. One should note, however, that 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 5(a)-(d). Figures 5(a)-(d) also show that the absolute values for the delays are comparable in all schemes.

In Figures 6(e) and 6(g), 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 on Figure 6(h).

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 6(a)-(c) prove that in the simulations, the loss rates of

[5]This measure is adopted from [9].

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 on Figure 6(d). This shows that, in this experiment, all schemes only drop packets when a buffer overflow occurs.

6.3 Experiment 2: Relative and Absolute Differentiation

In this experiment, we evaluate how well JoBS can satisfy a mix of absolute and relative delay constraints. In this experiment, we only present results for JoBS (optimization) and JoBS (heuristic). Note that WTP/PLR(∞) and MDP/Drop-Tail do not support both relative and absolute guarantees, and are therefore not included.

We consider the same simulation setup and the same relative constraints (RDCs and RLCs) as in Experiment 1, but add an absolute delay constraint (ADC) for Class 1 with a delay bound of

$$d_1 = 1,000\mu s \ .$$

We call this scenario "with ADC, all RDCs". Note that, with the given relative delay constraints from Experiment 1, the other classes have implicit absolute delay constraints, which are approximately[6] 4,000 μs for Class 2, 16,000 μs for Class 3, and 64,000 μs for Class 4.

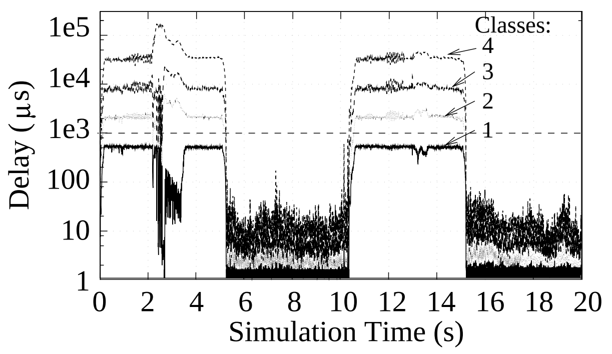
These 'implicit' absolute constraints can be avoided, by removing the RDC which governs the ratio of the delays between Class 2 and Class 1. The resulting constraint set, referred to as "with ADC, one RDC removed", is included in this experiment. For reference purposes, we also include the results for JoBS (heuristic) from Experiment 1. We refer to this constraint set as "no ADC, all RDCs". Figures 7(e-f) show that, without the ADC, the delays for Class 1 are as high as $5,000\mu s$.

In Figures 7, 8, 9 and 10 we respectively plot the absolute delays of all packets, the ratios of the delays for successive classes, the loss rates of each class, and the ratios of the loss rates for successive classes. The plots on Figures 8, 9, and 10 use once again averages over a sliding window of size $0.1s$.

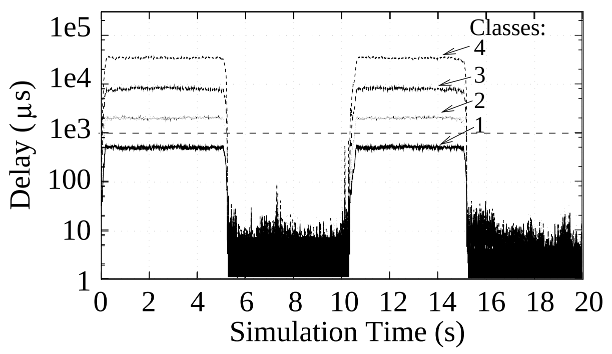
Based on this set of plots, one can make three observations. The first observation is that, as shown on Figures 7(a-d), the absolute delay constraints, either explicit (in the case of Class 1) or implicit (in the case of Classes 2, 3 and 4 in the scenario "With ADC, all RDCs"), are enforced. The second observation is that the optimization model seems to be less stable than the heuristic model in the experiment "With ADC, all RDCs". In Figure 7, one can notice some oscillations in the time intervals $[2s, 4s]$ and $[13s, 14s]$. These oscillations are due to the fact that the set of constraints is infeasible during the time intervals $[2s, 5s]$ and $[13s, 15s]$, and thus needs to be relaxed. In that respect, JoBS (optimization) and JoBS (heuristic) behave quite differently. As shown on Figures 9(a) and 10(a), JoBS (optimization) relaxes the relative loss constraints only when the system is infeasible. However, Figure 9(a) also shows that this relaxation is performed suddenly. Since one of the objectives of the optimizer is to minimize the changes applied to the system, despite this relaxation of the RLCs, the set of constraints might still be infeasible due to the current state of the system. This happens in the time intervals $[2s, 4s]$ and $[13s, 14s]$. In such a case, the relative delay constraints are also relaxed, as shown on Figure 8(a).

Conversely, JoBS (heuristic) does not take into account the relative loss constraints when resolving an ADC violation. During the times when the link is overloaded, ADC violations for all classes are predicted

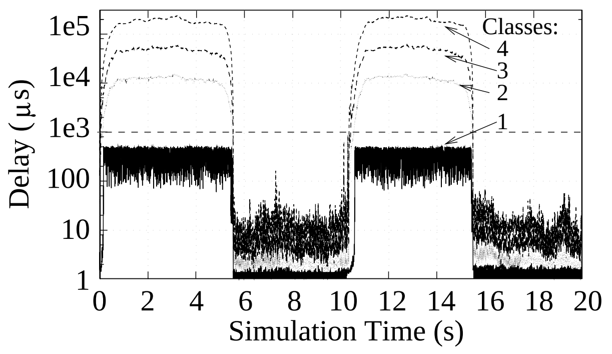
[6]Due to the tolerance value ε the exact values are not multiples of 1,000: the inferred ADC are respectively 4040 μs for Class 2, 16322 μs for Class 3, and 65940 μs for Class 4 when $\varepsilon = 0.01$, for instance.



