Marking Algorithms for Service Differentiation of TCP Traffic [*]

Technical Report: University of Virginia, CS-2003-04

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

Abstract

Class-based service architectures for quality-of-service (QoS) differentiation typically provide loss, throughput, and delay differentiation. However, proposals for class-based QoS differentiation generally do not account for the needs of TCP traffic, which are characterized by a coupling of packet losses and achievable throughput. Ignoring this coupling may result in poor service differentiation at the microflow level. This paper shows how Explicit Congestion Notification (ECN) can be used to achieve service differentiation for TCP traffic classes at the microflow level. We present a traffic marking algorithm for routers, which, if used in conjunction with ECN, regulates the transmission rate of TCP sources in such a way that packet drops due to buffer overflows are avoided. We demonstrate how the algorithm can be integrated in a service architecture with absolute and proportional QoS guarantees. Simulation results illustrate the effectiveness of the presented algorithms at avoiding packet losses and regulating traffic for meeting service guarantees, and provide a comparison with other algorithms proposed in the literature.

Key Words: TCP, ECN, QoS, Packet Losses, Congestion Control.

[*] This work is supported in part by the National Science Foundation through grants ANI-9730103 and ANI-0085955.

1 Introduction

Since the late 1990s, a significant amount of research, e.g., [5, 6, 8, 22, 28], has been devoted to devising simple and scalable architectures for quality-of-service (QoS) differentiation between classes of traffic. These proposals, mainly formulated in the context of the Differentiated Services (*DiffServ*, [5]) framework, generally provide loss, delay and throughput differentiation between classes of traffic.

However, the scheduling and/or buffer management algorithms involved usually do not take into account the sensitivity of TCP traffic to losses. TCP traffic, which accounts for more than 90% of the total traffic on the Internet [7], is a feedback-driven protocol that uses losses as an indicator for congestion avoidance and control [3, 23, 24]. Hence, TCP packet losses lead to significant performance degradation of the throughput of TCP sources. Furthermore, due to the relatively complex relationship between packet losses and TCP throughput [29] and the lack of discriminating mechanisms between flows belonging to the same service class, quantitative loss differentiation on traffic aggregates can result in unpredictable throughput differentiation between individual TCP flows.

In an effort to reduce losses in TCP/IP networks, Explicit Congestion Notification (ECN, [32]) has been proposed as an additional congestion signal for TCP flows. ECN allows to mark packets with a Congestion Experienced (CE) codepoint. When a packet marked with the CE codepoint is received by its destination, the data is acknowledged with a packet containing the CE-ECHO codepoint. When the CE-ECHO marked acknowledgment reaches the sender, the sender reduces its throughput, as if a loss had happened in the network.

The emergence of ECN has stimulated research on appropriate marking algorithms that indicate congestion to TCP sources to avoid packet losses resulting from buffer overflows [4, 14, 15, 16, 18, 20, 21, 26, 30]. The key idea behind these algorithms is to mark packets proactively, that is, before congestion occurs, to limit the amount of lost traffic in the network. For instance, RED [18] and its extensions, e.g., [26], use a smoothed average of the buffer occupancy, $\overline{Q}$, to infer impending congestion. If $\overline{Q}$ is between two thresholds min_{TH} and max_{TH}, packets are marked with a probability increasing in $\overline{Q}$. Packets are only dropped if $\overline{Q} > max_{TH}$. Other algorithms, e.g., Blue [14] or Stochastic Fair Blue [15] use different factors such as link utilization to infer congestion, or, in the case of Random Exponential Marking (REM, [4]), a price, depending on the queue occupancy and the difference between aggregate input and output rates. The PI algorithm [21] uses a feedback-based model for TCP arrival rates [27] to let the buffer occupancy converge to a target value, but assumes a priori knowledge of the round-trip times and of the number of flows traversing the router.

While all of the proactive marking algorithms discussed above can, to some extent, reduce the amount of losses in the network, this paper tries to address a broader question. Since ECN provides congestion signals that can be conveyed before any traffic is dropped, we are exploring if ECN can be integrated with scheduling and buffer management into a service differentiation scheme for TCP traffic.

We first explore if it is feasible to devise a marking algorithm which can ensure that the traffic load at a router remains at a level that entirely avoids losses due to buffer overflows at routers without wasting available network bandwidth. The basic idea is to anticipate the behavior of TCP sources at the routers, by tracking the window size and the round-trip time of flows at the router, and to use ECN marking to have the senders adjust the window size of the flows. More precisely, when a router predicts future losses, the router sends congestion signals to the sources via ECN with the goal of reducing the sources' sending rates before a loss occurs. To that effect, we present a reference marking algorithm that tracks and controls all TCP flows at a router to prevent impending buffer overflows. This reference algorithm can probably not be implemented in routers with the hardware currently available, due to the computational overhead of maintaining per-flow information for all flows, but is useful to assess the viability of our design. To address scalability issues, we note that in practice, only a small number of flows contribute to the majority of traffic [2, 12, 13]. We conjecture that tracking and marking only these ”heavy-hitters” is sufficient for avoiding packet drops. Based on this idea of filtering flows, we present a heuristic approximation of the reference algorithm that we expect to be computationally efficient enough to be deployed in edge routers, where congestion is most likely to occur [19]. Then, we examine if ECN can be used to concurrently pursue both objectives of avoiding losses and regulating traffic to meet per-class service guarantees.

This paper is organized as follows. In Section 2, we present the reference algorithm for avoiding buffer overflows. In Section 3, we describe the heuristic algorithm. In Section 4, we show how the proposed marking algorithms can be used for traffic regulation in the context of class-based service differentiation. We compare the performance of the reference and heuristic algorithms to other algorithms in Section 5, and draw brief conclusions in Section 6.

2 A Reference Marking Algorithm for Avoiding Losses

In this section, we describe a reference algorithm for marking TCP traffic at network routers. The objective of the algorithm is to determine when to mark TCP traffic and which flows to mark in order to completely avoid packet losses due to router buffer overflows, while maximizing the utilization of the network capacity.

Throughout this section, we assume that all traffic uses TCP. While in practice one can expect a mix of flows using different protocols (e.g., TCP, UDP, SCTP), we can make this assumption without loss of generality, since one can always reserve fixed resources at each router for TCP traffic such as a dedicated buffer and a fixed portion of the output link capacity. Furthermore, we assume that ECN is available in the network. This assumption is realistic for future networks, as ECN is being rapidly deployed on the Internet: relatively recent operating systems such as FreeBSD 4.5 or Linux 2.4 already support ECN. For the description of the reference algorithm in the remainder of this section, we assume that enough resources are available to perform the needed computations.

We next describe the algorithm at a single router, for a single greedy TCP flow, i.e., a TCP flow that always has data to transmit. For the time being, we assume that there is no other traffic in the network, and that the only cause of packet losses at the router is a buffer overflow. The router estimates the congestion window size and the round-trip time of the TCP flow. With these estimates, future traffic arrivals are projected, and impending buffer overflows are inferred, as illustrated in Fig. 1. In the case of Fig. 1, at time t, a buffer overflow is projected for time t'. If a packet loss is projected, the algorithm reduces the congestion window size of the TCP source by marking packets with ECN. By reducing the congestion window size, the sending rate of the TCP source is reduced, and impending packet losses can be avoided. Note that the proposed algorithm does not require any changes to TCP, and only relies on ECN to reduce the traffic load.

