A Quantitative Assured Forwarding Service *

Technical Report: University of Virginia, CS-2001-21

Nicolas Christin *Jörg Liebeherr* *Tarek F. Abdelzaher*
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

Abstract

The Assured Forwarding (AF) service of the IETF DiffServ architecture provides a qualitative service differentiation between classes of traffic, in the sense that a low-priority class experiences higher loss rates and higher delays than a high-priority class. However, the AF service does not quantify the difference in the service given to classes. In an effort to strengthen the service guarantees of the AF service, we propose a *Quantitative Assured Forwarding* service with absolute and proportional differentiation of loss, service rates, and packet delays. We present a feedback-based algorithm which enforces the desired class-level differentiation on a per-hop basis, without the need for admission control or signaling. Measurement results from a testbed of FreeBSD PC-routers on a 100 Mbps Ethernet network show the effectiveness of the proposed service, and indicate that our implementation is suitable for networks with high data rates.

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

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

1 Introduction

The Assured Forwarding (AF, [14]) service of the Differentiated Services (*DiffServ*, [6]) architecture is an attempt to provide a scalable solution to the problem of service differentiation in the Internet. In the AF service, flows with similar QoS requirements are grouped into classes, using the DiffServ CodePoint field (DSCP, [26]) in the IP header. An attractive feature of the AF service is that it does not require admission control or per-flow classification, and is therefore scalable on both the control and data paths. However, the AF service only provides qualitative differentiation between classes, in the sense that some classes receive lower delays and a lower loss rate than others, but the differentiation is not quantified, and no absolute service bounds are offered.

Recently, research efforts have tried to strengthen the guarantees that can be provided within the context of the AF service without sacrificing its scalability and its simplicity, either by trying to quantify the difference in the level of service received by different classes, or by offering absolute bounds on service parameters, e.g., delays, to a specific set of classes. For instance, the *proportional service differentiation* model [9, 10] quantifies the difference in the service by making the ratios of delays or loss rates of different classes roughly constant. This type of service can be implemented through scheduling algorithms [9, 10, 11, 24, 25] and/or buffer management algorithms [7, 10]. Recent works have tried to combine the scheduling and dropping decisions in a single algorithm [20, 31]. Most scheduling and/or buffer management algorithms aim at proportional differentiation, but do not support absolute service guarantees.

In a different approach to strengthening the AF service, the Alternative Best-Effort (ABE) service considers two traffic classes. The first class obtains absolute delay guarantees, and the second class has no delay guarantees, but is given a better loss rate than the first class. Scheduling and buffer management algorithms for the ABE service are presented in [16]. The service model in [19] also supports absolute delay bounds, and qualitative loss and throughput differentiation, but no proportional differentiation.

These recent efforts to strengthen the AF service raise questions on the best possible class-based service model that can be achieved by entirely relying on scheduling and dropping algorithms at routers, and without admission control, traffic policing, or signaling. In an attempt to explore the limits of such a class-based service, we define in this paper a "Quantitative Assured Forwarding" [1] service that offers, on a per-hop basis, both absolute and proportional guarantees to classes. Each node enforces any mix of absolute and proportional guarantees. Absolute guarantees apply to loss rates, delays, or throughput, and define a lower bound on the service received by each class. Proportional guarantees apply to loss rates and queueing delays. As an example of the guarantees in the Quantitative Assured Forwarding service for three classes of traffic, one could specify service guarantees of the form "Class-1 Delay $\leq$ 2 ms", "Class-2 Delay $\approx$ 4·Class-1 Delay", "Class-2 Loss Rate $\leq$ 1%", "Class-3 Loss Rate $\approx$ 2·Class-2 Loss Rate", and "Class-3 Service Rate $\geq$ 1 Mbps". Clearly, without admission control, it is not feasible to satisfy all absolute guarantees at all times. Thus, when absolute constraints cannot be satisfied, we allow that some service guarantees can be temporarily relaxed according to a specified order.

We present a formal description of the Quantitative Assured Forwarding service, and we devise an algorithm that enforces guarantees on loss, delay and throughput for classes by adjusting the service rate allocation to classes and by selectively dropping traffic. We apply linear feedback control theory for the design of the algorithm, and, to this effect, make assumptions which approximate the non-linearities in the system of study, similar to [15, 21, 22, 28].

[1] The name "quantitative differentiated service" was recently used in [19].

This paper is organized as follows. In Section 2, we define the Quantitative Assured Forwarding service. In Sections 3 and 4, we describe the algorithms which provide the Quantitative Assured Forwarding service. In Section 5, we present an implementation of these algorithms in FreeBSD PC-routers. We evaluate the algorithms using the implementation in Section 6, and present brief conclusions in Section 7.

2 The Quantitative Assured Forwarding Service

In this section, we describe the Quantitative Assured Forwarding Service, and outline a solution for an algorithm that realizes this service.

2.1 Formal Description

We assume that all traffic that arrives to the transmission queue of the output link of a router is marked to belong to one of N classes. We use a convention whereby a class with a lower index receives a better service. We consider a discrete event system, where events are traffic arrivals. We use $t(n)$ to denote the time of the n-th event in the current busy period[2], and $\Delta t(n)$ to denote the time elapsed between the n-th and $(n+1)$-th events. We use $a_i(n)$ and $l_i(n)$, respectively, to denote the class-i arrivals and the amount of class-i traffic dropped ('lost') at the n-th event. We use $r_i(n)$ to denote the service rate allocated to class-i at the time of the n-th event. The service rate of a class i is a fraction of the output link capacity, which can vary over time, and is set to zero if there is no backlog of class-i traffic in the transmission queue. For the time being, we assume bursty arrivals with a fluid-flow service, that is, the output link is viewed as simultaneously serving traffic from several classes. Such a fluid-flow interpretation is idealistic, since traffic is actually sent in discrete sized packets. In Section 5, we discuss how the fluid-flow interpretation is realized in a packet network.

All service guarantees are enforced over the duration of a busy period. An advantage of enforcing service guarantees over short time intervals is that the output link can react quickly to changes of the traffic load. Further, enforcing guarantees only within a busy period requires little state information, and, therefore, keeps the implementation overhead limited. As a disadvantage, at times of low load, when busy periods are short, enforcing guarantees only with information on the current busy period can be unreliable. However, at underloaded links transmission queues are mostly idle and all service classes receive a high-grade service.

The following presentation specifies the service differentiation independently for each busy period. Let $t(0)$ define the beginning of the busy period. The arrival curve at n-th event, $A_i(n)$, is the total traffic that has arrived to the transmission queue of an output link at a router since the beginning of the current busy period, that is

$$A_i(n) = \sum_{k=0}^{n} a_i(k) \, .$$

The input curve, $R_i^{in}(n)$, is the traffic that has been entered into the transmission queue at the n-th event,

$$R_i^{in}(n) = A_i(n) - \sum_{k=0}^{n} l_i(k) \, .$$

[2]The beginning of the current busy period is defined as the last time when the transmission queue at the output link was empty.

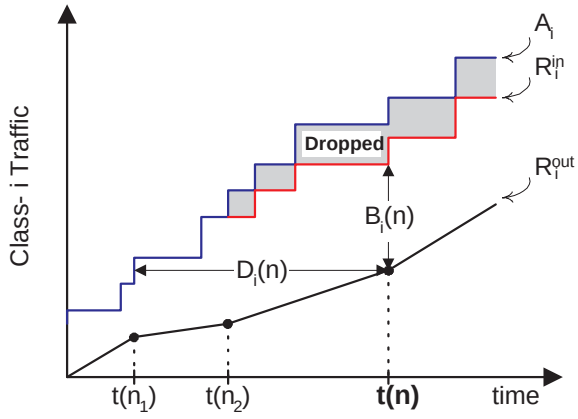


Figure 1: **Delay and backlog at the transmission queue of an output link.** A_i is the arrival curve, R_i^{in} is the input curve and R_i^{out} is the output curve.

The output curve is the traffic that has been transmitted since the beginning of the current busy period, that is

$$R_i^{out}(n) = \sum_{k=0}^{n-1} r_i(k)\Delta t(k) \ . \tag{3}$$

In Figure 1, we illustrate the concepts of arrival curve, input curve, and output curve for class-i traffic. At any time $t(n)$, the service rate is the slope of the output curve. In the figure, the service rate is adjusted at times $t(n_1), t(n_2)$ and $t(n)$.

As illustrated in Figure 1, for event n, the vertical and horizontal distance between the input and output curves, respectively, denote the class-i backlog $B_i(n)$ and the class-i delay $D_i(n)$. For the n-th event, we have

$$B_i(n) = R_i^{in}(n) - R_i^{out}(n) \ ,$$

and

$$D_i(n) = t(n) - t\big(\max\{k < n \mid R_i^{out}(n) \geq R_i^{in}(k)\}\big) \ . \tag{4}$$

Eqn. (4) characterizes the delay of the class-i traffic that departs at the n-th event.

We define the 'loss rate' to be the ratio of dropped traffic to the arrivals. That is

$$p_i(n) = \frac{A_i(n) - R_i^{in}(n)}{A_i(n)} \ . \tag{5}$$

Since, from the definition of $A_i(n)$ and $R_i^{in}(n)$, the $p_i(n)$ are computed only over the current busy period, they correspond to long-term loss rates only if busy periods are long. We justify our choice with the observation that traffic is dropped only at times of congestion, i.e., when the link is overloaded, and, hence, when the busy period is long.

With these metrics, we can express the service guarantees of a Quantitative Assured Forwarding service. An absolute delay guarantee on class i is specified as

$$\forall n : D_i(n) \leq d_i \ , \tag{6}$$

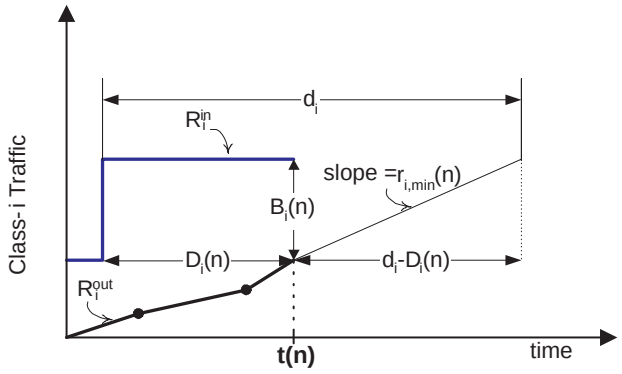


Figure 2: **Determining service rates for delay guarantees.**

where d_i is the delay bound of class i. Similarly, an absolute loss rate bound for class i is defined by

$$\forall n : p_i(n) \leq L_i \ . \tag{7}$$

An absolute rate guarantee for class i is specified as

$$\forall n : B_i(n) > 0 \ , r_i(n) \geq \mu_i \ . \tag{8}$$

The proportional guarantees on delay and loss, respectively, are defined, for all n such that $B_i(n) > 0$ and $B_{i+1}(n) > 0$, as

$$\frac{D_{i+1}(n)}{D_i(n)} = k_i \ \ , \tag{9}$$

and

$$\frac{p_{i+1}(n)}{p_i(n)} = k_i' \ , \tag{10}$$

where k_i and k_i' are constants that quantify the proportional differentiation desired.