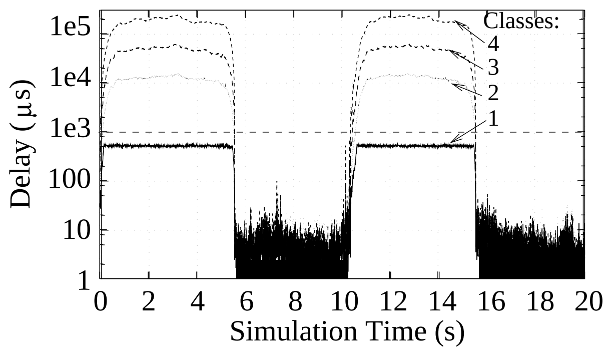
(a) With ADC, all RDCs - JoBS (optimization)



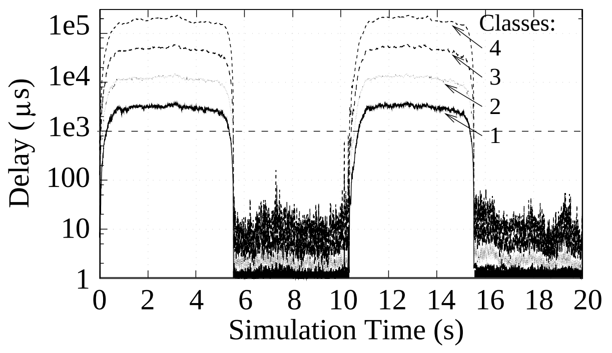
(b) With ADC, all RDCs - JoBS (heuristic)



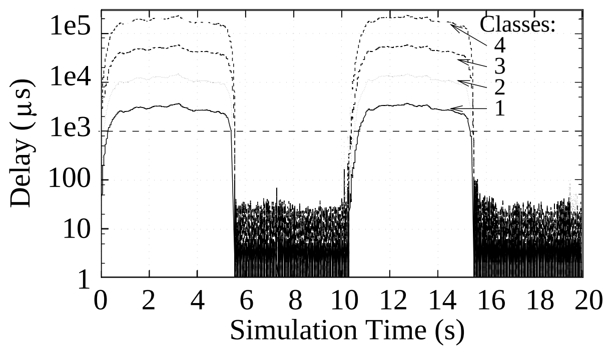
(c) With ADC, one RDC removed - JoBS (optimization)



(d) With ADC, one RDC removed - JoBS (heuristic)

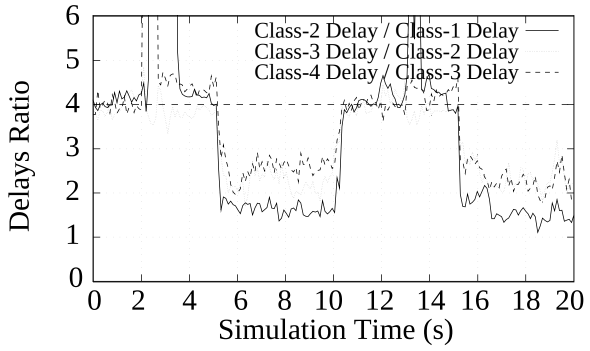


(e) No ADC, all RDCs - JoBS (optimization)

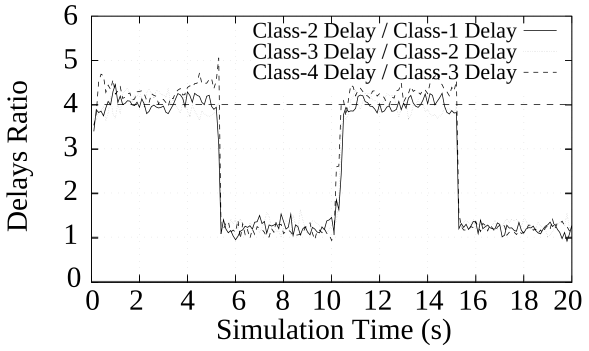


(f) No ADC, all RDCs - JoBS (heuristic)

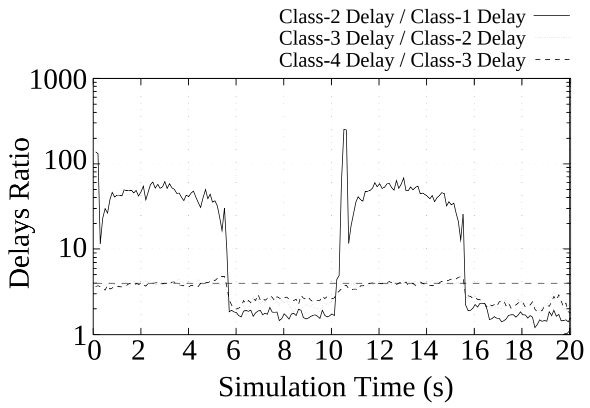
Figure 7: **Experiment 2: Delay Differentiation.** The graphs show the absolute delay values.



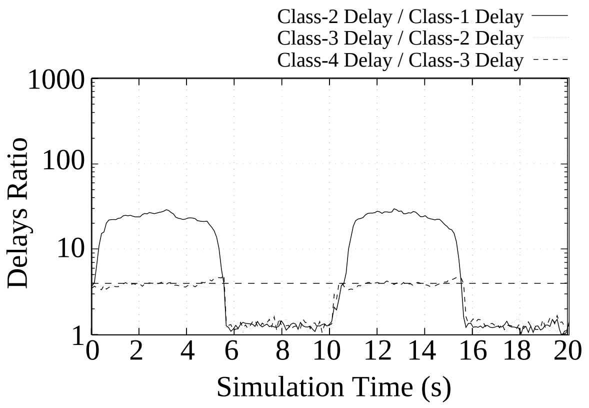
(a) With ADC, all RDCs - JoBS (optimization)