The remainder of this section describes the calculations at the router to project future packet losses. We explain how to use the projections to mark traffic and avoid packet losses using the simplified model of a

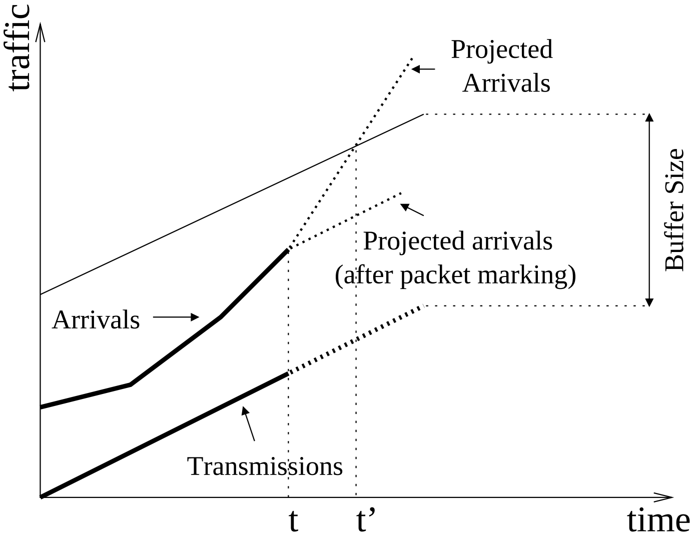


Figure 1: **Overview of the algorithm.** At time t of a packet arrival, the router projects future arrivals, by inferring how the TCP source will send traffic. When an impending buffer overflow is predicted, at time t' here, a packet is marked to reduce future arrivals.

single TCP flow. We then generalize the proposed technique to multiple TCP flows with different sources and destinations crossing paths at a same router.

2.1 Projecting Traffic Arrivals to Prevent Losses

Let us assume for now that packet losses can only be caused by a buffer overflow at the considered router, and let B_{lim} denote the size of the router's buffer. We refer to the input curve, $R^{in}(t)$, as the total amount of traffic that has entered the router until time t, excluding dropped traffic. We refer to the output curve, $R^{out}(t)$, as the total amount of traffic that has left the router until time t. At any time t, the backlog at the router is equal to $R^{in}(t) - R^{out}(t)$. Hence, we have the following constraint:

$$\forall t: \; R^{in}(t) - R^{out}(t) \leq B_{lim} \; . \tag{1}$$

Assume that the output link capacity of the router has a constant rate C, and that the router uses a work-conserving scheduler. Thus, for any t and $\tau > 0$ such that traffic is always backlogged over $[t, t+\tau]$,

$$R^{out}(t+\tau) = R^{out}(t) + C \cdot \tau \; . \tag{2}$$

Since Eqn. (2) characterizes R^{out} whenever there is a backlog, the algorithm only needs to infer $R^{in}(t+\tau)$ for $\tau > 0$, to ensure Eqn. (1) holds at $t+\tau$, thereby avoiding impending buffer overflows. To clearly distinguish between known, measured values and future, *projected* values of the arrivals and of the departures, we define $\widetilde{R}^{in}_t(t+\tau)$ as the value projected at time t for the input curve at time $t+\tau$.

To project future arrivals $\widetilde{R}^{in}_t(t+\tau)$ for $\tau > 0$, we need to examine how traffic is sent at the source, so that we can infer how much traffic is received by the router. For this discussion, we consider "segments" and "packets" as synonymous. Furthermore, we ignore the slow-start phase of TCP, since the flow is unlikely to send enough traffic to create a buffer overflow during slow-start, and only focus on the congestion avoidance phase. Every time an acknowledgment is received at the source, the source sends a number of packets equal to the maximum of the receiver's advertised window size, $adv(t)$, and the source's congestion window size, $cwnd(t)$, [1] minus the number of packets sent and not yet acknowledged. $cwnd(t)$ is increased by $\frac{1}{cwnd(t)}$ every time an acknowledgment is received, unless the acknowledgment is marked with the CE-ECHO codepoint or a packet drop is inferred by reception of a triple-duplicate acknowledgment, in which case $cwnd(t)$ is decreased to $\frac{cwnd(t)}{2}$. Last, if the retransmission timer of the TCP source expires, $cwnd(t)$ is reset to one and the flow is back to slow-start.

Since $cwnd(t)$ is conditioned by receiving acknowledgments at the source, the round-trip time (RTT), that is, the time difference between a packet is sent and its acknowledgment is received at the source, is central to the evolution of $cwnd(t)$. The RTT depends on time, due to variable queueing delays, and/or changing routes. We denote by $RTT(t)$ the value of the RTT at time t, and define a series of "rounds" as follows. The first round starts when the first packet is sent by the source, and ends when the acknowledgment to the first packet sent in the first round is received. The $(k+1)$-th round starts immediately after the k-th round ends. Therefore, denoting by s_k the start time of the k-th round at the source, the s_i are linked by the recursive equation

$$s_{i+1} = s_i + RTT(s_i) \ .$$

Now, within the i-th round, i.e., between times s_i and s_{i+1}, a TCP source sends at most $W(s_i)$ packets with $W(s_i) = \max\{adv(s_i), cwnd(s_i)\}$. Furthermore, it can be shown (see [29], or the example in [33], Chap. 21) that, in absence of retransmission timer timeouts, and if the TCP source is not in slow-start mode, $W(s_{i+1})$, the number of packets sent in the $(i+1)$-th round is bounded by

$$\frac{1}{2}W(s_i) \leq W(s_{i+1}) \leq W(s_i) + 1 \ . \tag{3}$$

The lower bound is given by the fact that at most one ECN congestion signal is taken into account per round [32], while the upper bound is reached only if all packets sent in the i-th round are successfully acknowledged by the destination. Note that Eqn. (3) is general enough to capture the behavior of Delayed-ACKs implementations, which issue on AVErage only one acknowledgment for each two data packets.

Since $W(t)$ and $RTT(t)$ are not known by a router, Eqn. (3) tells us that a router that wants to estimate future traffic arrivals must be able to estimate, at any time t, $RTT(t)$, $W(t)$, and s_i for the current round. We denote by $\widehat{W}(t)$, $\widehat{RTT}(t)$, and $\widehat{s}(t)$ the estimates at the considered router of $W(t)$, $RTT(t)$, and of s_i, respectively.

These estimates are computed as follows. The first time a packet is received at the router, the current time, T_1, is recorded. When the second packet arrives at the router, at time T_2, the value of $\widehat{RTT}(t)$ is initialized to $T_2 - T_1$,[2] and $\widehat{W}(t)$ is initialized to 1. At time T_2, $\widehat{s}(t)$ is initialized to T_2.

After time T_2, the key idea to update the RTT estimates is to discriminate the rounds. Measurement studies [11, 31] show that the RTT is generally significantly larger than the time needed to receive all

[1] At the end hosts, $cwnd(t)$ is internally expressed in bytes, which does not affect our present discussion.

[2] This method is equivalent to the SYN-ACK algorithm of [25].

packets from a given round.[3] Thus, monitoring the packets' interarrival times at the router can determine alone if a new round has started. More specifically, if, for a constant $K > 1$, T_{i-1} and T_i satisfy

$$T_i - T_{i-1} > \frac{\widehat{RTT}(T_{i-1})}{K} \ , \tag{4}$$

the router considers that T_i marks the start of a new round.