2.2 Rate Allocation and Drop Decisions

We now sketch a solution for providing the service guarantees specified in Eqs. (6)-(10) at the output link of a router with capacity C and buffer size B. We assume per-class buffering of incoming traffic, thus, each class is transmitted in a First-Come-First-Served manner. In the proposed solution, the service rates $r_i(n)$ and the amount of dropped traffic $l_i(n)$ are adjusted at each event n so that the constraints defined by Eqs. (6)-(10) are met. If not all constraints in Eqs. (6)-(10) can be met at the n-th event, then some service guarantees need to be temporarily relaxed. We assume that the order in which guarantees are relaxed is given.

The absolute delay guarantee on class i, d_i, imposes a minimum required service rate in the sense that all backlogged class-i traffic at the n-th event will be transmitted within its delay bound if

$$r_i(n) \geq \frac{B_i(n)}{d_i - D_i(n)} \ .$$

This condition can be verified by inspection of Figure 2. If the condition holds for any n, the delay bound d_i is never violated. If class i has, in addition, an absolute rate guarantee μ_i, the expression for the minimum rate needed by class i at the n-th event, becomes [3]

$$r_{i,min}(n) = \max\left\{\frac{B_i(n)}{d_i - D_i(n)}, \mu_i \cdot \chi_{B_i(n)>0}\right\}. \tag{12}$$

The service rate that can be allocated to class i is upper bounded by the output link capacity minus the minimum service rates needed by the other classes, that is,

$$r_{i,max}(n) = C - \sum_{j \neq i} r_{j,min}(n) .$$

Therefore, the service rate can take any value $r_i(n)$ with

$$r_{i,min}(n) \leq r_i(n) \leq r_{i,max}(n) ,$$

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

We view the computation of $r_i(n)$ in terms of the recursion

$$r_i(n) = r_i(n-1) + \Delta r_i(n) , \tag{15}$$

where $\Delta r_i(n)$ is selected such that the constraints of proportional delay differentiation are satisfied at event n. From Eqs. (3) and (4), the delay $D_i(n)$ at the n-th event is a function of $r_i(k)$ with $k < n$. By monitoring $D_i(n)$ we can thus determine the deviation from the desired proportional differentiation resulting from past service rate allocations, and infer the adjustment $\Delta r_i(n) = f(D_i(n))$ needed to attenuate this deviation.

If no feasible service rate allocation for meeting all delay guarantees exist at the n-th event, or if there is a buffer overflow at the n-th event, traffic must be dropped, either from a new arrival or from the current backlog. The loss guarantees determine which class(es) suffer(s) traffic drops at the n-th event.

To enforce loss guarantees, we rewrite the loss rate, defined by Eqn. (5), as a difference equation

$$p_i(n) = p_i(n-1)\frac{A_i(n-1)}{A_i(n)} + \frac{l_i(n)}{A_i(n)} . \tag{16}$$

From Eqn. (16), we can determine how the loss rate of class i evolves if traffic is dropped from class i at the n-th event. Thus, we can determine the set of classes that can suffer drops without violating absolute loss guarantees. In this set, we choose the class whose loss rate differs by the largest amount from the objective of Eqn. (9).

Having expressed the service rate and the loss rate in terms of a recursion, we can characterize the service rate allocation and dropping algorithm as feedback control problems. In the next sections, we will describe two feedback problems: one for delay and absolute rate differentiation ('delay feedback loop'), and one for loss differentiation ('loss feedback loop'). We describe the interaction of the two feedback problems in Section 5.

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

3 The Delay Feedback Loop

In this section, we present feedback loops which enforce the desired delay and rate differentiation given by Eqs. (6), (8), and (9). We have one feedback loop for each class with proportional delay guarantees. In the feedback loop for class i, we characterize changes to service rate $\Delta r_i(n)$ by approximating the non-linear effects of the service rate adjustment on the delays by a linear system, and derive stability conditions for the linearized control loop.

3.1 Objective

Let us assume for now that all classes are offered proportional delay guarantees. Later, this assumption will be relaxed. The set of constraints given by Eqn. (9) leads to the following system of equations:

$$\begin{aligned} D_2(n) &= k_1 \cdot D_1(n) \,, \\ &\vdots \\ D_N(n) &= \left(\textstyle\prod_{j=1}^{N-1} k_j\right) D_1(n) \,. \end{aligned} \tag{17}$$

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

$$D_i^*(n) = \left(\prod_{k=1,\, k\neq i}^{N} m_k \right) D_i(n) \,. \tag{18}$$

By multiplying each line of Eqn. (17) with $\prod_{j\neq i} m_j$, we see that the desired proportional delay differentiation is achieved for all classes if

$$\forall i, j \,, \forall n : D_i^*(n) = D_j^*(n) \,. \tag{19}$$

Eqn. (19) is equivalent to

$$\forall i, j \,, \forall n : D_i^*(n) = \overline{D}^*(n) \,,$$

where

$$\overline{D}^*(n) := \frac{1}{N} \sum_i D_i^*(n) \,.$$

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

Remark: We view event numbers, n, as sampling times on a virtual time axis in which events are equidistant. Hence, convergence of the control loop applies to virtual time. However, the relationship between delay and rate is independent of the time axis chosen. By virtue of this independence, and since real-time is monotonically increasing with virtual time, we make the assumption that the skew between virtual-time and real-time can be neglected, and that the convergence condition we present later applies to real-time as well.

3.2 Service Rate Adjustment

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

$$e_i(n) = \overline{D}^*(n) - D_i^*(n) \,. \tag{22}$$

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

$$\sum_i e_i(n) = N\overline{D}^*(n) - \sum_i D_i^*(n) = 0 \, .$$

If proportional delay differentiation is achieved, we have $e_i(n) = 0$ for all classes. We use the error $e_i(n)$ to compute the service rate adjustment $\Delta r_i(n)$ needed for class i to satisfy the proportional delay differentiation constraints. From Eqn. (22), we note that if $e_i(n) < 0$, $D_i^*(n) > \overline{D}^*(n)$, class i delays are too high with respect to the desired proportional delay differentiation. Therefore, $r_i(n)$ must be increased. Conversely, $e_i(n) > 0$ indicates that class i delays are too low, and $r_i(n)$ must be decreased. Hence, the rate adjustment $\Delta r_i(n)$ is a decreasing function of the error $e_i(n)$, written as $\Delta r_i(n) = f(e_i(n))$, where $f(.)$ is a monotonically decreasing function. We choose

$$\Delta r_i(n) = K(n) \cdot e_i(n) \, , \tag{24}$$

where $K(n) < 0$, which, in feedback control terminology, is the controller. An advantage of this controller is that it requires a single multiplication, and hence is easily implemented in a real system. Another advantage is that, at any n, we have

$$\sum_i \Delta r_i(n) = K(n) \sum_i e_i(n) = 0 \, . \tag{25}$$

Therefore, the controller produces a work-conserving system, as long as the initial condition $\sum_i r_i(0) = C$ is satisfied. Note that systems that are not work-conserving, i.e., where the link may be idle even if there is a positive backlog, are undesirable for networks that need to achieve a high resource utilization.

We now express limits on $K(n)$ so that we can characterize the service rate adjustment needed. These limits are imposed by two different factors. We derive a first condition on $K(n)$ so that the feedback loops are stable, in the sense that they attenuate the errors $e_i(n)$ over time, thereby achieving proportional differentiation. We then derive a second condition on $K(n)$ so that the rate adjustments $\Delta r_i(n)$ do not create a violation of the absolute delay and rate constraints.

3.3 Deriving a Stability Condition on the Delay Feedback Loop

Our goal is to derive a stability condition $K(n)$, so that we can ensure that the delay feedback loops enforce proportional delay differentiation. To derive the stability condition, we first model the effect of the rate adjustment $\Delta r_i(n)$ on the delay $D_i(n)$.

We note that the relationship between delays and rates is non-linear. We thus have two possible strategies for deriving a stability condition on the delay feedback loop of class i [30]. We can either derive a stability condition on the non-linear system, or define the operating point of the system as a triplet $\left(B_i, D_i, r_i\right)$, linearize the non-linear system of study around the operating point, and derive a stability condition on the linearized model. In an effort to get a simple stability condition, we choose the second approach, hence we will linearize the relationships between delays and rates around the operating point, so that we can apply tools provided by linear control theory. This linearization technique has been first described by Lyapunov in [23], which "serves as the fundamental justification of using linear control techniques in practice" [30].

We start our modeling of the relationship between rate and delays, by first defining $\tau_i(n)$ as:

$$D_i(n) = t(n) - t(n - \tau_i(n)) \, .$$

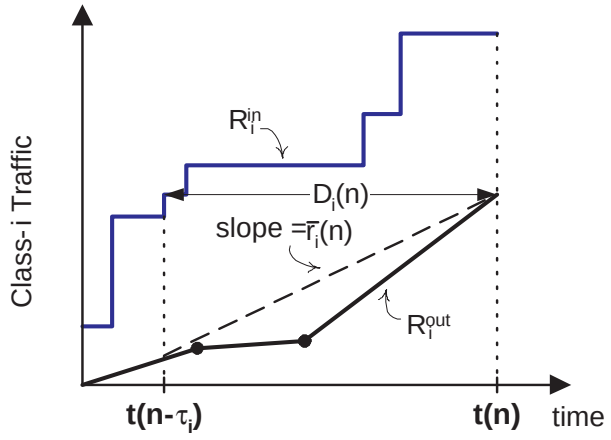


Figure 3: **Definition of the average rate, $\overline{r_i}(n)$.** This figure shows the relationship between $D_i(n)$, τ_i and $\overline{r_i}(n)$.

In other words, $\tau_i(n)$ denotes the delay of class-i traffic departing at the n-th event, expressed as a number of events. Let us make a first assumption,

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

$$D_i(n+1) \approx D_i(n),$$

which implies

$$\tau_i(n) \approx \tau_i(n+1)\ .$$

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

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

$$\overline{r_i}(n) = \frac{B_i(n-\tau_i)}{D_i(n)}\ , \tag{29}$$

which we illustrate in Figure 3. To model the effects of a rate adjustment $\Delta r_i(n)$ on the delay, we can thus first express the relationship between $\Delta r_i(n)$ on $\overline{r_i}(n)$. For this, we make a second assumption,

Assumption (A2). The backlog of class-i traffic does not vary significantly between events $(n-\tau_i)$ and $(n+1-\tau_i)$. Then we can write

$$B_i(n+1-\tau_i) \approx B_i(n-\tau_i),$$

With Assumptions (A1) and (A2), on the virtual time axis described in the remark at the end of Subsection 3.1, where events are equally spaced, we get

$$\overline{r_i}(n+1) = \frac{(\tau_i - 1)\overline{r_i}(n) + r_i(n)}{\tau_i}\ . \tag{31}$$

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

$$\Delta\overline{r_i}(n+1) = \overline{r_i}(n+1) - \overline{r_i}(n)\ . \tag{32}$$

Combining Eqs. (31) and (32), we get

$$\Delta\overline{r_i}(n+1) = \frac{(\tau_i - 1)\Delta\overline{r_i}(n) + \Delta r_i(n)}{\tau_i} \; . \tag{33}$$

Eqn. (33) characterizes the relationship between a change in the service rate and a change in the average rate.