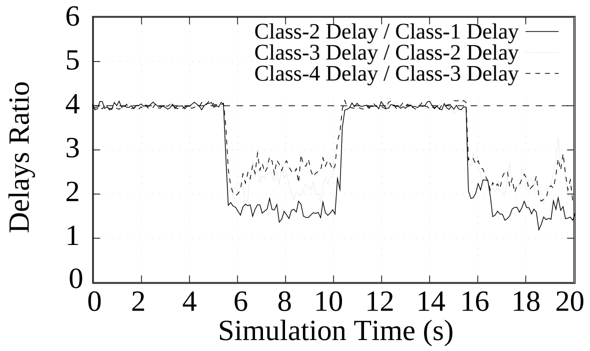
(b) With ADC, all RDCs - JoBS (heuristic)



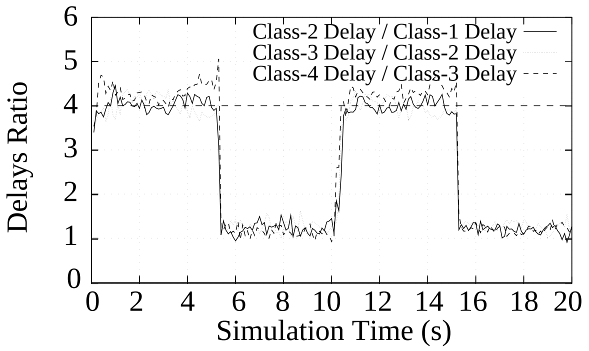
(c) With ADC, one RDC removed - JoBS (optimization)



(d) With ADC, one RDC removed - JoBS (heuristic)

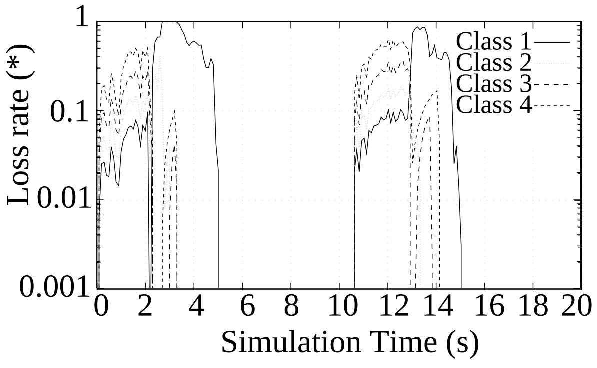


(e) No ADC, all RDCs - JoBS (optimization)

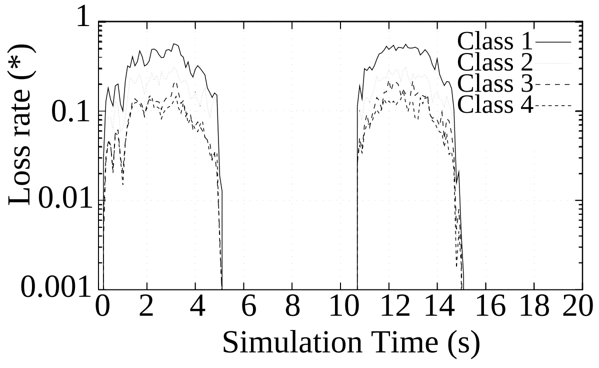


(f) No ADC, all RDCs - JoBS (heuristic)

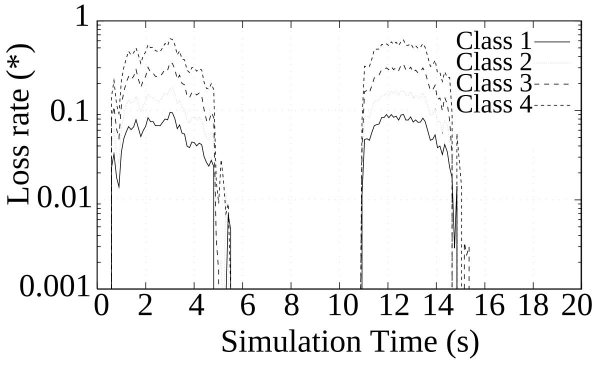
Figure 8: **Experiment 2: Delay Differentiation.** The graphs show the ratios of delays for successive classes. The target value is $k = 4$.



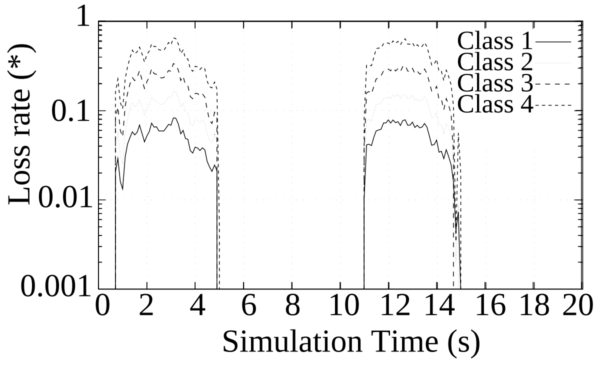
(a) With ADC, all RDCs - JoBS (optimization)



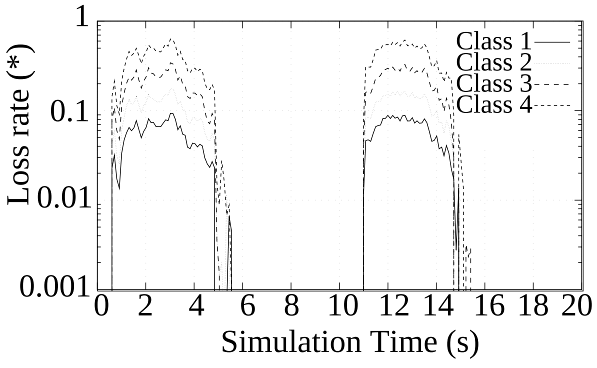
(b) With ADC, all RDCs - JoBS (heuristic)