If Eqn. (4) does not hold, T_{i-1} and T_i are part of the same round, and $\widehat{RTT}(t)$ is set equal to $\widehat{RTT}(T_{i-1})$, $\widehat{W}(t)$ is set to $\widehat{W}(T_{i-1})+1$, and $\widehat{s}(t)$ is set equal to $\widehat{s}(T_{i-1})$. Conversely, if Eqn. (4) holds, the router updates the estimates as follows:

$$\begin{aligned}
\widehat{s}(t) &= T_i \ , \\
\widehat{W}(t) &= 1 \ , \\
\widehat{RTT}(t) &= \alpha \cdot \widehat{RTT}(\widehat{s}(T_{i-1})) + (1-\alpha) \cdot (\widehat{s}(t) - \widehat{s}(T_{i-1})) \ ,
\end{aligned}$$

where $0 \leq \alpha \leq 1$ is a constant. The round-trip time estimator used at the TCP sources uses $\alpha = 0.9$, which has shown to provide reasonably accurate results [23]. We point out that except in rare cases of persistent link failure, where packets end up being re-routed, the RTT does not vary significantly over time, and thus, the algorithm should be rather insensitive to the selection of α.

With the estimates of the RTT and the window size, the router can project future window sizes. Specifically, for any time t and any time $\tau > 0$, denoting by $\widetilde{W}_t(t+\tau)$ the projection of the window size at time $t+\tau$, the router computes $\widetilde{W}_t(t+\tau)$ as

$$\widetilde{W}_t(t+\tau) = \begin{cases}
\widehat{W}(t) & \text{if } t+\tau < \widehat{s}(t) + \widehat{RTT}(t), \\
\widehat{W}(t)+1 & \text{if } t+\tau \geq \widehat{s}(t) + \widehat{RTT}(t) \\
 & \text{and no packet has been} \\
 & \text{marked (or dropped)} \\
 & \text{in } [\widehat{s}(t), t], \\
\frac{1}{2}\widehat{W}(t) & \text{if } t+\tau \geq \widehat{s}(t) + \widehat{RTT}(t) \\
 & \text{and at least one packet} \\
 & \text{has been marked (or} \\
 & \text{dropped) in } [\widehat{s}(t), t].
\end{cases} \tag{5}$$

The router can discover if a packet has been marked (or dropped upstream) in $[\widehat{s}(t), t]$ by checking the ECN bits and the TCP sequence numbers. From Eqn. (5), the router projects that the window size does not change until the end of the current round, and that its value at the beginning of the next round depends on whether or not a packet has been dropped or marked during the current round. Thus, Eqn. (5) captures the fact that, at the earliest, ECN signals have an effect only at the beginning of the *next* round.

We shall note that this projection is correct only when all packets in a round have been received by the router. This may seem a restriction, but since the RTT is generally much larger than the time needed to receive all packets in a given round [11, 29, 31], the projection is generally accurate. With $\widetilde{W}_t(t+\tau)$ given by Eqn. (5), a router can project the input curve with the following expression:

$$\widetilde{R}_t^{in}(t+\tau) = R^{in}(t) + MSS \cdot \gamma_t(\tau) \cdot \widetilde{W}_t(t+\tau) \ , \tag{6}$$

[3] The same assumption is used in [29] for modeling the sending rate of a TCP source, and has been confirmed in experimental measurements.

where MSS is the maximum segment size of the TCP flow, and

$$\gamma_t(\tau) = \begin{cases} 1 & \text{if } t + \tau \geq \widehat{s}(t) + \widehat{RTT}(t), \\ 0 & \text{otherwise.} \end{cases}$$

That is, a router can assume that all traffic sent in the next round arrives in a batch right at the start of the next round. This projection thus assumes a ”worst-case scenario.” In practice, such bursts of traffic are rarely observed.

Next, we discuss the marking algorithm. To determine if an arrival at time t must be marked, a router checks that the flow has not already been marked (or has experienced some losses) during the current round. This test is necessary since at most one ECN-marked packet per round has an impact on the arrivals. If the flow has not experienced any losses or packet marking during $[\widehat{s}(t), t]$, the router verifies if the following condition holds:

$$\widetilde{R}_t^{in}(t + \tau) - R^{out}(t) - C \cdot \tau \leq B_{lim} \; . \tag{7}$$

This condition tests if a buffer overflow is going to occur at he beginning of the next round. Since ECN feedback does not have any impact until the beginning of the next round, the condition in Eqn. (7) is checked for $\tau = \widehat{s}(t) + \widehat{RTT}(t) - t$. If the condition of Eqn. (7) is violated, then the router marks the packet at the head of the transmission queue with the CE codepoint. Marking the packet at the head of the queue minimizes the delay needed for the ECN feedback to reach the source.

We conclude with a discussion on the robustness of the above estimators. If the constant K in Eqn. (4) is too small (e.g., $K < 1$), or if $W(t)$ is extremely large and data transmission appears continuous, the test described in Eqn. (4) may not be able to discriminate between rounds. In the worst-case, the router may never infer the start of a new round, and $\widehat{W}$ grows unbounded. To address this problem, we use a safeguard, based on Eqn. (3) as follows. If, at time T_i, we have[4]

$$\widehat{W}(T_i) > \widehat{W}(\widehat{s}(T_i)^-) + 1 \; ,$$

the router infers that T_i marks the start of a new round, *even if Eqn. (4) does not hold*. Now, if K is too large (e.g., $K > 1000$), the router incorrectly infers that each packet arrival marks the start of a new round, and thus, $\widehat{W}$ and $\widehat{RTT}$ underestimate W and RTT. In the worst-case, when $\widehat{W} \rightarrow 0$ and $\widehat{RTT} \rightarrow 0$, no projection is performed, thus no traffic is marked, and the algorithm degenerates to Drop-Tail. Our experiments show that the algorithm is quite robust to changes of the parameters. In fact, the experimental results gathered in Section 5 with $K = 10$, $\alpha = 0.9$ are almost identical to those obtained with any value $10 \leq K < 100$, and $0.7 < \alpha < 1$.

2.2 Generalization to Multiple TCP Flows

We next consider a more general situation with N greedy TCP flows. We use $\widehat{RTT}_i(t)$, $\widehat{W}_i(t)$, MSS_i, and $\widehat{s}_i(t)$ to denote the estimated round-trip time, congestion window size, maximum segment size and start time of the current round for TCP flow i, respectively. Let us assume, for the moment, that the router is able to monitor all N TCP flows and can keep track of all the $\widehat{RTT}_i(t)$, $\widehat{W}_i(t)$, MSS_i and $\widehat{s}_i(t)$.

[4] For any time t, we define t^- as $\lim_{\varepsilon \to 0}\{t - \varepsilon\}$, with $\varepsilon > 0$.