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

$$\Delta D_i(n+1) = D_i(n+1) - D_i(n) \; .$$

Since we have

$$D_i(n) = \frac{B_i(n-\tau_i)}{\overline{r_i}(n)} \; ,$$

and

$$D_i(n+1) = \frac{B_i(n+1-\tau_i)}{\overline{r_i}(n+1)} \; ,$$

we get

$$\Delta D_i(n+1) = \frac{B_i(n+1-\tau_i)}{\overline{r_i}(n+1)} - \frac{B_i(n-\tau_i)}{\overline{r_i}(n)} \; . \tag{37}$$

We introduce here a third assumption:

Assumption (A3). The variations in the average rate are small compared to the average rate. In other words,

$$\Delta\overline{r_i}(n+1) \ll \overline{r_i}(n) \; . \tag{38}$$

We now linearize Eqn. (37), using Assumptions (A1), (A2), and (A3), and obtain

$$\Delta D_i(n+1) = -\frac{B_i(n-\tau_i)}{\overline{r_i}^2(n)}\Delta\overline{r_i}(n+1) + \omega_i(n) \; , \tag{39}$$

where $\omega_i(n)$ is an error resulting from Assumptions (A1), (A2) and (A3). Then, the relationship between delay variations and the delay is given by

$$D_i(n+1) = \sum_{k=0}^{n+1} \Delta D_i(k) \; , \tag{40}$$

$D_i(n+1)$ is used to compute $D_i^*(n+1)$, using Eqn. (18) which characterizes the new deviation $e_i(n+1)$ at the next sampling time (i.e., the next time a rate adjustment is performed). This remark completes the description of a linearized model of the delay feedback loop. We can now turn to the derivation of a stability condition for our linearized model.

The derivation of the stability condition on the linearized model relies on a modeling of the loop using z-transforms of the equations we presented above. We denote the z-transform of a function $f(n)$ by $Z[f(n)]$. We represent the delay feedback loop using z-transforms in Figure 4. Eqs. (18), (22), (24), (39) are unchanged when using z-transforms. Eqn. (33) yields

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

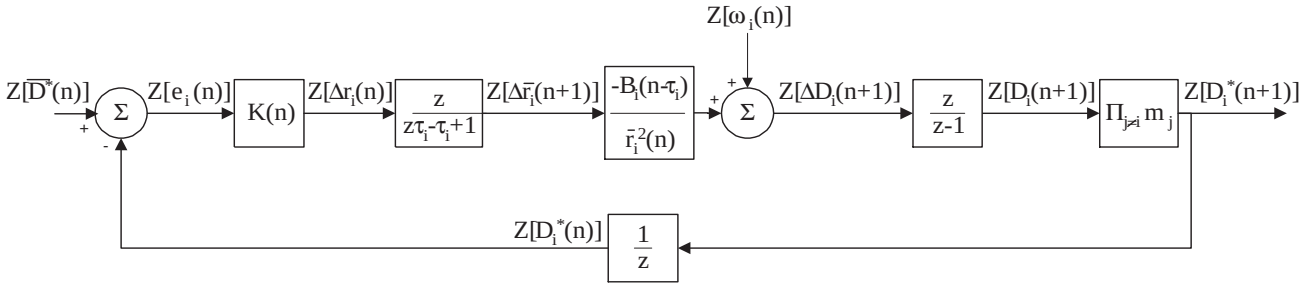


Figure 4: **The class-i delay feedback loop.** This model uses z-transforms of the relationships derived in Section 3.2.

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

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

and by simple reordering of the terms, we obtain,

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

which is equivalent to

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

Similarly, using z-transforms, Eqn (40) becomes

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

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

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

We first notice that in our model, some quantities (e.g., τ_i, B_i, $\overline{r_i}$) are time-dependent. This does not cause stability problems if the product of all individual blocks in Figure 4 (called the 'loop gain'), is non-increasing over time. Since the coefficient $K(n)$ is time-dependent, we will have to select $K(n)$ so that the loop gain is non-increasing over time.

Denoting the loop gain by $G(z)$, a necessary and sufficient condition for the loop to be stable is that the roots of the so-called characteristic equation

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

have a module less than one [13]. Taking the products of all blocks in Figure 4, we get

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

The negative sign comes from the negative feedback $-\frac{1}{z}$. The expression obtained for $G(z)$ is too complicated to yield a stability condition usable in a high-speed computation. We thus approximate the gain of the second block, $\frac{z}{z\tau_i+1-\tau_i}$ by 1. This approximation is motivated by the fact that we have

$$\Delta\overline{r_i}(n+1) \leq \Delta r_i(n) \; .$$

After this approximation, we get a new loop gain, $G'(z)$ such that

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

The characteristic equation on the approximate system is

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

which has exactly one root,

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

We then obtain the following stability condition

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

or, equivalently,

$$-1 \leq 1 + \frac{\left(\prod_{j\neq i} m_j\right) B_i(n-\tau_i)K(n)}{\overline{r_i}^2(n)} \leq 1 \; . \tag{54}$$

All quantities in Eqn. (54), with the exception of $K(n)$, are positive. Hence, the rightmost condition described by Eqn. (54) simply reduces to $K(n) \leq 0$. The leftmost condition in Eqn. (54) becomes

$$K(n) \geq -2 \cdot \frac{\overline{r_i}^2(n)}{\left(\prod_{j\neq i} m_j\right) B_i(n-\tau_i)} \; . \tag{55}$$

Since, from Eqn. (29), we have

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

Eqn (55) becomes

$$K(n) \geq -2 \cdot \frac{B_i(n-\tau_i)}{\left(\prod_{j\neq i} m_j\right) D_i^2(n)} \; . \tag{56}$$

The condition given by Eqn. (56) requires to keep a history of the backlogs, which may be difficult to implement at high speeds. To alleviate this problem, we replace Assumption (A2) by the stronger assumption:

Assumption (A2'). The backlog of class-i traffic does not vary significantly between events $(n - \tau_i)$ and n. We can write

$$B_i(n - \tau_i) \approx B_i(n),$$

which allows us to get a simplified expression for the stability condition for the class-i delay feedback loop:

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

Since $K(n)$ must be common to all classes for Eqn. (25) to hold, we finally get

$$-2 \cdot \min_i \left\{ \frac{B_i(n)}{\prod_{j \neq i} m_j \cdot D_i^2(n)} \right\} \leq K(n) \leq 0 \, . \tag{59}$$

The condition (59) ensures that the delay feedback loops will not engage in divergent oscillations, provided that:

- Assumptions (A1), (A2'), and (A3) hold,
- The approximation that the gain of the second block in Figure 4 is less than one (or equal to one) holds, and
- The clock skew between the virtual-time axis (where events are equally spaced) and the real-time axis (where events are not equally spaced) can be neglected.

However, in practice, we cannot be certain of the validity of these assumptions. Thus, while we cannot make any claim as to the stability of the delay feedback loops resulting from the analysis presented here, the numerical data in Section 6 suggests that the loops converge adequately well.

3.4 Including the Absolute Delay and Rate Constraints.

We have obtained a stability condition on $K(n)$, which is necessary to enforce proportional differentiation. However, so far, we have not considered the absolute delay and rate constraints in the construction of the delay feedback loops. These absolute delay and rate constraints can be viewed as a "saturation constraint" on the rate adjustment, and yield a second bound on $K(n)$. To satisfy the constraints $r_i(n) \geq r_{i,min}(n)$, we may need to clip $\Delta r_i(n)$ when the new rate is below the minimum. This, however, may violate the work-conserving property resulting from Eqn. (25). Hence, we use the following to compute $K(n)$ that would satisfy the saturation constraint

$$r_i(n-1) + K(n) e_i(n) \geq r_{i,min}(n) \, ,$$

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

$$K(n) \geq \max_i \left(\frac{r_{i,min}(n) - r_i(n-1)}{e_i(n)} \right) \, . \tag{61}$$

If

$$\max_i \left(\frac{r_{i,min}(n) - r_i(n-1)}{e_i(n)} \right) > 0 \, ,$$

we see that we cannot have $K(n) < 0$. In other words, we cannot satisfy absolute delay and rate guarantees and proportional delay differentiation at the same time. In such a case, we relax either Eqn. (59) or (61) according to the given precedence order on the service guarantees.

Remark: If proportional delay differentiation is requested for some, but not for all classes, constraints as in Eqn. (17) can be defined for each group of classes with contiguous indices. Then, the feedback loops are constructed independently for each group.

4 The Loss Feedback Loop

We now describe the feedback loop which controls the traffic dropped from a class i to satisfy proportional loss differentiation within the limits imposed by the absolute loss guarantees. As before, we assume that all classes have proportional loss guarantees. The assumption is relaxed similarly as described in the remark at the end of Section 3.

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

$$B \geq \sum_{k=1}^{N} \big(B_k(n-1) + a_k(n) - l_k(n)\big) - \Delta t(n-1)C \; . \tag{63}$$

To provide absolute delay and rate guarantees, the following condition must be satisfied

$$C \geq \sum_{k=1}^{N} \max \left\{ \frac{B_k(n-1) - r_k(n-1)\Delta t(n-1) + a_k(n) - l_k(n)}{d_k - D_k(n)}, \mu_k \cdot \chi_{B_k(n)>0} \right\} \; . \tag{64}$$

To choose the amount of traffic to drop from each class so that Eqs. (63) and (64) hold, we define the weighted loss rate to be

$$p_i^*(n) = \left(\prod_{j=1,\; j\neq i}^{N} m_j' \right) p_i(n) \; ,$$

where $m_i' = \prod_{j=1}^{i-1} k_j'$ for $i > 1$ and $m_1' = 1$. With this definition, Eqn. (10) is equivalent to

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

We choose the following set point for the loss feedback loop

$$\bar{p}^*(n) = \frac{1}{N} \sum_i p_i^*(n) \; ,$$

and we use the set point to describe an error

$$e_i'(n) = \bar{p}^*(n) - p_i^*(n) \; .$$

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

Absolute loss guarantees impose an upper bound, $l_i^*(n)$, on the traffic that can be dropped at event n from class i. The value of $l_i^*(n)$ is determined from Eqs. (7) and (16) as

$$l_i^*(n) = A_i(n)L_i - p_i(n-1)A_i(n-1) \ .$$

If the conditions in Eqs. (63) and (64) are violated, traffic is dropped from class i_1 until the conditions are satisfied, or until the maximum amount of traffic $l_{i_1}^*(n)$ has been dropped. Then traffic is dropped from class i_2, and so forth. Suppose that the conditions in Eqs. (63) and (64) are satisfied for the first time if $l_j^*(n)$ traffic is dropped from classes $j = i_1, i_2, \ldots, i_{\hat{k}-1}$, and $\hat{x}(n) \leq l_{\hat{k}}^*(n)$ traffic is dropped from class $i_{\hat{k}}$, then we obtain:

$$l_i(n) = \begin{cases} l_i^*(n) & \text{if } i = i_1, i_2, \ldots, i_{\hat{k}-1} \ , \\ \hat{x}(n) & \text{if } i = i_{\hat{k}} \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{70}$$

If $l_k(n) = l_k^*(n)$ for all $k = i_1, i_2, \ldots, i_R$, we allow absolute delay and rate conditions to be violated. In other words, condition (64) is relaxed.

The loss feedback loop never increases the maximum error $e_i'(n)$, if $e_i'(n) > 0$ and more than one class is backlogged. Thus, the errors remain bounded and the algorithm presented will not engage in divergent oscillations around the target value $\overline{p}^*(n)$. Additionally, the loss feedback loop and the delay feedback loops are independent of each other, since we always drop traffic from the tail of each per-class buffer, losses do not have any effect on the delays of traffic admitted into the transmission queue.

5 Implementation

We implemented the algorithms presented in Sections 3 and 4 on PC-routers running the FreeBSD v4.3 [1] operating system, using the ALTQ v3.0 package [8]. ALTQ allows programmers to modify the operations of the transmission queue in the IP layer of the FreeBSD kernel. We will discuss the operations performed in our implementation when a packet is entered into the transmission queue of an IP router (packet enqueuing) and when a packet is selected for transmission (packet dequeuing).