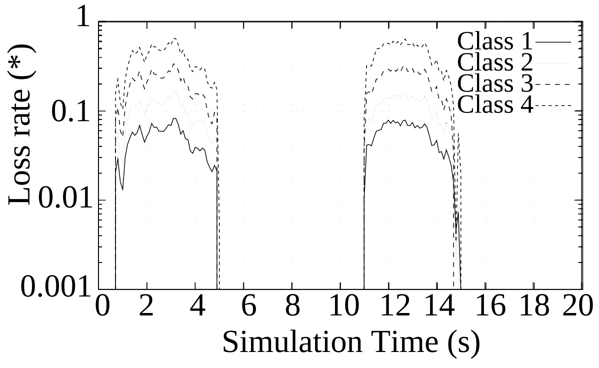
(c) With ADC, one RDC removed - JoBS (optimization)



(d) With ADC, one RDC removed - JoBS (heuristic)

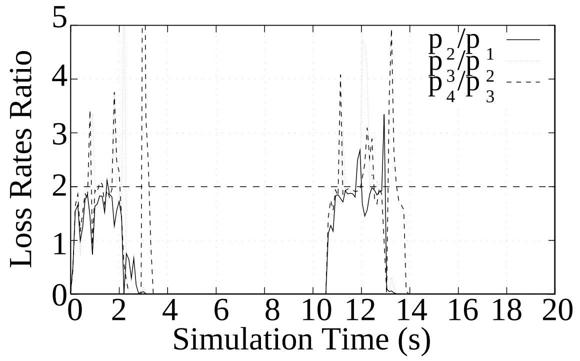


(e) No ADC, all RDCs - JoBS (optimization)

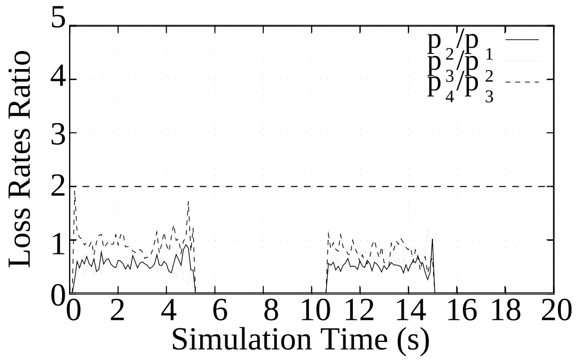


(f) No ADC, all RDCs - JoBS (heuristic)

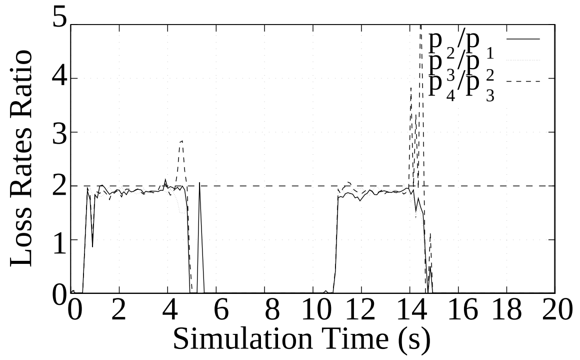
Figure 9: **Experiment 2: Loss Differentiation.** The graphs show the loss rates of all classes. (*) The loss rate of class i is expressed as a fraction of the arrival rate of class i.



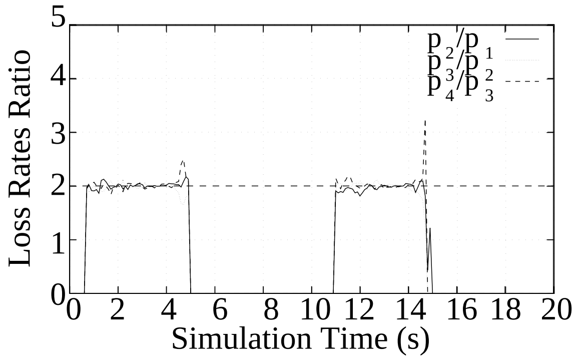
(a) With ADC, all RDCs - JoBS (optimization)



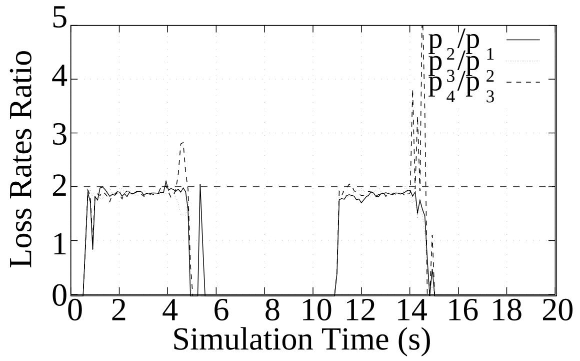
(b) With ADC, all RDCs - JoBS (heuristic)



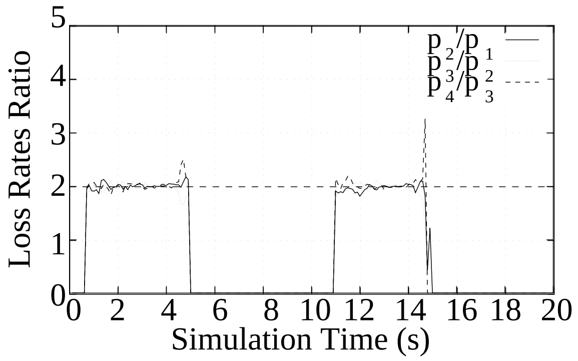
(c) With ADC, one RDC removed - JoBS (optimization)



(d) With ADC, one RDC removed - JoBS (heuristic)



(e) No ADC, all RDCs - JoBS (optimization)



(f) No ADC, all RDCs - JoBS (heuristic)

Figure 10: **Experiment 2: Loss Differentiation.** The graphs show the ratios of loss rates for successive classes. The target value is $k' = 2$.