Now, by defining for each flow i, at any time t,

$$\tau_i = \widehat{s}_i(t) + \widehat{RTT}_i(t) - t \; , \tag{8}$$

i.e., τ_i is the (estimated) remaining time before the start of the next round for TCP flow i, and by iterating the projection technique of Section 2.1 for all flows, the router first computes the projected congestion window in the next round, $\widetilde{W}_{i,t}(t + \tau_i)$ for each flow i, using Eqn. (5). Then, for any $\tau > 0$, the projected arrivals are

$$\widetilde{R}_t^{in}(t + \tau) = R^{in}(t) + \sum_i MSS_i \cdot \gamma_{i,t}(\tau) \cdot \widetilde{W}_{i,t}(t + \tau) \; , \tag{9}$$

where

$$\gamma_{i,t}(\tau) = \begin{cases} 1 & \text{if } \tau \geq \tau_i, \\ 0 & \text{otherwise.} \end{cases}$$

If the condition given in Eqn. (7) is violated for any of the τ_i's of Eqn. (8), the algorithm proactively marks the oldest backlogged packet from flow j with

$$j = \arg\max\{i \mid \widetilde{W}_{i,t}(t + \tau_i) = \widehat{W}_i(t) + 1\} \; , \tag{10}$$

that is, the algorithm marks the flow with the largest congestion window that has not yet been marked (or experienced a packet drop) during in its current round. As soon as the oldest backlogged flow-j packet is marked, $\widetilde{W}_{j,t}(t + \tau_j)$ is set to $\widehat{W}_j(t)/2$, and the condition of Eqn. (7) is reevaluated. The marking process is repeated until Eqn. (7) does not hold for any of the τ_i's, or all flows have one packet marked in the current round.

3 Emulating the Reference Algorithm with a Scalable Heuristic

The per-flow information required by the algorithm presented in Section 2 involves a significant amount of overhead. We now present a heuristic approximation of the reference algorithm, which uses flow filtering to reduce the number of tracked TCP flows, and employs linear interpolation to reduce the computational complexity of the projection algorithm. Our goal is to design a heuristic algorithm that is deployable in an edge or an access router.

3.1 Flow Filtering

As observed in measurement studies [2, 12, 13], only a small percentage of flows ("heavy-hitters") accounts for a large percentage of traffic. These heavy-hitters transmit at a high data rate due to (1) a large congestion window, and (2) a relatively small round-trip time. From the description of the reference algorithm in Section 2, these are generally the only flows that marked by the reference algorithm. Thus, by limiting the tracking algorithm to the heavy-hitters we expect that the reference algorithm can be closely approximated.

To identify the heavy-hitters, we use the serial multistage filter proposed in [9]. The objective of the multistage filter is to identify, at any time t, the flows that have sent more than M bytes during the time interval $(\lfloor t/\Delta \rfloor \cdot \Delta, t)$, where M is a given threshold, $\Delta > 0$ is a fixed time constant denoting the sampling interval used for measurement. The serial multistage filter proposed in [9] works as follows. Every time a packet arrives at the router, a hash function is applied to the source and destination IP addresses and port numbers. Flows are then grouped into buckets depending on the value returned by the hash function. Then, flows in the largest buckets are hashed by a second, independent, hash function and grouped into second-level buckets. The same type of hashing operation is repeated a third time. Flows belonging to the largest buckets after the third hash are recorded into memory. The authors of [9] showed that the serial multistage filter minimizes false positives (i.e., only a few flows with a small sending rate are assumed to be heavy-hitters) and avoids false negatives (i.e., all flows with a large sending rate are tracked).

We implement flow filtering as follows. We use two linked lists in the router's memory, L_1 for current sampling, and L_2 for flows previously recorded. Initially, both L_1 and L_2 are empty. In the first sampling interval, flows are added to L_1 only if they pass the multistage filter, while L_2 remains empty. At time $t = \Delta$, L_1 is copied into L_2 before being reset.[5] The process is iterated every Δ seconds. At any time t, the router updates the estimates $\widehat{RTT}$, $\widehat{W}$, and $\widehat{s}$ for all flows in L_1 and L_2.

Only the flows in L_2 are used for the projections, and therefore, the projection of Eqn. (9) always underestimates the input curve. To alleviate this problem, at any time t, we introduce a correction factor, $\rho(t)$, whose value is updated at $t = k\Delta$, where k is a positive integer, with

$$\rho(t) = \frac{R^{in}(t) - R^{in}((k-1)\cdot\Delta)}{\sum_{i\in L_2}\left(R_i^{in}(t) - R_i^{in}((k-1)\cdot\Delta)\right)} ,$$

where $R_i^{in}(t)$ denotes the amount of flow-i traffic received by the router by time t. That is, at any time t, $\rho(t)$ denotes the ratio of the total amount of traffic received by the router in the previous sampling interval over the amount of traffic that was identified in the previous sampling interval. Note that we always have

$$\rho(t) \geq 1 .$$

The case $\rho(t) = 1$ is an extreme case where all flows pass the filter during the previous sampling interval. As an example, for $t = 5.5$ seconds, and $\Delta = 1$ second, if $\rho(t) = 1.1$, we know that 90.9 % of all traffic received by the router in the time interval $(4\text{ s}, 5\text{ s})$ has been identified. At any time t, the projection of the input curve for the Class-n traffic aggregate, $\widetilde{R}_{i,t}^{in}$ is set equal to the sum of the projection of the input curves of the flows in L_2, multiplied by the correction factor $\rho(t)$, that is

$$\widetilde{R}_t^{in}(t+\tau) = \rho(t)\cdot\sum_{i\in L_2}\widetilde{R}_{i,t}^{in}(t+\tau) .$$

Remark: We note that the selection of the parameters Δ and M presents a trade-off between computational overhead and accuracy of the algorithm. With a larger sampling interval Δ, the updates to main memory, L_2, are performed less frequently, at the expense of using possibly obsolete data. With a larger value for M, the number of recorded flows, X, remains small, but the accuracy of the projections may be poor. Thus, we infer that both M and Δ should be tuned according to the computational power available. In particular, routers at high-speed access points, and a large number of flows, should be configured with relatively large values for M and Δ.

[5] This operation can be implemented efficiently by swapping the two pointers on L_1 and L_2, and resetting the pointer on L_1.

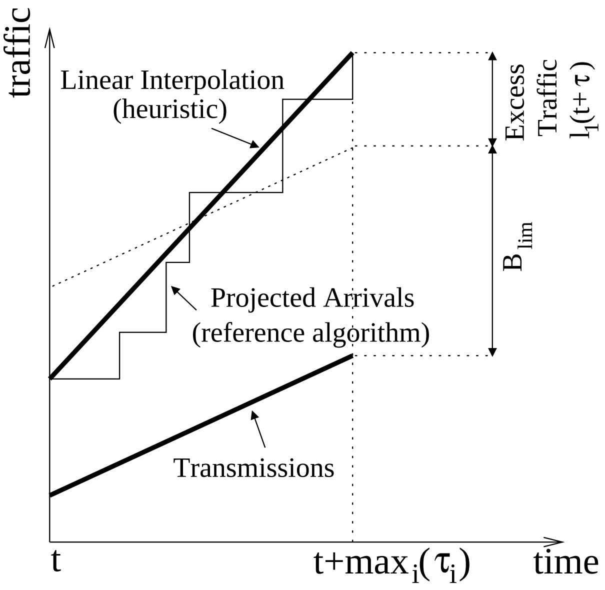


Figure 2: **Linear interpolation.** In the heuristic, only the value $\widetilde{R}_t^{in}(t+\max_i\{\tau_i\})$ is computed, and is used to determine the excess traffic that will arrive at the router.

3.2 Linear Interpolation

Flow filtering limits the amount of state information recorded at the router, but does not alleviate the computational overhead for constructing the projected input curve. We next describe a technique that reduces the complexity of the projection of Eqn. (9).

First, instead of using individual values of the congestion windows of all recorded flows in the construction of the projected input curve, we consider that all recorded flows have a congestion window size (in bytes) equal to the mean congestion window size (in bytes), $\bar{\Omega}$, given by

$$\bar{\Omega}(t) = \frac{1}{X}\sum_{i\in L_2} MSS_i \cdot \widehat{W}_i(t) \ .$$

Since we perform flow filtering and ignore flows with small congestion windows, this approximation is reasonably accurate.