We use the DSCP field in the header of a packet to identify the class index of an IP packet. The DSCP field is set by the edge router; in our testbed implementation, this is the first router traversed by a packet.

In our implementation, we chose the following precedence order for relaxing constraints. Absolute loss guarantees have higher precedence than absolute delay and rate guarantees, which have in turn higher precedence than proportional guarantees.

5.1 Packet Enqueuing

The enqueue procedure are the operations executed in the IP layer when a packet is entered into the transmission queue of an output link. Since the FreeBSD kernel is single-threaded, the execution of the enqueue procedure is strictly sequential.

The enqueue procedure performs the dropping decisions and the service rate allocation. We avoid floating point operations in the kernel of the operating system, by expressing delays as machine clock cycles, service rates as bytes per clock cycle (multiplied by a scaling factor of 2^{32}) , and loss rates as fractions of 2^{32}. Then, 64-bit (unsigned) integers provide a sufficient degree of accuracy.

In our modified enqueue procedure, the transmission queue of an output link has one FIFO queue for each class, implemented as a linked list. We limit the total number of packets that can be queued to $B = 200$. Whenever a packet is entered into the FIFO queue of its class, the arrival time of the packet is recorded, and the waiting times of the packets at the head of each FIFO queue are updated.

The enqueue procedure uses the loss feedback loop described in Section 4 to determine if and how much traffic needs to be dropped from each class. In our implementation, the algorithm of Section 4 is run twice. The first time, buffer overflows are resolved by ignoring condition (64); The second time, violations of absolute delay and rate guarantees are resolved by ignoring condition (63).

Next, the enqueue procedure computes new values for $r_{i,min}(n)$ from Eqn. (12), and determines new service rates, using Eqs. (15) and (24), with the constraints on $K(n)$ given in Eqs. (59) and (61). If no feasible value for $K(n)$ exists, Eqn. (59) is ignored, thus, giving absolute delay guarantees precedence over proportional delay guarantees.

5.2 Packet Dequeuing

The dequeue procedure selects one packet from the backlog for transmission. In our implementation, dequeue selects one of the traffic classes, and picks the packet at the head of the FIFO queue for this class.

The dequeue procedure uses a rate-based scheduling algorithm to adapt the transmission rates $r_i(n)$ from a fluid-flow view to a packet-level environment. Such an adaptation can be performed using well-known rate-based scheduling algorithm techniques, e.g., as VirtualClock [32] or PGPS [27]. These scheduling algorithms translate a rate allocation, into 'virtual' deadlines of individual packets. In our implementation, we use a modified Deficit Round Robin (DRR, [29]) scheduling algorithm. Let $Xmit_i(n)$ denote the number of bytes of class-i traffic that have been transmitted in the current busy period, the scheduler selects a packet from class i for transmission if

$$i = \arg\max_k \left\{ R_k^{out}(n) - Xmit_k(n) \right\} .$$

In other words, the dequeue procedure selects the class which is the most behind its theoretical output curve.

6 Evaluation

We present experimental measurements of our implementation of the Quantitative Assured Forwarding service on a testbed of PC routers. The PCs are Dell PowerEdge 1550 with 1 GHz Intel Pentium-III processors and 256 MB of RAM. The system software is FreeBSD 4.3 and ALTQ 3.0. Each system is equipped with five 100 Mbps-Ethernet interfaces.

In our experiments we determine if and how well our algorithm provides the desired service differentiation on a per-node basis. In addition, we want to observe the stability of the feedback loops, and their robustness to changes in the network topology and in the service guarantees. To that effect we propose four experiments, with different network topologies, service guarantees, and traffic patterns. We then present an evaluation of the overhead associated to our proposed algorithm.

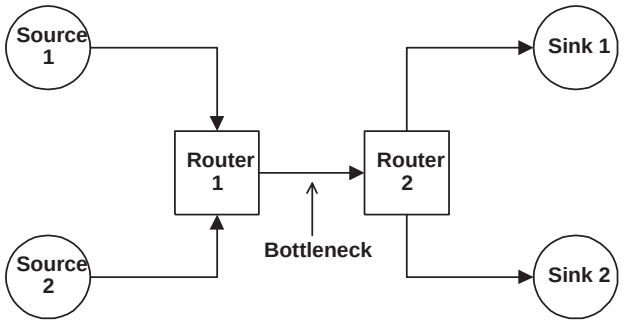


Figure 5: **Experiments 1 and 2: Network Topology.** All links have a capacity of 100 Mbps. We measure the service provided by Router 1 at the indicated bottleneck link.

Class	**Service Guarantees**				
	d_i	L_i	μ_i	k_i	k_i'
1	3 ms	0.1 %	–	–	–
2	–	–	35 Mbps	2	2
3	–	–	–	2	2
4	–	–	–	N/A	N/A

Table 1: **Experiments 1 and 2: Service guarantees.** The guarantees are identical at each router.

6.1 Experiment 1: Single-node topology, near-constant load

We use a local network topology using point-to-point Ethernet links as shown in Figure 5. All links are full-duplex and have a capacity of $C = 100$ Mbps. Two PCs are set up as routers, indicated in Figure 5 as Router 1 and 2. Other PCs are acting as sources and sinks of traffic. The topology has a bottleneck, which is the link between Routers 1 and 2. As mentioned earlier, the buffer size at the output link of each router is set to $B = 200$ packets.

We consider four traffic classes with service guarantees as summarized in Table 1. Sources 1 and 2 send traffic to Sinks 1 and 2, respectively. Source 1 transmits traffic from classes 1 and 2, Source 2 transmits traffic from classes 3 and 4. The traffic mix, the number of flows per class, and the characterization of the

Source	**Class**	**No. of flows**	**Type**	
			Protocol	Traffic
1	1	6	UDP	On-off
	2	10	TCP	Greedy
2	3	10	TCP	Greedy
	4	10	TCP	Greedy

Table 2: **Experiment 1: Traffic mix.** The on-off UDP source sends bursts of 25 packets during an on-period, and have a 150 ms off-period. All TCP sources are greedy, i.e., they always have data to transmit, and run the *NewReno* congestion control algorithm [12].

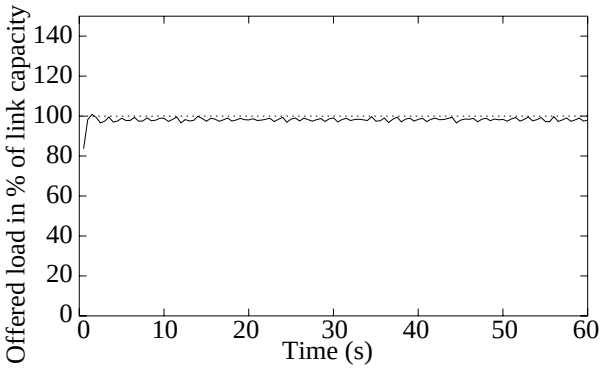


Figure 6: **Experiment 1: Offered Load.** The graph shows the offered load at Router 1.

flows for each source is as shown in Table 2. Class 1 traffic consists of on-off UDP flows, and the other classes consist of greedy TCP flows. All sources start transmitting packets with a fixed size of 1024 Bytes at time $t = 0$ until the end of the experiments at $t = 60$ seconds. Traffic is generated using the *netperf* v2.1pl3 tool [3]. The network load is initially zero and quickly ramps up to generate an overload at the bottleneck link of Figure 5. Congestion control at the TCP sources then maintains the total load at a level of about 99% of the link capacity, as shown in Figure 6.

We measure the delay, the loss rate, and the throughput of each traffic class at the output link of Router 1, which is the bottleneck link. Delays are measured as the waiting time of a packet in the transmission queue, i.e., as the difference of the times read of the machine clock when the packet enters and departs the transmission queue. Throughput and loss rates are obtained from reports generated every 0.5 sec by the OS kernel. In the plots, which summarize our measurements, we depict delay measurements of individual packets. Measurement of delay ratios, loss rates, ratios of loss rates and throughput are shown as averages over a sliding window of size 0.5 sec.

In Figure 7, we present our measurements of the service received at the bottleneck link. Fig. 7(a) depicts the ratios of the delays of Classes 4 and 3, and the delays of Classes 3 and 2. The plots show that the target value of $k_2 = k_3 = 2$ (from Table 1) is achieved. The plots indicate that the delay feedback loops appear to be stable, despite the simplified model we used for determining $K(n)$ in Section 3.

In Fig. 7(b) we show the delay of Class-1 packets at Router 1. The delay bound of $d_1 = 3$ ms is satisfied, with few ($< 1\%$) exceptions at times when it is not possible to satisfy simultaneously absolute loss and delay guarantees; as discussed in Section 5, such a conflict is resolved by giving precedence to the loss guarantee. Note that even if delay bounds are violated, no class-1 packet experiences a delay which exceeds 4 ms. Delay values, averaged over sliding windows of size 0.5 s, of other classes are shown in Fig. 7(c) and are in the range 10-40 ms.

In Figs. 7(d) and (e), we show the measurements of the loss rates. Fig. 7(d) depicts the ratios of loss rates for Classes 4 and 3, and for Classes 3 and 2. The desired ratios of $k_2' = k_3' = 2$ are maintained most of the time. As Fig. 7(e) indicates, the bound on the loss rates for Class 1 of $L_1 = 0.1$ % is always kept (Recall that we give highest precedence to absolute loss guarantees.) We also note that the maximum loss rate of classes 2–4 is below 1% over the entire experiment. Since all TCP sources start transmitting at the same time, i.e., the TCP flows are synchronized, we observe a transient effect shortly after the beginning of the experiment, at time 0.5 s. We believe that this transient effect is due to the slow-start mechanism in TCP,