Experiment	Time to complete (FIFO/Drop-Tail)	Time to complete (optimization)	Time to complete (heuristic)
With ADC, all RDCs	13 min	1428 min	34 min
With ADC, one RDC removed	13 min	8039 min	26 min
No ADC, all RDCs	13 min	7983 min	36 min

Table 1: **Comparison of the computational overhead of JoBS (optimization) and JoBS (heuristic).**

each time the system is checked. Since this test is performed as soon as a buffer overflow occurs (see Section 5), ADC violations are predicted upon each arrival in such cases, and JoBS ignores the RLCs for resolving them. The amounts of traffic dropped due to a buffer overflow comply to the RLCs, but become negligible compared to the amount of traffic dropped for satisfying the ADCs. Thus, JoBS (heuristic) drops mostly from Class 1, as shown on Figures 9(b) and 10(b). When the RLCs are ignored, the set of constraints is reduced to relative and absolute delay constraints only, and is feasible. Hence, JoBS (heuristic) does not suffer from these delay oscillations, at the expense of violating the relative loss constraints.[7] To a lesser extent, Figures 7(c)-(f) also show that even when the relative delay constraint between Classes 1 and 2 is removed, JoBS (optimization) presents more oscillations as far as Class-1 delays are concerned. Here, JoBS (optimization) changes the rate allocation upon each arrival, while JoBS (heuristic) changes the rate allocation every N arrivals only. Since there are no requirements on the delay variations (jitter), JoBS (optimization) tries to minimize Class-1 delays as much as it can, which explains these oscillations.

The third observation is that in the second experiment, removing the RDC between classes 1 and 2 is sufficient for ensuring that the set of constraints is feasible. Hence, the relative loss constraints are respected, as shown on Figures 9(c-d) and 10(c-d). Figures 8(c-d) show that, in absence of this RDC, the ratio of Class-2 delays over Class-1 delays exceeds a factor of 30 under high loads, and even reaches 500 in the case of the optimization model.

6.4 Computational Overhead

We conclude by giving some comparative values of the computational overheads of both JoBS (optimization) and JoBS (heuristic). All simulations were performed on Sun Ultra-SPARC 250, running Solaris 2.7, with a 400 MHz CPU. The simulations were the only load on the processor. On Table 1, we present the time to complete the three cases of Experiment 2. We use the time to complete the same experiments with a FIFO/Drop-Tail queue as a benchmark. Since such a queue does not provide any support for service guarantees, the time to complete is the same in all three cases.

Based on these results, one can make three observations. First, in the case of JoBS (optimization), the experiment that completed the fastest is ‘With ADC, all RDCs’, which is the most constrained system we tested. In such a case, the solution space is smaller, and consequently, the search for the optimal solution is easier.

Second, the optimization model is computationally expensive. The time to complete the experiments is two orders of magnitude larger than our benchmark in all cases. This confirms that JoBS (optimization) is

[7]The delay values for Classes 2, 3, and 4 in Figures 7(b) and (d) appear similar, especially since we use a log-scale. We emphasize that the values are *not* identical, and that the results are consistent.

too slow for a practical implementation at high speeds.

Third, JoBS (heuristic) runs one or two orders of magnitude faster than JoBS (optimization). The results presented earlier showed that the heuristic closely approximates the optimization model in terms of service guarantees provided. Thus, even if the heuristic is slower by a factor of 2-3 than a FIFO/Drop-Tail implementation, one can infer that a close approximation of the optimization model is achievable at high speeds.

7 Discussion and Conclusions

The main contribution of this paper is a new framework, referred to as JoBS (Joint Buffer Management and Scheduling), for reasoning about relative and absolute per-class service differentiation in a network without information on traffic arrivals. JoBS reconciles scheduling and buffer management into a single algorithm, thus, acknowledging that scheduling and buffer management are not orthogonal issues, but should be dealt with in concert. JoBS makes predictions on the delays of backlogged traffic, and uses the predictions to update the service rates of classes and the amount of traffic to be dropped. A unique capability of JoBS is its ability to provide relative and absolute per-class service differentiation for delays and loss rate. We have demonstrated the effectiveness of JoBS in a set of simulation experiments.

As future work, we are interested in extending the JoBS approach to support TCP congestion control. As a point of departure, we conjecture that many active queue management algorithms, e.g., RED [12] and RIO [6], can be expressed within the JoBS framework. We are also working towards an implementation of JoBS-style algorithms on PC-based IP routers, running the Alternate Queueing Framework (ALTQ, [5]) under the FreeBSD operating system [14].

References

[1] G. Armitage and K. Adams. ATM adaptation layer packet reassembly during cell loss. *IEEE Network*, 7(10):26–34, September 1993.

[2] 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.

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

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

[5] K. Cho. A framework for alternate queueing: Towards traffic management by PC-UNIX based routers. In *Proceedings of USENIX 1998 Annual Technical Conference*, New Orleans, LA, June 1998.

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

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

[8] C. Dovrolis and P. Ramanathan. A case for relative differentiated services and the proportional differentiation model. *IEEE Networks*, 13(5):26–34, September 1999. Special issue on Integrated and Differentiated Services on the Internet.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] The FreeBSD project. http://www.freebsd.org.

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

[16] 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.

[17] 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.

[18] 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, 1996.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] A. Romanow and S. Floyd. Dynamics of TCP traffic over ATM networks. *IEEE Journal on Selected Areas in Communications*, 13(4):633–641, May 1995.

[28] 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.

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

[30] I. Stoica and H. Zhang. Providing guaranteed services without per-flow management. In *Proceedings of ACM SIGCOMM'99*, pages 81–94, Boston, MA., August 1998.

[31] 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.

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

A Expressing the Constraints in Function of $\mathbf{x}_s$

The objective of this appendix is to give a full specification of the constraints $g_j(\mathbf{x}_s)$ and $h_j(\mathbf{x}_s)$ as stated in the general form of the optimization problem given in Eqn. (21), where $g_j(\mathbf{x}_s)$ denotes equality constraints and $h_j(\mathbf{x}_s)$ denotes inequality constraints.