Second, we use linear interpolation to reduce the complexity of the construction of the projected input curve and illustrate our method in Figure 2. Rather than constructing the whole projected input curve, only the value $\widetilde{R}_t^{in}(t+\max_i\{\tau_i\})$ is computed. Intermediary values $\widetilde{R}_t^{in}(t+\tau)$ for $0<\tau<\max_i\{\tau_i\}$ are approximated using a linear interpolation, based on the value obtained for $\widetilde{R}_t^{in}(t+\max_i\{\tau_i\})$. The reason for selecting $\max_i\{\tau_i\}$ as the basis for the linear interpolation, instead of, for instance, $\min_i\{\tau_i\}$, is that the projection can take into account all recorded flows.

Next, Eqn. (7) tells us that

$$l_1(t) = \widetilde{R}_t^{in}(t+\max_i\{\tau_i\}) - R^{out}(t) - C\cdot\max_i\{\tau_i\} - B_{lim}$$

is the amount by which the traffic must be reduced to prevent buffer overflows. From $l_1(t)$ and $\bar{\Omega}(t)$, the algorithm can infer the number of flows that have to be marked, and only update the projected input curve once, which reduces the worst-case complexity of the entire projection algorithm to $O(1)$. If $l_1(t) > 0$, the marking process performs at most $O(Q)$ operations where Q is the number of backlogged packets. The worst-case occurs when all packets backlogged have to be marked at the same time. In practice however, we only expect at most a couple of flows to be marked upon each packet arrival, since projections are performed over short time intervals, which makes this heuristic efficient in practice.

4 Traffic Regulation with ECN Marking in Class-Based Service Architectures

In this section, we build on the algorithms we described in Sections 2 and 3 to describe how ECN marking can be used to support class-based service guarantees for TCP traffic. We consider a QoS architecture that supports class-based service guarantees at each router (on a hop-by-hop basis). Traffic in the same class has the same QoS requirements. We assume that there is no admission control and no signaling, and the only method to control the traffic into a router is by dropping or by notifying TCP sources using ECN. Our goal is to design a novel approach, solely relying on ECN, for traffic regulation in QoS networks.

We consider a router in the network with output link capacity C, and assume that each class n is transmitted at time t with a service rate $r_n(t)$, such that for any t, $\sum_n r_n(t) = C$, where $r_n(t) > 0$ only if there is a positive backlog of Class-n packets.

Let us introduce the "Class-n delay", $D_n(t)$, as the queueing delay experienced by the last Class-n packet that has been transmitted before time t. Consider that a given class n is offered a bound d_n on the queueing delay of all packets in Class n, i.e., for all t, $D_n(t) \leq d_n$, and a guaranteed throughput μ_n such that at all times Class n traffic is backlogged, $r_n(t) \geq \mu_n$. Denoting by $R^{in}_{n,*}(t)$ the Class-n input curve (i.e., the total amount of Class-n traffic to have arrived by time t) and by $R^{out}_{n,*}(t)$ the Class-n output curve, following [6], a sufficient condition for all Class-n traffic to meet its delay and throughput guarantees at any time t when Class n is backlogged is

$$r_n(t) \geq r_n^{\min}(t) = \max\left\{\mu_n, \frac{R^{in}_{n,*}(t) - R^{out}_{n,*}(t)}{d_n - D_n(t)}\right\} .$$

Thus, at any time t, we need to have

$$\sum_n \max\left\{\mu_n, \frac{R^{in}_{n,*}(t) - R^{out}_{n,*}(t)}{d_n - D_n(t)}\right\} \leq C . \tag{11}$$

If the condition of Eqn. (11) is violated, one can reduce $R^{in}_{n,*}(t)$ by dropping traffic. Since our objective is to avoid any traffic drops, we use the projections described earlier to ensure that the $R^{in}_{n,*}(t)$'s always satisfy Eqn. (11).

Assuming the throughput guarantees are appropriately chosen, that is,

$$\sum_n \mu_n < C ,$$

we propose the following approach. At time t, in addition to the projections on the input curve of all flows i in Class n, $\widetilde{R}^{in}_{n,i,t}(t+\tau)$, which is given by Eqn. (6), and the class-n projected input curve, given by

$$\widetilde{R}^{in}_{n,*,t}(t+\tau) = \sum_i \widetilde{R}^{in}_{n,i,t}(t+\tau) \; .$$

We also project the Class-n output curve, $\widetilde{R}^{out}_{n,*,t}(\tau)$ by

$$\widetilde{R}^{out}_{n,*,t}(t+\tau) = R^{out}_{n,*}(t) + \tau \cdot r_n(t) \; ,$$

where $\tau > 0$. If the rate allocation r_n remains unchanged between t and $t+\tau$, this projection of $\widetilde{R}^{out}_{n,*,t}(t+\tau)$ is exact. Since we only use the projection for small values of τ (in the order of a round-trip time) we can assume that the projection is reasonably accurate, even if r_n changes between t and $t+\tau$. Furthermore, let us assume that the delay of Class n remains roughly constant during $[t, t+\tau]$. With these projections defined, we can project the minimum service rates $\widetilde{r}^{\min}_{n,t}(t+\tau)$ needed at time $t+\tau$, so that all service guarantees on throughputs and delays are met:

$$\widetilde{r}^{\min}_{n,t}(t+\tau) = \max\left\{ \mu_n, \frac{\widetilde{R}^{in}_{n,*,t}(t+\tau) - \widetilde{R}^{out}_{n,*,t}(t+\tau)}{d_n - D_n(t)} \right\} \; .$$

To ensure that the set of service rates required for meeting service guarantees is always feasible, we must enforce

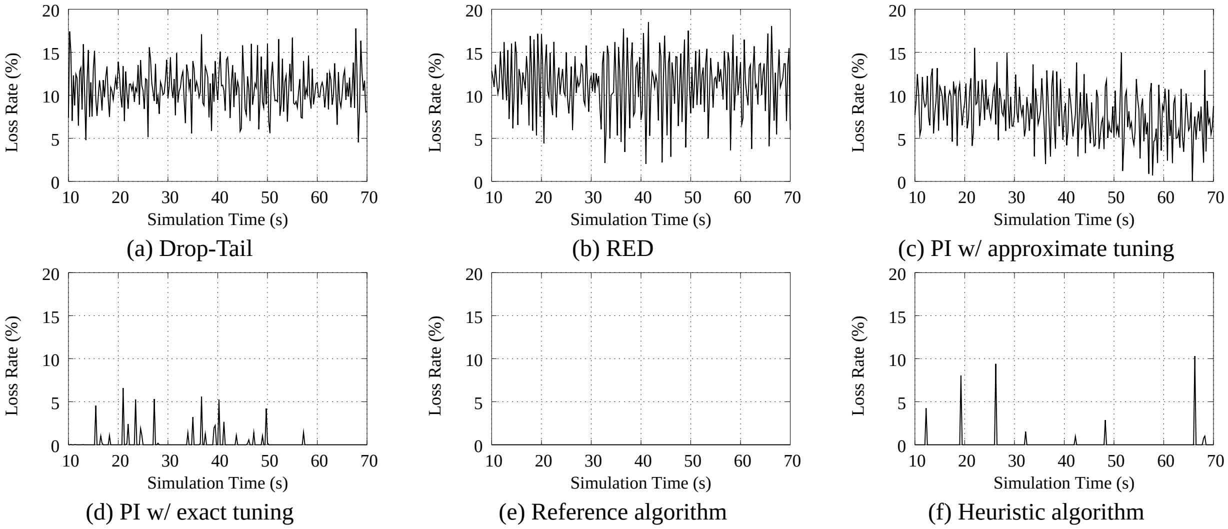
$$\sum_n \widetilde{r}^{\min}_{n,t}(t+\tau_i) \leq C \; , \tag{12}$$

for all τ_i's defined by Eqn. (8). If Eqn. (12) does not hold, the incoming traffic needs to be reduced. To that effect, we propose to first identify the set of classes where $\widetilde{r}^{\min}_{n,t}(t+\tau) > \mu_n$, which are the classes where decreasing the traffic arrivals has an effect on the minimum service rate required. Since $\sum_n \mu_n < C$, we know that there is at least one class in that set.