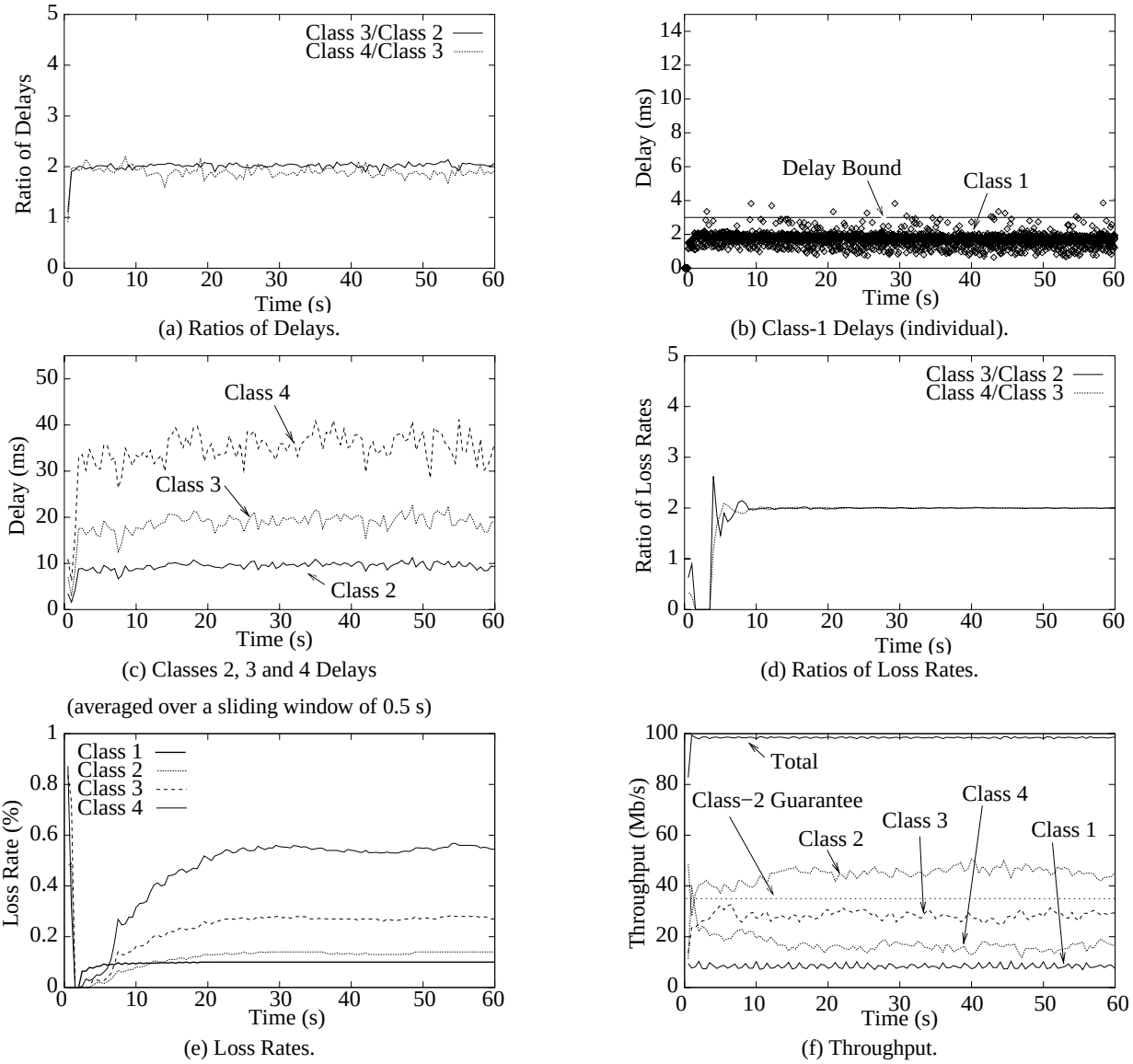


Figure 7: **Experiment 1.** The graphs show the service obtained by each class at the output link of Router 1.

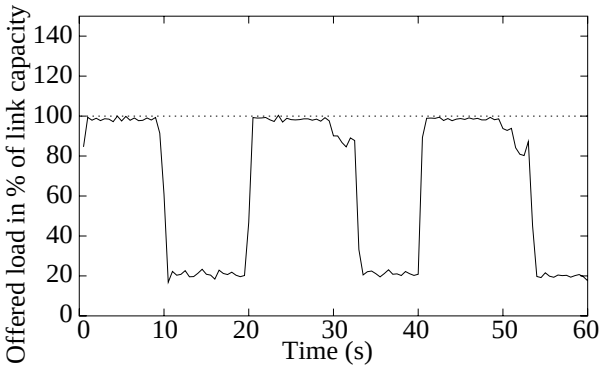


Figure 8: **Experiment 2: Offered Load.** The graph shows the offered load at Router 1.

Source	**Class**	**No. of flows**	**Type**	
			Protocol	Traffic
1	1	6	UDP	On-off
	2	10	TCP	Greedy/On-off
2	3	10	TCP	Greedy/On-off
	4	10	TCP	Greedy/On-off

Table 3: **Experiment 2: Traffic mix.** The on-off UDP source sends bursts of 25 packets during an on-period, and have a 150 ms off-period. TCP sources are greedy during time intervals $[0s, 10s]$, $[20s, 30s]$, and $[40s, 50s]$, and transmit chunks of 8 KBytes with a pause of 175 ms between each transmission during time intervals $[10, 20s]$, $[30, 40s]$, and $[50s, 60s]$. TCP sources run the *NewReno* congestion control algorithm.

which increases the sending rate of each source exponentially at the beginning of a sending period. With a network initially empty, all TCP flows suffer their first packet drops for at the same time.

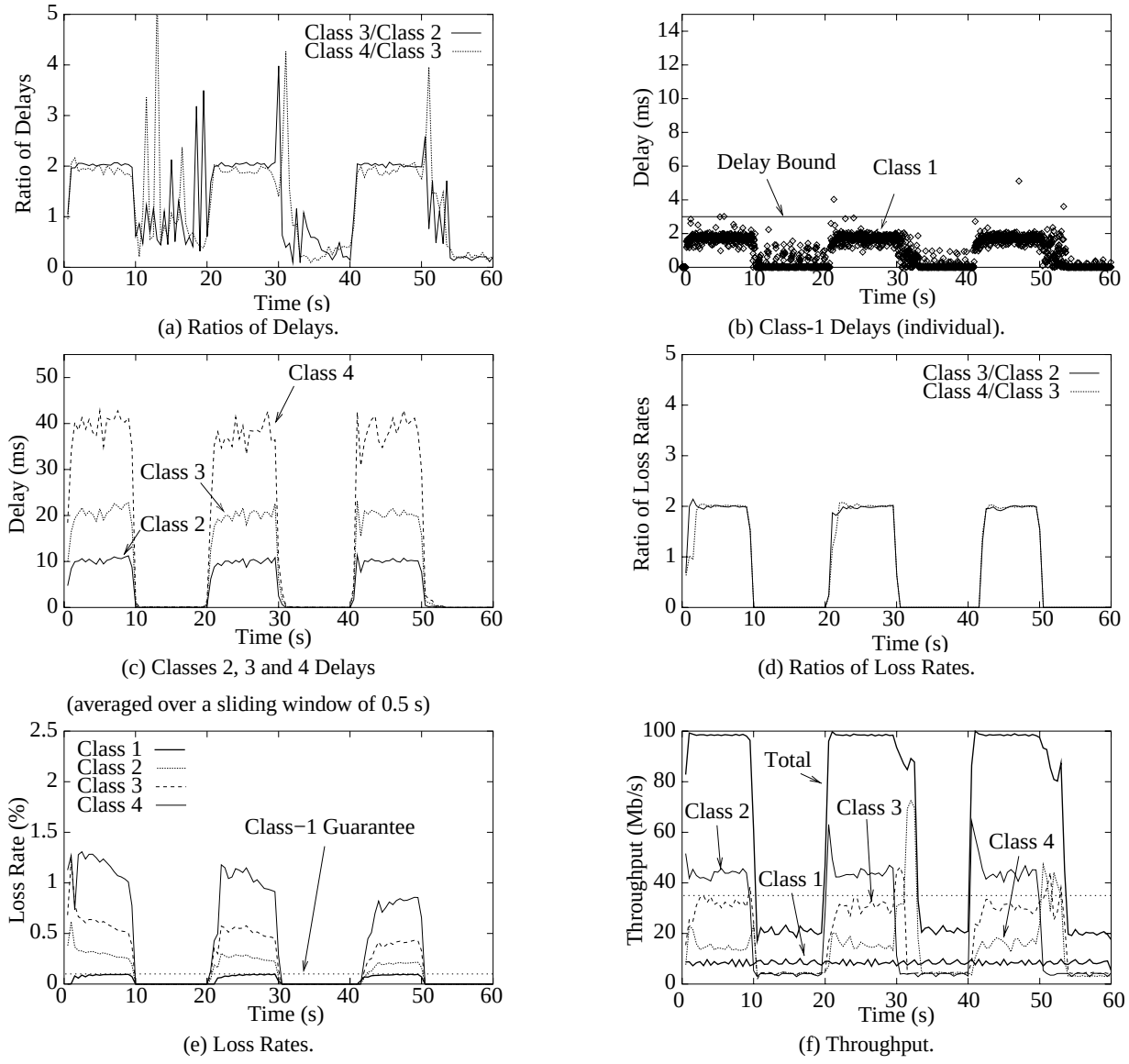
Finally, in Fig. 7(f) we include the throughput measurements of all classes. We observe that the rate guarantee for Class 2 of $\mu_2 = 35$ Mbps is maintained. The total throughput of all classes, labeled in Fig. 7(f) as 'Total', is close to the link capacity of 100 Mbps at each router.

6.2 Experiment 2: Single-node topology, highly variable load

The second experiment uses the same network topology, buffer size at routers, and service guarantees as in Experiment 1. Thus, Table 1 and Figure 5 apply to Experiment 2. The difference between Experiments 1 and 2 consists in the traffic generation of TCP flows. Instead of using greedy TCP sources over the whole experiment, we configured the TCP sources to be greedy during time intervals $[0s, 10s]$, $[20s, 30s]$ and $[40s, 50s]$. In the remaining time intervals, the TCP sources send chunks of 8KB of data and pause for 175 ms between the transmission of each chunk. We summarize the traffic mix for Experiment 2 in Table 3. This modification to the behavior of the TCP sources results in an highly variable offered load at Router 1, which we present in Figure 8.

Similar to Experiment 1, we measure the delay, the loss rate, and the throughput of each traffic class at the bottleneck link and present our results on Figure 9.

In Fig. 9(a), we present the ratios of the delays of Classes 4 and 3, and the delays of Classes 3 and 2.



(a) Ratios of Delays.

(b) Class-1 Delays (individual).

(c) Classes 2, 3 and 4 Delays

(averaged over a sliding window of 0.5 s)

(d) Ratios of Loss Rates.

(e) Loss Rates.

(f) Throughput.

Figure 9: **Experiment 2.** The graphs show the service obtained by each class at the output link of Router 1.

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

Table 4: **Experiments 3 and 4: Service guarantees.** The guarantees are identical at each router.

We observe that when the load is high, in time intervals $[0s, 10s]$, $[20s, 30s]$, and $[40s, 50s]$, the target value of $k_2 = k_3 = 2$ is achieved. Conversely, when the load is low, we observe oscillations in the ratios of delays. These oscillations do not characterize an unstable feedback loop, but come from the work-conserving constraint. As can be seen in Figs. 9(b) and (c), the delays of all classes are close to zero: if a packet arrives when the transmission queue is empty, it is forwarded immediately, regardless of the service guarantees. Given that the delays of all classes are extremely low during periods of underload, the delay ratios do not carry a lot of meaning during such periods. We also see that, at times $t = 0$, $t = 20$ and $t = 40$, when the load increases abruptly over a short period of time, the delay differentiation is realized almost immediately, which tends to show that the delay feedback loops are robust to rapid increases in the offered load. As was the case with Experiment 1, Fig. 9(b) shows that the absolute delay guarantee of Class 1, $d_1 = 3$ ms is enforced with a few exceptions at times when it is not possible to satisfy simultaneously absolute loss and delay guarantees. In Experiment 2, the delay bound violations occur for less than 0.15% of all Class-1 transmitted traffic.

In Figs. 9(d) and (e), we plot the measurements of the loss rates. We see that, in periods of packet drops, proportional loss guarantees ($k_2' = k_3' = 2$) and absolute loss guarantees ($L_1 = 0.1\%$) are satisfied. Thus, the algorithm for loss differentiation exhibits robustness to changes in the offered load.