The work to be done is to express the constraints defined by Eqs. (22), (23), (24), in terms of the optimization variable $\mathbf{x}_s$, that is, in terms of the $r_i(s)$'s and $\ell_i(s)$'s.

Throughout this appendix, we will use the following shorthand notation:

$$\forall(t_1,t_2)\ ,\ \left\{ \begin{array}{rcl} R_i^{in}(t_1,t_2) & = & R_i^{in}(t_2) - R_i^{in}(t_1) \\ A_i(t_1,t_2) & = & A_i(t_2) - A_i(t_1) \end{array} \right. . \tag{31}$$

In addition to this, it will become clear that most constraints have the same form for all classes. Thus, we will use h_j^i to denote the j-th constraint applied to class i.

A.1 Equality Constraints

There is only one equality constraint, which is the work- conserving property $\sum_{i=1}^{Q} r_i(s) = C$. Thus,

$$g_1(\mathbf{x}_s) = \sum_{i=1}^{Q} r_i(s) - C\ . \tag{32}$$

A.2 Inequality Constraints

We will use the convention that h_j^i denotes the j-th inequality constraint, applied to class i.

A.2.1 General Bounds

For each class i, we have $r_i(s) \geq 0$, $\ell_i(s) \geq 0$ and $\ell_i(s) \leq B_i(s)$. Thus, we get

$$\begin{array}{ll} h_2^i(\mathbf{x}_s) = r_i(s)\ , & i = 1, \dots, Q\ , \\ h_3^i(\mathbf{x}_s) = \ell_i(s)\ , & i = 1, \dots, Q\ , \\ h_4^i(\mathbf{x}_s) = B_i(s) - \ell_i(s)\ , & i = 1, \dots, Q\ . \end{array} \tag{33}$$

A.2.2 Buffer Size

At time s, we need to have $\sum_i B_i(s) \leq B$. Since

$$B_i(s) = B_i(s^-) + a_i(s) - \ell_i(s), \quad \forall i\ , \tag{34}$$

the buffer size constraint can be rewritten as

$$\sum_i \big(B_i(s^-) + a_i(s) - \ell_i(s)\big) \leq B\ , \tag{35}$$

where the only variable is $\ell_i(s)$. This gives us

$$h_5(\mathbf{x}_s) = B - \sum_{i=1}^{Q} \big(B_i(s^-) + a_i(s) - \ell_i(s)\big)\ . \tag{36}$$

Note that the finite buffer size constraint translates into a single constraint for the whole system, which is the reason why we do not use a superscript here.

A.2.3 Absolute Delay Constraints (ADC)

Here, we need to express Eqn. (22) in terms of the optimization variable. We have derived a sufficient condition for meeting the ADC for class i in Eqn. (30), which immediately gives:

$$B_i(s^-) + a_i(s) - \ell_i(s) - r_i(s)(d_i - D_i(s)) \leq 0 . \tag{37}$$

Here, the only variables are $r_i(s)$ and $\ell_i(s)$, and we obtain

$$h_6^i(\mathbf{x}_s) = (d_i - D_i(s))r_i(s) - (B_i(s^-) + a_i(s) - \ell_i(s)) , \quad i = 1, \dots , Q . \tag{38}$$

A.2.4 Relative Delay Constraints (RDC)

We start from Eqn. (23). This equation uses $\overline{D}_{i,s}$, which is defined in Eqn. (15) and solely depends on the projected horizon and on the values of $\tilde{D}_{i,s}(.)$. As soon as a class is backlogged, the projected horizon is defined. We obtain:

$$\begin{aligned} \tilde{T}_{i,s} &= \frac{B_i(s)}{r_i(s)} \\ &= \frac{B_i(s^-) + a_i(s) - \ell_i(s)}{r_i(s)} , \end{aligned} \tag{39}$$

which shows that the projected horizon is a function of $\ell_i(s)$ and $r_i(s)$. This implies that $\tilde{D}_{i,s}$, as defined in Eqn. (10), is also a function of $\mathbf{x}_s$. This, in turn, implies that $\overline{D}_{i,s}$ is a function of $\mathbf{x}_s$ as well. We finally get

$$\left\{ \begin{array}{lcll} h_7^i(\mathbf{x}_s) &=& \overline{D}_{i+1,s} - k_i(1-\varepsilon)\overline{D}_{i,s} , & i = 1, \dots , Q-1 , \\ h_8^i(\mathbf{x}_s) &=& k_i(1+\varepsilon)\overline{D}_{i,s} - \overline{D}_{i+1,s} , & i = 1, \dots , Q-1 . \end{array} \right. \tag{40}$$

According to Eqs. (15), (10), and (39), in Eqn. (40), the only variables are $r_i(s)$, $r_{i+1}(s)$, $\ell_i(s)$ and $\ell_{i+1}(s)$. Note that the functions $h_7^i(\mathbf{x}_s)$ and $h_8^i(\mathbf{x}_s)$ are not linear, given that $\overline{D}_{i,s}$ is not a linear function of $\mathbf{x}_s$.