Once the classes whose traffic need to be throttled have been identified, the marking process is carried out in the same manner as in the case of an impending buffer overflow, by merely replacing the condition given in Eqn. (7) by the condition given in Eqn. (12).

5 Evaluation

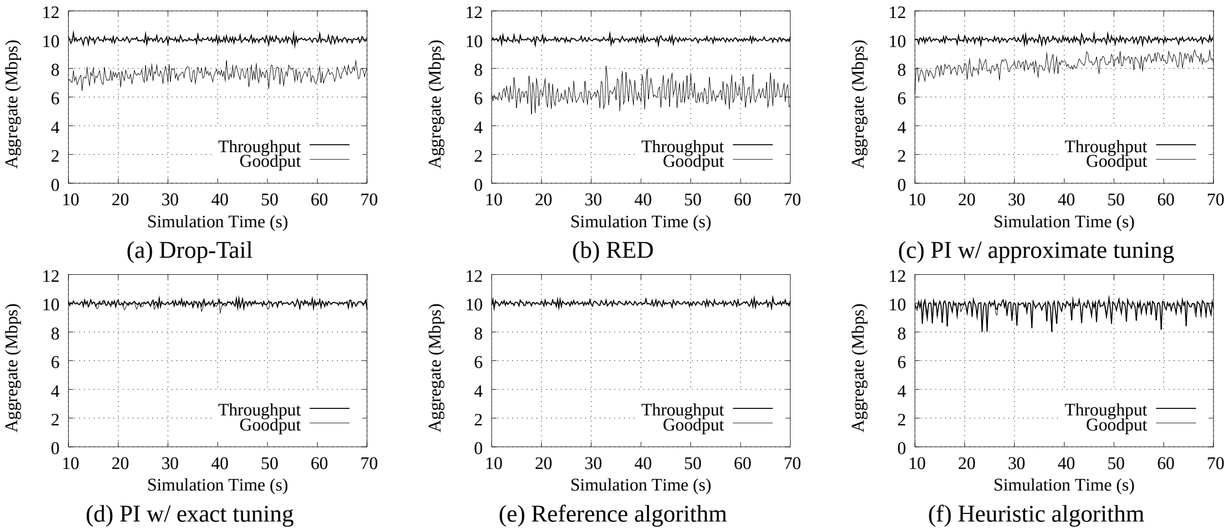
In this section, we evaluate our proposed algorithms via simulation, using the *ns-2* network simulator [1]. The evaluation has three objectives. First, we compare the performance of the reference and heuristic algorithms. Second, we compare the performance of our proposed algorithms to state-of-the-art active queue management algorithms. Third, we illustrate the potential of the proposed approach for traffic regulation in class-based service architectures. To that effect, we propose two simulation experiments. The first experiment evaluates the efficiency of the proposed approach with respect to buffer management, while the second experiment evaluates the performance of the heuristic algorithm for traffic regulation when providing QoS guarantees.



(a) Drop-Tail (b) RED (c) PI w/ approximate tuning

(d) PI w/ exact tuning (e) Reference algorithm (f) Heuristic algorithm

Figure 3: **Loss rates**. The figures compare the loss rates obtained with all six algorithms. Note that the reference algorithm does not discard any traffic.



(a) Drop-Tail (b) RED (c) PI w/ approximate tuning

(d) PI w/ exact tuning (e) Reference algorithm (f) Heuristic algorithm

Figure 4: **Measured throughput and goodput at the receivers**. The figures compare the aggregate throughput and goodput seen at the receivers with all six algorithms. All schemes are efficient at maximizing the utilization of the bottleneck link (10 Mbps). A perfectly tuned PI, and both the reference and heuristic algorithm have a goodput (amount of traffic passed to the application layer at the destinations) almost equal to the throughput (total amount of traffic received at the destinations) by avoiding packet drops.

5.1 Experiment 1: Active Queue Management

In the first simulation experiment, we consider a bottleneck link with capacity $C = 10$ Mbps, and buffer size of $B =$ 150,000 bytes. All traffic at this single bottleneck link is TCP (NewReno), and is generated by 60 greedy FTP flows, and 180 on-off flows, aiming at emulating HTTP connections. The sources of the on-off flows send on average 300 packets during an "on" period, and pause on average for one second between two "on" periods. The actual number of packets sent and the wait time between two transmissions are exponentially distributed. All packets have a size of 500 bytes. In the absence of queueing and transmission delays in the network, the RTTs of all flows are independent identically distributed random variables uniformly distributed between 24 ms and 180 ms, and to avoid synchronization effects, sources start transmitting at different times, uniformly distributed between 0 s and 5 s. The experiment lasts for 70 seconds of simulated time, and ECN is available in the entire network. We compare the performance of six different algorithms at the router governing the bottleneck link:

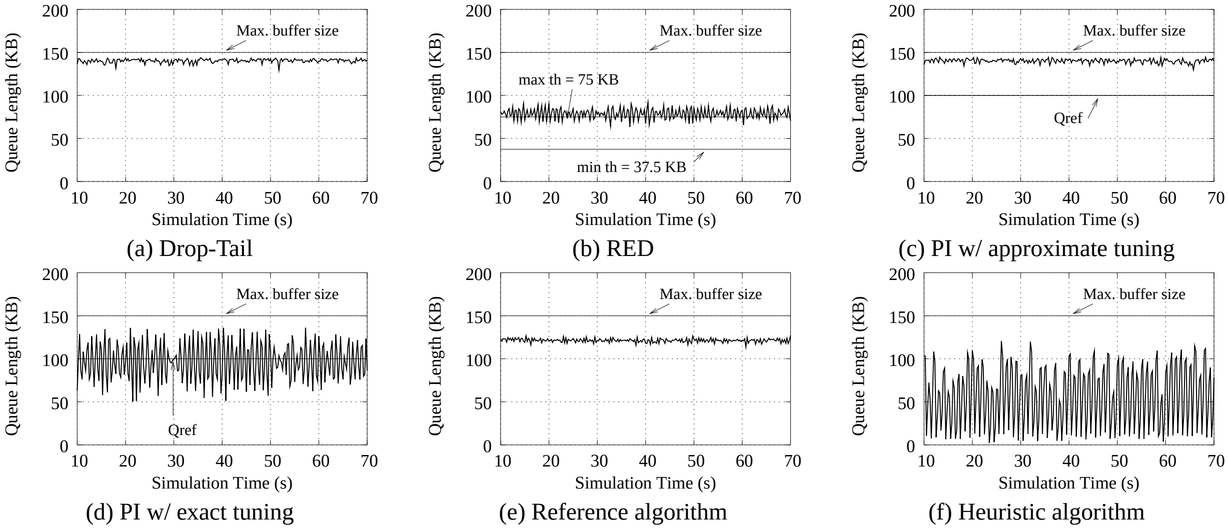
- **Drop-Tail.** We use Drop-Tail to have an estimate of the loss rates encountered without active queue management. With Drop-Tail, incoming packets are discarded only when the buffer is full.