We include the throughput measurements for all classes in Figure 9(f). The rate guarantee for Class 2 ($\mu_2 = 35$ Mbps) is maintained whenever Class 2 is sending at more than 35 Mbps, but cannot be satisfied during periods of underload. We also notice a quick increase in the transmission rate of Class 4 (and, to a lesser extent, in the transmission rate of Class 3) at times $t = 30$ and $t = 50$ and, less noticeably, at time $t = 10$. We offer the following explanation for this increase. *netperf* uses a request-response type of protocol at the application level [3]. When *netperf* generates on-off TCP traffic over a time interval T, the receiver estimates the total amount of data that it must receive by the end of the interval T. Due to the high delays, and relatively low throughput, encountered by Classes 3 and 4 during times of overload, the receiver is still waiting on some outstanding data at times $t = 30$ and $t = 50$ and does not notify the sender that the transmission is over. At these times, there is almost no Class 1 and 2 traffic in the routers anymore, and thus, the outstanding data is transmitted in about two seconds. This explanation is consistent with the *netperf* reports we observed, which indicated that the transmission of some Class 3 and 4 flows lasted almost 12 seconds instead of the 10 seconds we specified.

6.3 Experiment 3: Multiple node topology, near-constant load

We consider now a multiple node topology. We use a local network topology using point-to-point Ethernet links as shown in Figure 10. All links are full-duplex and have a capacity of $C = 100$ Mbps. Three PCs are

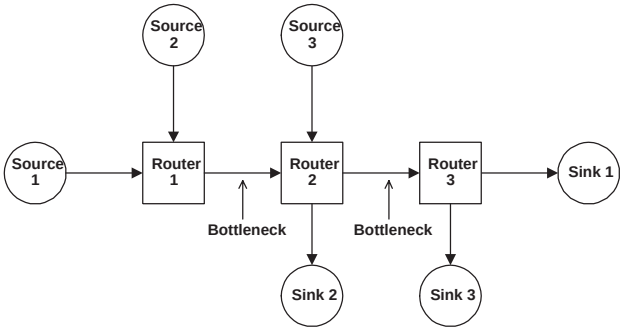


Figure 10: **Experiments 3 and 4: Network Topology.** All links have a capacity of 100 Mbps. We measure the service provided by Router 1 and 2 at the indicated bottleneck links.

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

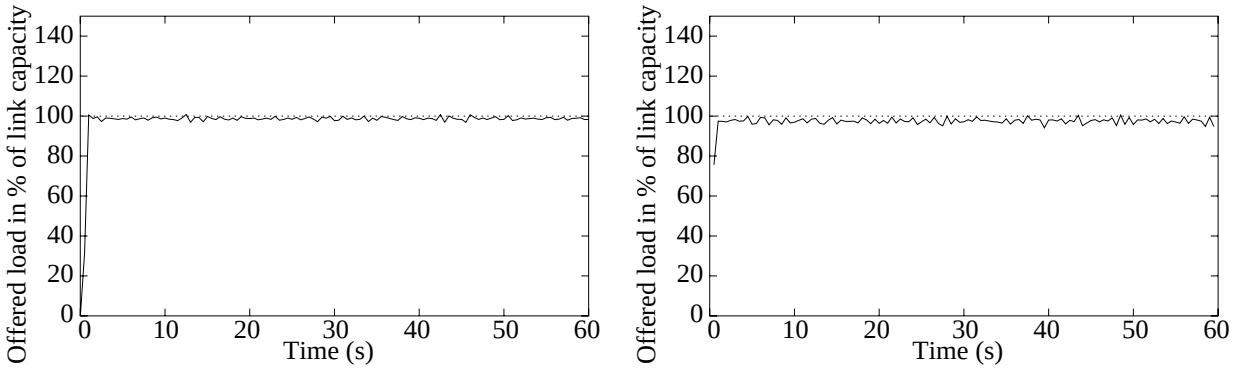
Table 5: **Experiment 3: Traffic mix.** The traffic mix is identical for each source-sink pair. The on-off UDP sources send bursts of 20 packets during an on-period, and have a 150 ms off-period. All TCP sources are greedy, i.e., they always have data to transmit, and run the *NewReno* congestion control algorithm.

set up as routers, indicated in Figure 10 as Router 1, 2 and 3. Other PCs are acting as sources and sinks of traffic. The topology has two bottlenecks: the link between Routers 1 and 2, and the link between Routers 2 and 3. As mentioned earlier, the buffer size at the output link of each router is set to $B = 200$ packets.

We consider four traffic classes with service guarantees as summarized in Table 4. The proportional service guarantees are the same as in Experiments 1 and 2, but the absolute delay and loss guarantees are different.

Sources 1, 2 and 3 send traffic to Sinks 1, 2 and 3, respectively. Different from Experiments 1 and 2, each source transmits traffic from all four classes. The traffic mix, the number of flows per class, and the characterization of the flows, is identical for each source, and as shown in Table 5. Each source transmits 6 flows from each of the classes. Class 1 traffic consists of on-off UDP flows, and the other classes consist of greedy TCP flows. All sources start transmitting packets with a fixed size of 1024 Bytes at time $t = 0$ until the end of the experiments at $t = 60$ seconds. The network load is initially zero and quickly ramps up to generate an overload at the bottleneck links of Figure 10. The offered load at both of Routers 1 and 2 is shown in Figure 11.

In Figures 12 and 13, we present our measurements of the service received at the bottleneck links of Routers 1 and 2, respectively. Figs. 12(a) and 13(a) depict the ratios of the delays of Classes 4 and 3, and the delays of Classes 3 and 2. The plots show that the target value of $k = 2$ (from Table 4) is achieved. The plots indicate that the delay feedback loops appear to be stable in the case of a multiple node topology. This result, coupled to the result we obtained in Experiment 1, also suggests that the delay feedback loops are



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

Figure 11: **Experiment 3: Offered Load.** The graphs show the offered load at Routers 1 and 2.

robust to changes in the network topology.

In Figs. 12(b) and 13(b) we show the delay of Class-1 packets at Router 1 and Router 2. The delay bound of $d_1 = 8$ ms is satisfied, with few ($< 1.5\%$) exceptions, due, again, to the precedence order we chose for our absolute guarantees. No class-1 packet ever experiences a delay higher than 10 ms at either Router 1 or 2. Figs. 12(c) and 13(c) indicate that delay values of other classes, averaged over sliding windows of size 0.5 s, are in the range 10-50 ms.

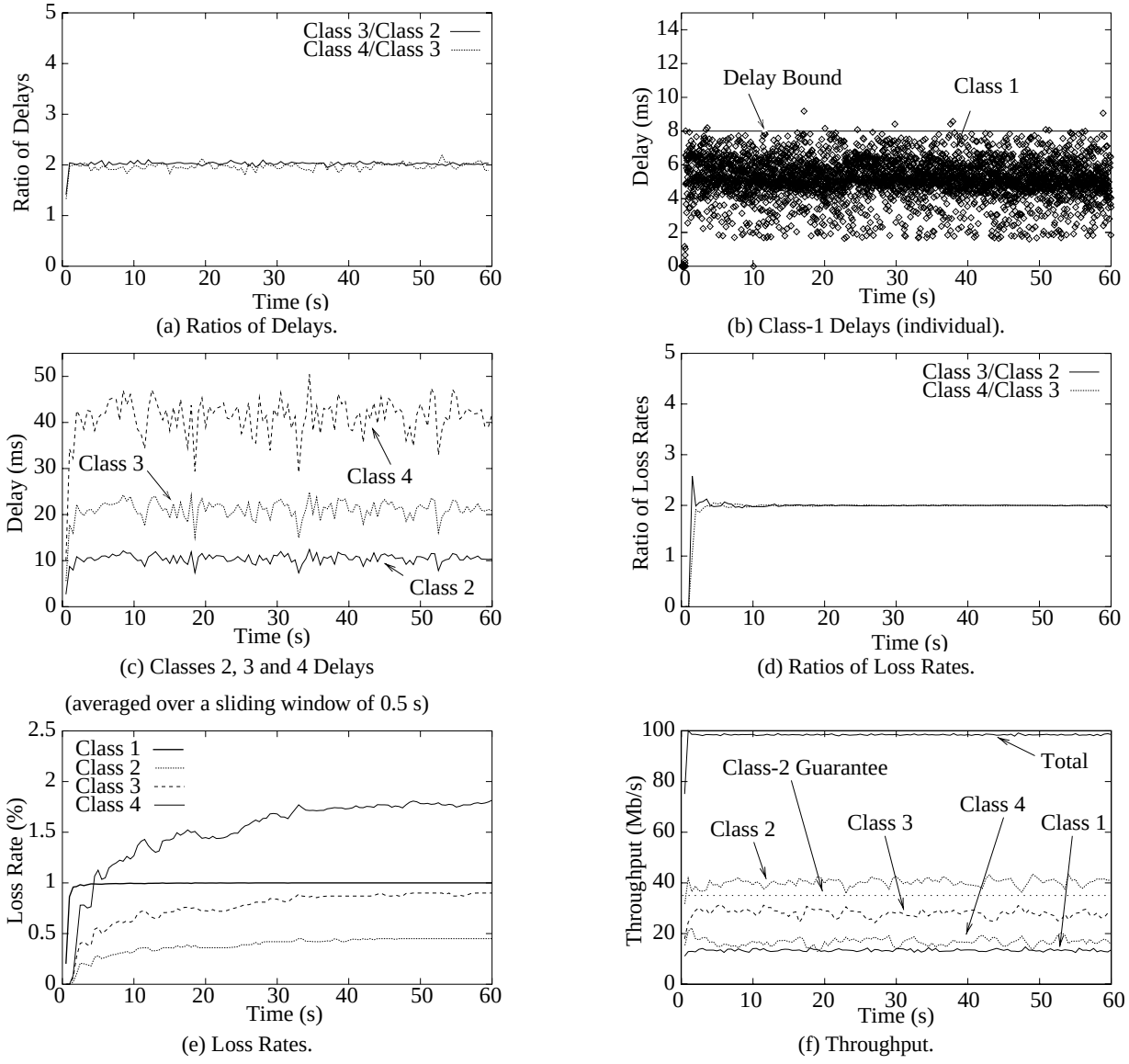
In Figs. 12(c) and (d), and Figs. 13(c) and (d), we show the measurements of the loss rates. Figs. 12(c) and 13(c) depict the ratios of loss rates for Classes 4 and 3, and for Classes 3 and 2. The desired ratios of $k_2' = k_3' = 2$ are maintained most of the time. As Figs. 12(d) and 13(d) indicate, the bound on the loss rates for Class 1 of $L_1 = 1$ % is always kept. We also see that, contrary to Experiments 1 and 2, the loss rate of Class 1 may be higher than the loss rate of other classes. This is result can be explained by the absence of proportional guarantees on Class 1, which excludes Class 1 from the ordering of Section 4. Our implementation always drop first from Class 1, until the loss bound L_1 as been reached, before using the ordering provided by proportional loss guarantees. Note that much less traffic is dropped at Router 2. This comes from the fact that Router 2 receives traffic from Source 3 and Router 1, instead of receiving traffic from two sources. Therefore, half of the traffic arriving at Router 2 has already been policed by Router 1.

Finally, in Figs. 12(e) and 13(e) we include the throughput measurements of all classes. We observe that the rate guarantee for Class 2 of $\mu_2 = 35$ Mbps is maintained. The total throughput of all classes, labeled in Figs. 12(e) and 13(e) as 'Total', is close to the link capacity of 100 Mbps at each router.

6.4 Experiment 4: Multiple node topology, highly variable load

For the sake of completeness, we run a fourth experiment, which uses the same network topology and service guarantees as Experiment 3, described in Table 4 and Figure 10. The difference between Experiments 3 and 4 consists in the traffic generation of TCP flows. Instead of greedy TCP sources, we use TCP sources which behave in the same manner as in Experiment 2, that is, which alternate between greedy transfers and on-off transfers. We summarize the traffic mix in Table 6 and obtain a variable load at each router, shown in Figs. 14(a) and (b).