A.2.5 Absolute Loss Constraints (ALC)

The ALCs at time s are defined by Eqn. (24). Using our definition of loss, given by Eqn. (17), and denoting the beginning of the current busy period by t_0, Eqn. (24) reduces to

$$1 - \frac{R_i^{in}(s-t_0, s^-) + a_i(s) - \ell_i(s)}{A_i(s-t_0, s)} \leq L_i , \tag{41}$$

which immediately gives

$$\ell_i(s) \leq (L_i - 1)\, A_i(s-t_0, s) + R_i^{in}(s-t_0, s^-) + a_i(s) , \tag{42}$$

where the only variable is $\ell_i(s)$. This translates into:

$$h_9^i(\mathbf{x}_s) = (L_i - 1)\, A_i(s-t_0, s) + R_i^{in}(s-t_0, s^-) + a_i(s) - \ell_i(s) , \quad i = 1, \dots , Q . \tag{43}$$

A.2.6 Relative Loss Constraints (RLC)

At time s, the relative loss constraints are specified by Eqn. (25). From Eqn. (17), $p_{i,s}$ can be expressed as a function of $\ell_i(s)$. Therefore, the RLCs can be expressed as a function of $\mathbf{x}_s$. We finally get:

$$\begin{aligned} h^i_{10}(\mathbf{x}_s) &= p_{i+1,s} - k'_i(1-\varepsilon')p_{i,s} \ , \quad i = 1, \ldots, Q-1 \ , \\ h^i_{11}(\mathbf{x}_s) &= k'_i(1+\varepsilon')p_{i,s} - p_{i+1,s} \ , \quad i = 1, \ldots, Q-1 \ . \end{aligned} \tag{44}$$

A.3 Summary

This concludes the derivation of the constraints in function of the optimization variable $\mathbf{x}_s$. At time s, all of the QoS and system constraints are expressed in terms of $r_i(s)$ and $\ell_i(s)$, therefore in terms of the vector $\mathbf{x}_s$. Among this set of constraints, the RDCs are non-linear, which makes this problem solvable by non-linear programming methods only. In summary, the optimization problem described by Eqn. (21) is fully defined by Eqs. (26), (32), (33), (36), (38), (40), (43), and (44):

$$\begin{array}{lll} \textbf{Minimize} & \sum_{i=1}^{Q}(r_i(s) - r_i(s^-))^2 + C^2 \sum_{i=1}^{Q} \ell_i(s) & \\ \textbf{Subject to} & \sum_{i=1}^{Q} r_i(s) - C & = 0 \ , \\ & r_i(s) & \geq 0 \ , \quad i = 1, \ldots, Q \ , \\ & \ell_i(s) & \geq 0 \ , \quad i = 1, \ldots, Q \ , \\ & B_i(s) - \ell_i(s) & \geq 0 \ , \quad i = 1, \ldots, Q \ , \\ & B - \sum_{i=1}^{Q} (B_i(s^-) + a_i(s) - \ell_i(s)) & \geq 0 \ , \\ & (d_i - D_i(s))r_i(s) - (B_i(s^-) + a_i(s) - \ell_i(s)) & \geq 0 \ , \quad i = 1, \ldots, Q \ , \\ & \overline{D}_{i+1,s} - k_i(1-\varepsilon)\overline{D}_{i,s} & \geq 0 \ , \quad i = 1, \ldots, Q-1 \ , \\ & k_i(1+\varepsilon)\overline{D}_{i,s} - \overline{D}_{i+1,s} & \geq 0 \ , \quad i = 1, \ldots, Q-1 \ , \\ & (L_i - 1)\, A_i(s - t_0, s) + R_i^{in}(s - t_0, s^-) + a_i(s) - \ell_i(s) & \geq 0 \ , \quad i = 1, \ldots, Q \ , \\ & p_{i+1,s} - k'_i(1-\varepsilon')p_{i,s} & \geq 0 \ , \quad i = 1, \ldots, Q-1 \ , \\ & k'_i(1+\varepsilon')p_{i,s} - p_{i+1,s} & \geq 0 \ , \quad i = 1, \ldots, Q-1 \ . \end{array} \tag{45}$$

B Systems of Equations Used in JoBS (heuristic)

In this appendix, we fully specify the systems of equations that is solved by the JoBS heuristic in Section 5 in the case of a buffer overflow or an RDC violation. We will also discuss the run-time complexity of the computation of the solutions to these systems of equations.

B.1 Buffer Overflow

We denote the size of the arriving packet by z, and the beginning of the current busy period by t_0. Using Eqn. (17) as the definition of the loss in the RLCs given by Eqn. (25) (with $\varepsilon' = 0$), we get

$$\frac{A_{i+1}(s-t_0,s) - \left(R_{i+1}^{in}(s-t_0,s^-) + a_{i+1}(s) - \ell_{i+1}(s)\right)}{A_i(s-t_0,s) - \left(R_i^{in}(s-t_0,s^-) + a_i(s) - \ell_i(s)\right)} \cdot \frac{A_i(s-t_0,s)}{A_{i+1}(s-t_0,s)} = k_i' \ , \quad \forall i \in (1,\dots,Q-1) \ , \tag{46}$$

This gives us a system of $Q-1$ equations. If we add to this system the relationship giving the amount of traffic to drop, we then get the following system:

$$\left\{ \begin{array}{c} \frac{A_{i+1}(s-t_0,s) - \left(R_{i+1}^{in}(s-t_0,s^-) + a_{i+1}(s) - \ell_{i+1}(s)\right)}{A_i(s-t_0,s) - \left(R_i^{in}(s-t_0,s^-) + a_i(s) - \ell_i(s)\right)} \cdot \frac{A_i(s-t_0,s)}{A_{i+1}(s-t_0,s)} = k_i' \ , \quad \forall i \in (1,\dots,Q-1) \ , \\ \sum_{i=1}^{Q} \ell_i(s) = z \ , \end{array} \right. \tag{47}$$

that we can rewrite in terms of a matrix equality:

$$\overbrace{\begin{pmatrix} \alpha_1'(s) & \beta_1'(s) & & (0) \\ & \ddots & \ddots & \\ (0) & & \alpha_{Q-1}'(s) & \beta_{Q-1}'(s) \\ 1 & \dots & \dots & 1 \end{pmatrix}}^{\mathbf{M}'_s} \cdot \begin{pmatrix} \ell_1 \\ \vdots \\ \vdots \\ \ell_Q \end{pmatrix} = \begin{pmatrix} \gamma_1'(s) \\ \vdots \\ \gamma_{Q-1}'(s) \\ z \end{pmatrix} \ , \tag{48}$$

where

$$\begin{array}{rcl} \alpha_i'(s) & = & -k_i' A_{i+1}(s-t_0,s) \ , \\ \beta_i'(s) & = & A_i(s-t_0,s) \ , \\ \gamma_i'(s) & = & k_i'(A_i(s-t_0,s) - R_i^{in}(s-t_0,s^-) - a_i(s))A_{i+1}(s-t_0,s) \\ & & -(A_{i+1}(s-t_0,s) - R_{i+1}^{in}(s-t_0,s^-) - a_{i+1}(s))A_i(s-t_0,s) \ , \end{array} \tag{49}$$

are known at time s.