- **RED [18].** We use RED with the `gentle_` variant [17], with a minimum threshold $min_{TH} =$ 37,500 bytes, and a maximum threshold $max_{TH} =$ 75,000 bytes. The parameter max_P is set to 0.1, and the weight used in the computation of the average queue size is set to $w_q = 0.002$. While min_{TH} and max_{TH} are chosen so that traffic is dropped with a probability of one only if the buffer is full, other parameters are the default RED parameters in *ns-2*, and are therefore expected to cover a large range of operating conditions. RED is instructed to use ECN when needed.

- **PI [21] with approximate parameter tuning.** To account for the uncertainty on estimates of the RTTs and of the number of flows at router configuration time, we configure here the PI algorithm with crude estimates of the RTTs and of the number of flows. That is, we use a lower bound on the number of flows of $N = 50$, and a maximum RTT $R^+ = 300$ ms, with a sampling frequency of 160 Hz, yielding parameter values of $a = 0.2395e - 4$ and $b = 0.2388e - 4$. The target queue length is set to Q_{ref}=100,000 bytes.

- **PI with exact parameter tuning.** We configure the PI algorithm with the exact RTTs and number of flows we use in our simulation. In other words, we use a lower bound of $N = 60$ on the number of flows, and a tight upper bound on the round-trip times $R^+ = 180$ ms, with a sampling frequency of 160 Hz, and get $a = 1.643e - 4$ and $b = 1.628e - 4$. The target queue length Q_{ref} is set to 100,000 bytes. Note that such an exact parameter tuning is unrealistic in practice, since it imposes a priori knowledge of the number of flows and of the round-trip times of the flows that will traverse the router at router configuration time.

- **Reference algorithm.** This is the reference algorithm described in Section 2. Results are obtained for $K = 10$, $\alpha = 0.9$. We achieved similar results with parameter settings in the range $10 \leq K < 100$ and $0.7 < \alpha < 1$, which tends to show that our proposed algorithm is relatively insensitive to the parameter selection.



(a) Drop-Tail

(b) RED

(c) PI w/ approximate tuning

(d) PI w/ exact tuning

(e) Reference algorithm

(f) Heuristic algorithm

Figure 5: **Queue lengths**. The figures compare the queue lengths at the router for all six algorithms.

- **Heuristic algorithm.** This is the heuristic algorithm described in Section 3. The multistage filter consists of 3 stages of 8 buckets. The admission threshold is set to M =200,000 bits and $\Delta = 1$ s.

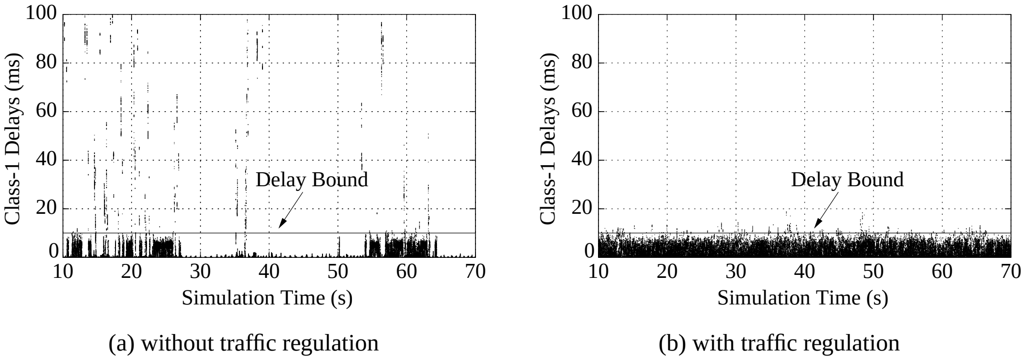
For each algorithm, we monitor the loss rates over a sliding window of size 0.25 s, and present our results in Fig. 3. We start monitoring at time $t = 10$ s to ignore transient effects linked to the fact the network is initially empty. Fig. 3(a) tells us that, without active queue management, one can expect loss rates in the order of 12 %. Fig. 3(b) and (c) show that RED with the default parameters, which turn out to be unsuitable for the traffic mix at hand, and a crudely configured PI algorithm, drop almost as much traffic as Drop-Tail. Conversely, a perfectly tuned PI algorithm manages to avoid most packet drops. The reference algorithm completely avoids packet losses, and the heuristic rarely drops any packets. Thus, the heuristic algorithm delivers results close to the reference algorithm.

Next, we monitor the aggregate throughput and goodput observed at the receivers, averaged over a moving window of size 0.25 s and present our results in Fig. 4. The throughput characterizes the total amount of traffic received by the transport layer at the destination, while the goodput characterizes the amount of traffic that is passed by the transport layer to the application layer. The main observation is that all schemes manage to achieve an aggregate throughput roughly equal to the capacity of the bottleneck link. Furthermore, we note that, by avoiding packet losses, an exactly tuned PI, and both the reference and heuristic algorithms manage to achieve a goodput close to the throughput.

Last, we monitor the queue size, averaged over a moving window of size 0.25 s, and we present our results in Fig. 5. Not surprisingly, the Drop-Tail queue is almost always full, which explains the relatively high loss rates. RED manages to stabilize the queue length around max_{TH} = 75,000 bytes. (This observation coupled with the result presented in Fig. 3 indicates that RED drops some packets proactively even when ECN is available.) With an approximate tuning of the configuration parameters, PI does not manage to track the desired queue length Q_{ref} = 100,000 bytes, and instead, the queue is almost always full. Conversely, a properly tuned PI algorithm manages to achieve the target Q_{ref}, albeit with some oscillations around the

Class	Number of on-off flows	QoS Guarantees		
		Delay	Loss Rate	Throughput
1	5	≤ 10 ms	≤ 1 %	≥ 5 Mbps
2	10	$\approx \frac{1}{4}D_3$	$\approx \frac{1}{2}p_3$	–
3	15	$\approx \frac{1}{4}D_4$	$\approx \frac{1}{2}p_4$	–
4	20	–	–	–

Table 1: **Traffic mix and service guarantees.** The second column indicates the number of on-off flows, and in the third and fourth rows, p_n denotes the loss rate of Class n over a busy period, D_n denotes the delay of Class n.



(a) without traffic regulation

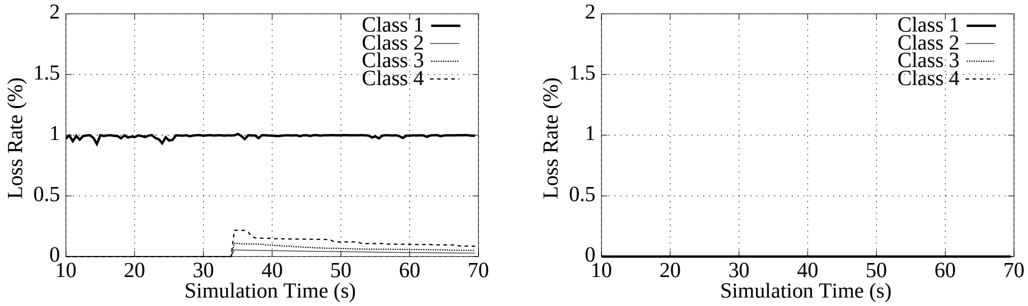
(b) with traffic regulation

Figure 6: **Class-1 packet delays.** Note that the number of violations is significantly lower with traffic regulation, and the violations are much smaller in magnitude.

target value. While stabilizing the queue length is not the primary objective of our algorithms, the reference algorithm manages to keep the queue length almost constant around 40,000 bytes. The heuristic algorithm also keeps the queue length in the vicinity of 40,000 bytes, with oscillations of a magnitude comparable to those of a well-configured PI controller. These oscillations are mostly due to the fact that the sampling interval is set to 1 s, and are reduced for higher sampling frequencies, at the expense of a higher computational overhead.