We present the results of Experiment 4 in Figures 15 and 16. The measurements obtained confirm the results we obtained in the previous experiments. In Fig. 15(f) and 16(f) we note the same apparent problem



(a) Ratios of Delays.

(b) Class-1 Delays (individual).

(c) Classes 2, 3 and 4 Delays

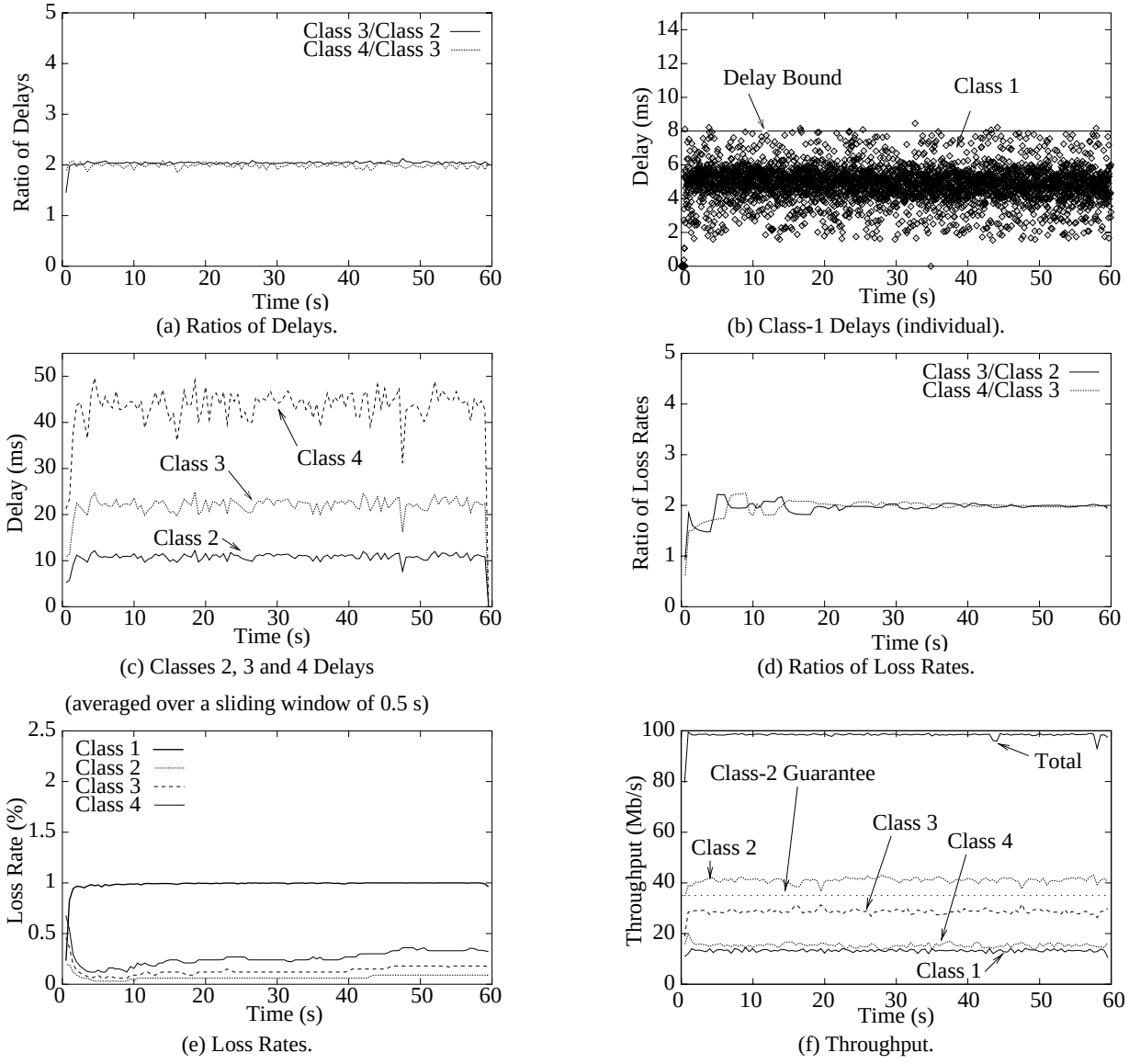
(averaged over a sliding window of 0.5 s)

(d) Ratios of Loss Rates.

(e) Loss Rates.

(f) Throughput.

Figure 12: **Experiment 3: Router 1.** The graphs show the service obtained by each class at the output link of Router 1.



(a) Ratios of Delays.

(b) Class-1 Delays (individual).

(c) Classes 2, 3 and 4 Delays

(averaged over a sliding window of 0.5 s)

(d) Ratios of Loss Rates.

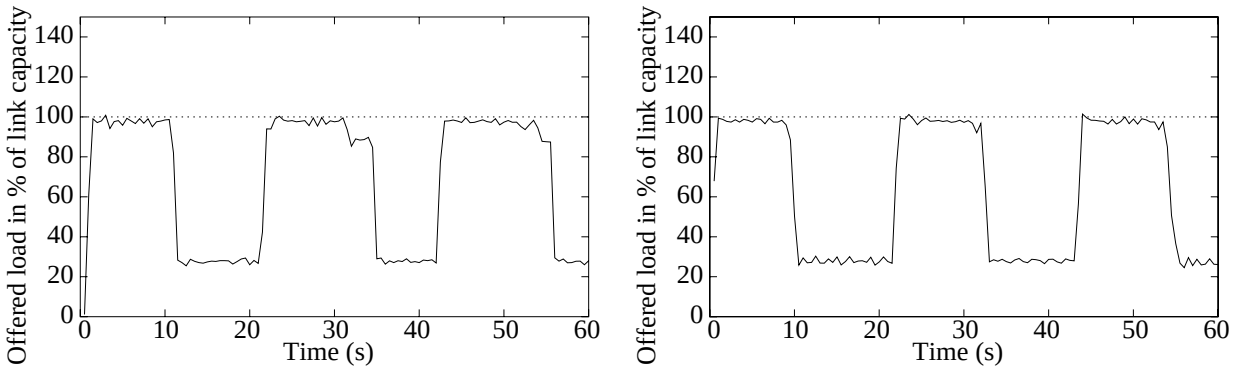
(e) Loss Rates.

(f) Throughput.

Figure 13: **Experiment 3: Router 2.** The graphs show the service obtained by each class at the output link of Router 2.

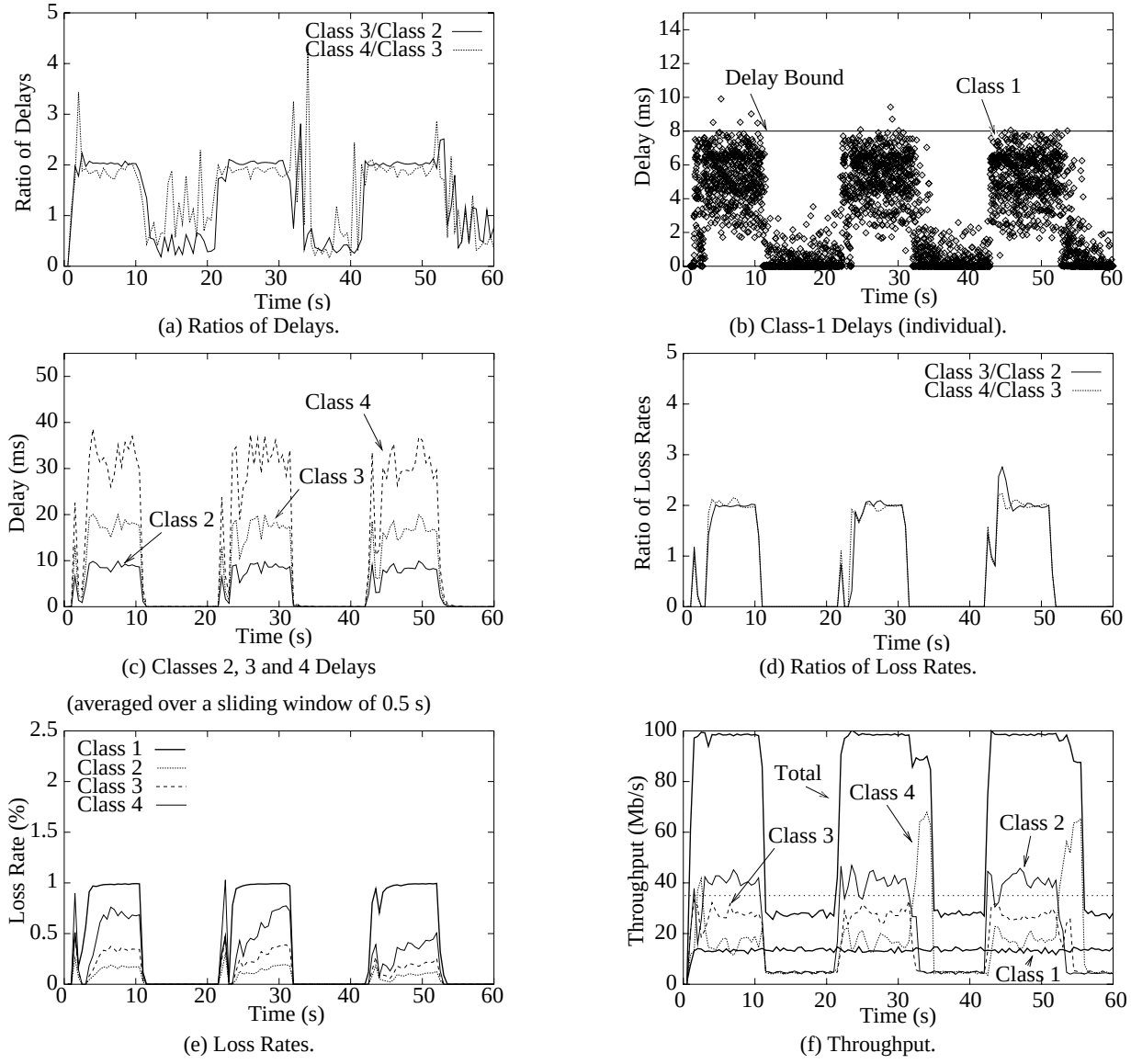
Class	No. of flows	Type	
		Protocol	Traffic
1	6	UDP	On-off
2	6	TCP	Greedy/On-off
3	6	TCP	Greedy/On-off
4	6	TCP	Greedy/On-off

Table 6: **Experiment 4: Traffic mix.** The traffic mix is identical for each source-sink pair. The on-off UDP sources send bursts of 20 packets during an on-period, and have a 150 ms off-period. TCP sources are greedy during time intervals $[0s, 10s]$, $[20s, 30s]$, and $[40s, 50s]$, and transmit chunks of 8 KBytes with a pause of 175 ms between each transmission during time intervals $[10, 20s]$, $[30, 40s]$, and $[50s, 60s]$. TCP sources run the *NewReno* congestion control algorithm.



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

Figure 14: **Experiment 4: Offered Load.** The graphs show the offered load at Routers 1 and 2.



(a) Ratios of Delays.

(b) Class-1 Delays (individual).