We ensure that $\alpha_i'(s) \neq 0$ and $\beta_i'(s) \neq 0$ by only considering RLCs between backlogged classes. In case some classes are not backlogged at time s, we set their ℓ_i to zero and use the solution of the subproblem taking into account only the backlogged classes. The derivation is similar to the general case we present in this paragraph. Thus, $\mathbf{M}'_s$ always has an inverse, which can be precomputed if the number of classes, Q, is known. Then, the general solution of the system is also known *a priori*. The runtime operation then reduces to multiplying ${\mathbf{M}'}_s^{-1}$ by the righthand vector, which can be done in $O(Q^2)$ in the worst case.

B.2 RDC Violation

In the context of a heuristic approximation that can only perform simple operations, the system used in the optimization model for enforcing the RDCs, as described by Eqn. (40), is too complex. First, since the system is not linear, there is no general method for finding an exact solution easily. Second, the computation of the average delay requires to look up the delays of all packets present in the queue. We work around both of these problems using the four following estimates:

$$\begin{array}{l}\text{(E1) The average delay is given by Eqn. (29)}\ ,\\ \text{(E2) } R_i^{in}(s) \approx R_i^{in}(s^-)\ ,\\ \text{(E3) } B_i(s) \approx B_i(s^-)\ ,\\ \text{(E4) } r_i(s)r_{i+1}(s) \approx r_i(s^-)r_{i+1}(s^-) + r_i(s^-)(r_{i+1}(s) - r_{i+1}(s^-)) + r_{i+1}(s^-)(r_i(s) - r_i(s^-))\ .\end{array} \tag{50}$$

The first estimate, (E1), detailed in Section 5, is an estimate of the average delay in the system that only requires to look up two packets. The second and third estimates, (E2) and (E3), express the fact that the input curve and the backlog shall not be affected in the search of a solution meeting the relative delay constraints. This also helps minimizing the losses. The fourth estimate, (E4), enables us to make the system linear by stating that the rate variations are small compared to the actual rates and that the second-order terms are negligible. Given that the system tries to minimize those rate variations, this last assumption generally holds, and the error due to this estimate is indeed negligible.

With the first three estimates (E1), (E2) and (E3), and $\varepsilon = 0$, we get the following set of $Q-1$ equations:

$$\frac{D_{i+1}(s) + \max_{x\geq 0}\{x \mid R_{i+1}^{in}(s^-) = R_{i+1}^{in}(s-x)\} + \frac{B_{i+1}(s^-)}{r_{i+1}(s)}}{D_i(s) + \max_{x\geq 0}\{x \mid R_i^{in}(s^-) = R_i^{in}(s-x)\} + \frac{B_i(s^-)}{r_i(s)}} = k_i\ . \tag{51}$$

The fourth estimate (E4) enables us to rewrite the RDC as

$$\alpha_i(s)r_i(s) + \beta_i(s)r_{i+1}(s) = \gamma_i(s)\ , \tag{52}$$

where

$$\begin{array}{rcl}\alpha_i(s) &=& B_{i+1}(s^-) - r_{i+1}(s^-)\omega_i(s)\ ,\\ \beta_i(s) &=& -k_iB_i(s^-) - r_i(s^-)\omega_i(s)\ ,\\ \gamma_i(s) &=& -r_i(s^-)r_{i+1}(s^-)\omega_i(s)\ ,\end{array} \tag{53}$$

and

$$\begin{array}{rcl}\omega_i(s) &=& k_i\left(D_i(s) + \max_{x\geq 0}\{x \mid R_i^{in}(s^-) = R_i^{in}(s-x)\}\right)\\ && -\left(D_{i+1}(s) + \max_{x\geq 0}\{x \mid R_{i+1}^{in}(s^-) = R_{i+1}^{in}(s-x)\}\right)\ .\end{array} \tag{54}$$

Together with the workconserving property, $\sum_i r_i(s) = C$, one obtains a system of Q equations, which can also be written in terms of a matrix equality:

$$\overbrace{\begin{pmatrix}\alpha_1(s) & \beta_1(s) & & (0)\\ & \ddots & \ddots & \\ (0) & & \alpha_{Q-1}(s) & \beta_{Q-1}(s)\\ 1 & \dots & \dots & 1\end{pmatrix}}^{\mathbf{M}_s} \cdot \begin{pmatrix} r_1\\ \vdots\\ \vdots\\ r_Q\end{pmatrix} = \begin{pmatrix}\gamma_1(s)\\ \vdots\\ \gamma_{Q-1}(s)\\ C\end{pmatrix}\ , \tag{55}$$

for which the algorithm picks a solution. This system of equations always has the same form, the matrix $\mathbf{M}_s$ that characterizes it is always invertible since we only consider the backlogged classes, and its inverse can be precomputed. Therefore, at runtime, the algorithm simply picks the solution by multiplying the precomputed inverse $\mathbf{M}_s^{-1}$ by the righthand vector. The worst-case complexity order of this operation is quadratic in function of the number of classes. If the solution violates an ADC, the RDCs are relaxed until the ADCs are satisfied.