(c) Classes 2, 3 and 4 Delays

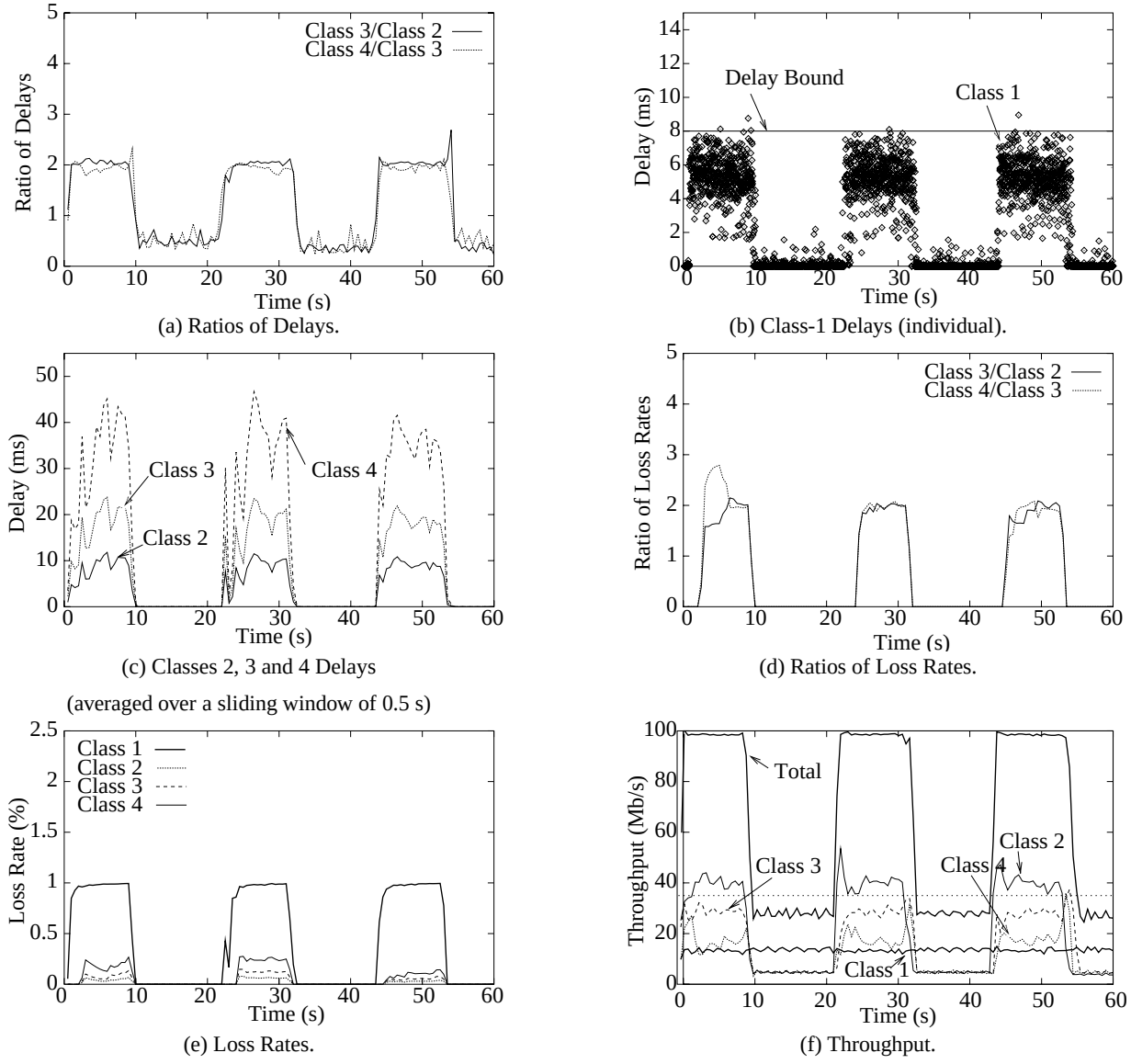
(averaged over a sliding window of 0.5 s)

(d) Ratios of Loss Rates.

(e) Loss Rates.

(f) Throughput.

Figure 15: **Experiment 4: Router 1.** The graphs show the service obtained by each class at the output link of Router 1.



(a) Ratios of Delays.

(b) Class-1 Delays (individual).

(c) Classes 2, 3 and 4 Delays

(averaged over a sliding window of 0.5 s)

(d) Ratios of Loss Rates.

(e) Loss Rates.

(f) Throughput.

Figure 16: **Experiment 4: Router 2.** The graphs show the service obtained by each class at the output link of Router 2.

Set	Enqueue		Dequeue		Pred.
	$\overline{X}$	s	$\overline{X}$	s	λ_{pred} (Mbps)
1	15347	2603	4053	912	186
2	11849	2798	3580	970	234
3	2671	1101	3811	826	557
4	2415	837	3810	858	580

Table 7: **Overhead Measurements.** This table presents, for the four considered sets of service guarantees, the average number of cycles ($\overline{X}$) consumed by the enqueue and dequeue operations, the standard deviation (s), and the predicted throughput λ_{pred} (in Mbps) that can be achieved. In the 1 GHz PCs we use, one cycle corresponds to one nanosecond.

with the *netperf* sources as we did in Experiment 2. An oddity occurs at time $t \approx 35$ s, where the throughput of Router 1 seems to be greater than the offered load at Router 2. In fact, we monitor the offered load at the output link of Router 2, but the output link of Router 1 is connected to an *input* link of Router 2, which we do not monitor. By inspecting reports from the OS kernel of Router 2, we saw that the "receive" buffer of the input link of Router 2 overflowed at $t \approx 35$ s, and that packets were discarded at the input link, which explains this anomaly.

In summary, the measurement experiments of a network with single or multiple bottlenecks, constant or variable load, show that our feedback algorithms achieve the desired service differentiation, and utilize the entire available bandwidth, while maintaining stability throughout and are robust to changes in the network topology, the traffic mix or the service guarantees.

6.5 Analysis of Overhead

In Subsections 6.1–6.4, we saw that our implementation can fully utilize the capacity of a 100 Mbps link, without overloading the PC router. We next present a detailed analysis of the overhead of our implementation, where we attempt to predict the data rates that can be supported by our PC router implementation, and where we measure the sensitivity of our implementation to the number of service constraints. We will show measurements of the enqueue and dequeue operations for four different sets of service guarantees, tested for four traffic classes.

Set 1: Same as Table 4.
Set 2: Set 1 with absolute guarantees from Set 1 removed.
Set 3: Set 2 with proportional guarantees from Set 1 removed.
Set 4: No service guarantees.

In the measurements we determine the number of cycles consumed for the enqueue and dequeue procedures. The timestamp counter register of the Pentium processor is read at the beginning and at the end of the procedures, for each execution of the procedure.

We compiled our implementation with a code optimizer, in our case, we use the *gcc* v2.95.3 compiler with the "-O2" flag set. The results of our measurements are presented in Table 7, where we include the

machine cycles consumed by the enqueue and dequeue operations. The measurements are averages of over 500,000 datagram transmissions on a heavily loaded link, using the same topology and traffic pattern as in Subsection 6.3. The measurements in Table 7 were collected at Router 1. Measurements collected at Router 2 showed deviations of no more than $\pm 5\%$ compared to Router 1.

Since the enqueue and dequeue operations is invoked once for each IP datagram, we can predict the maximum throughput of a PC router with

$$\lambda_{pred} = \frac{F}{\alpha + \beta} \cdot \overline{P}\,, \tag{72}$$

where F denotes the CPU clock frequency in Hz, α denotes the number of cycles consumed by the enqueue operation, β denotes the number of cycles consumed by the dequeue operation, and $\overline{P}$ is the average size of a datagram. In the case of our implementation on 1 GHz PCs, we have $F = 10^9$. Data from a recent report [4] indicates that the average size of an IP datagram on the Internet is $\overline{P} = 451.11$ bytes. Using these values for $\overline{P}$ and F in Eqn. (72) shows that, in the four sets of constraints considered, we estimate that our implementation can be run at data rates of at least 186 Mbps.

We next evaluate the sensitivity of the performance as a function of the number of constraints. Note from Section 4 that the number of cycles consumed by the dequeue operation is independent of the set of constraints. From Table 7, we see that the overhead associated to the absolute service guarantees (Set 3) is of approximately 10% compared to a set with no service guarantees (Set 4). The overhead is 29% when comparing a set with absolute and proportional service guarantees (Set 1), to a set with proportional guarantees only (Set 2). We thus see that the overhead incurred by absolute constraints is dependent on the presence of proportional guarantees, but remains relatively low. Proportional guarantees seem to incur more overhead than absolute guarantees. However, in the set of constraints we consider, there is a larger number of classes with proportional guarantees than classes with absolute guarantees, and thus, more computations are needed to enforce proportional guarantees.

7 Conclusions

We presented the Quantitative Assured Forwarding service, which provides proportional differentiation on loss and delay and absolute service guarantees on loss, throughput and delay for classes of traffic. We proposed a feedback based algorithm for realizing the Quantitative Assured Forwarding service at a router. The algorithm does not require prior knowledge of traffic arrivals, and does not rely on signaling. At times when not all absolute service guarantees can be satisfied simultaneously, the algorithm relaxes some of the guarantees by using a priority order. The algorithm has been implemented in FreeBSD PC-routers. Through experiments in a network of PC-routers, we showed that the proposed algorithm could fully utilize the available capacity of 100 Mbps. The measurements showed that the service guarantees of the Quantitative Assured Forwarding service are enforced. In future work, we will extend the approach presented in this paper to TCP congestion control mechanisms [5, 12, 17, 18]. We are also considering an implementation of our algorithm in programmable gigabit routers, such as the Intel IXP1200 [2].

References

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

[2] Intel's IXP 1200 network processor. http://developer.intel.com/design/network/products/npfamily/ixp1200.htm.

[3] The public *netperf* homepage. http://www.netperf.org.

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

[5] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. IETF RFC 2581, April 1999.

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

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

[8] K. Cho. A framework for alternate queueing: towards traffic management by PC-UNIX based routers. In *Proceedings of USENIX '98 Annual Technical Conference*, New Orleans, LA, June 1998.

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

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

[11] 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.

[12] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. IETF RFC 2582, April 1999.

[13] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital control of dynamic systems.* Addison-Wesley, Menlo Park, CA, 3rd edition, 1998.

[14] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group, June 1999. IETF RFC 2597.

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

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

[17] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, August 1988. Stanford, CA.

[18] V. Jacobson. Modified TCP congestion avoidance algorithm. Note sent to end2end-interest mailing list, April 1990. ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail.

[19] R. R.-F. Liao and A. T. Campbell. Dynamic core provisioning for quantitative differentiated service. In *Proceedings of IWQoS 2001*, pages 9–26, Karlsruhe, Germany, June 2001.

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

[21] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real Time Systems*, 2001. Special Issue on Control-Theoretical Approaches to Real-Time Computing. To appear.

[22] Y. Lu, A. Saxena, and T. Abdelzaher. Differentiated caching services; A control-theoretical approach. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 615–624, Phoenix, AZ, April 2001.

[23] A.M. Lyapunov. The general problem of motion stability, 1892.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] S. Parekh, N. Gandhi, J.L. Hellerstein, D. Tilbury, T.S. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Journal of Real-Time Systems*, 2001. Special Issue on Control-Theoretical Approaches to Real-Time Computing. To appear.

[29] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, June 1996.

[30] J.-J. E. Slotine and W. Li. *Applied nonlinear control*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[31] A. Striegel and G. Manimaran. Packet scheduling with delay and loss differentiation. *Computer Communications*, 2001. To appear.

[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.