5.2 Experiment 2: Providing Service Guarantees

Next, we assess the effectiveness of our algorithms at regulating traffic for providing service guarantees. To that effect, we run a second experiment, with a bottleneck link with capacity $C = 45$ Mbps, and a buffer size of $B = 250,000$ bytes. All traffic at the bottleneck link is TCP (NewReno), and consists of 12 greedy TCP flows, and 50 on-off TCP flows, following the same on-off pattern as in the first experiment. The RTTs of all greedy TCP flows are equal to 44 ms, and the RTTs of the on-off flows, in the absence of propagation and transmission delays, are uniformly distributed between 44 ms and 80 ms. All sources start transmitting at time $t = 0$ for 70 seconds of simulated time, and ECN is available.



(a) without traffic regulation

(b) with traffic regulation

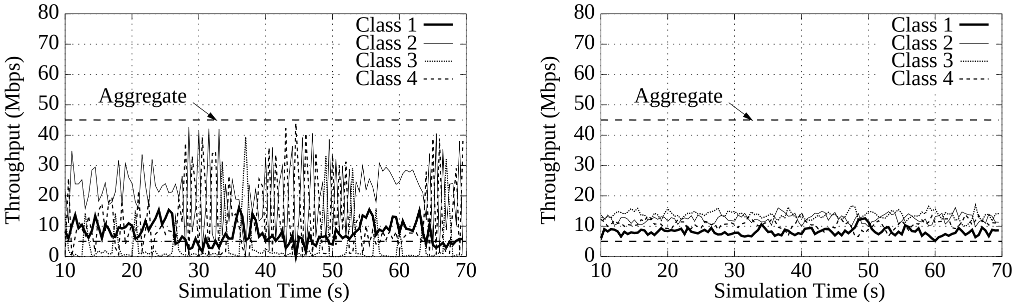
Figure 7: **Loss rates.** The traffic regulation algorithm prevents any traffic from being dropped.

We consider four classes of traffic, with the service guarantees and traffic mix described in Table 1. In addition to the on-off flows, each class contains three greedy TCP flows. We compare the performance of two algorithms in this experiment. The first algorithm is the algorithm described in [6], which can provide delay and loss guarantees to traffic classes, but does not regulate traffic. The second algorithm combines the algorithm of [6] with the traffic regulation algorithm described in Section 4 and the heuristic approximations described in Section 3, using a multistage filter of 3 stages of 8 buckets, $\Delta = 0.1$ s, $M = 200,000$ bits.

We plot the delays encountered by each Class-1 packet at the bottleneck link in Fig. 6. Fig. 6(a) shows that, given the traffic mix considered, about 11 % of all Class-1 packets exceed the delay bound of 10 ms, with queueing delays going as high as 100 ms. This is due to the fact that, when a loss guarantee and a delay bound conflict due to the absence of admission control, the algorithm of [6] gives precedence to the loss guarantee and relaxes the delay bound. Conversely, Fig. 6(b) shows that when the traffic regulation algorithm we described in this paper is used, violations rarely happen (< 2 %), and the delay does not exceed 20 ms.

Next, in Fig. 7, we plot the loss rates averaged over the length of the current busy period. Fig. 7(a) show that all loss guarantees are respected, notably the 1 % bound on Class-1 losses. However, as we have seen in Fig. 6(a), the loss rate bound is respected at the expense of the delay bound. Fig. 7(b) shows that, with the addition of the algorithm of Section 4, no packets are lost, and therefore, the objective of completely avoiding packet drops to meet service guarantees is met.

Finally, in Fig. 8 we present the throughput obtained by each class at the bottleneck link. Fig. 8(a) shows that in the absence of traffic regulation, severe oscillations of the throughput can be observed. Worse, the throughput bound on Class-1 is sometimes violated, due to the fact that there is not enough Class-1 traffic present in the router. Fig. 8(b) shows that traffic regulation stabilizes these oscillations in throughput, and that the throughput guarantee on Class 1 is always respected. We also note that both algorithms manage to achieve an aggregate throughput equal to the capacity of the bottleneck link, meaning that the stabilization in the throughputs provided by the traffic regulation algorithm does not come at the expense of under-utilization.



(a) without traffic regulation

(b) with traffic regulation

Figure 8: **Per-class throughputs.** Without traffic regulation, we observe oscillations and sporadic violations of the Class-1 throughput guarantee. The traffic regulation algorithm stabilizes these oscillations and ensures the throughput guarantees are respected.

6 Conclusions and Discussion

We investigated whether marking algorithms for ECN can be used for regulating traffic in the context of class-based service architectures, while avoiding packet losses due to buffer overflows. To that effect, we first described two packet marking algorithms for IP routers, which attempt to eliminate packet losses in TCP flows. The proposed approach infers how traffic is sent by TCP sources, by tracking the window size and RTT of large flows, and accordingly makes the marking decisions. We then showed how the proposed algorithms can be used for traffic regulation in the context of QoS architectures, in lieu of traffic policing or admission control. Experimental results illustrated the potential of the approach.

We note that the techniques used in the algorithms can be further improved by more accurate and robust estimators of the RTT values, e.g., [25], and of the congestion window sizes. Another area for improvement resides in the type of filter used in the heuristic algorithm. While the serial multistage filter [9] we use in this paper appears to exhibit good performance, a follow-up work described in [10], indicates that parallel multistage filters typically perform better than serial multistage filters, and are more amenable to mathematical analysis of their properties, such as probabilities of false negatives. Using a parallel multistage filter could therefore open the door for an analytical evaluation of our proposed algorithms, and help quantify the trade-offs in parameter selection. Furthermore, our current approach assumes TCP Reno or NewReno; extending it to other flavors of TCP such as SACK or Vegas is left for future research. Last, the heuristic algorithm proposed is probably efficient enough to be deployed at relatively low-speed links, such as the links at the edges of the network, where the speed is in the order of a few hundred megabits per second, but it is our belief that the proposed heuristic might still have a computational overhead too important to consider deployment in core routers operating at speeds in the order of tens of gigabits per second. Hence, further work is probably needed to propose heuristics that can also be deployed in the core of the network.

Acknowledgments

We wish to thank Cristian Estan for providing us with his implementation of the multistage filters used in the heuristic algorithm.

References

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

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

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

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

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

[6] N. Christin, J. Liebeherr, and T. F. Abdelzaher. A quantitative assured forwarding service. In *Proceedings of IEEE INFOCOM 2002*, volume 2, pages 864–873, New York, NY, June 2002.

[7] K. Claffy, G. Miller, and K. Thompson. The nature of the beast: recent traffic measurement from an Internet backbone. In *Proceedings of INET '98*, Geneva, Switzerland, July 1998.

[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. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2001 ACM SIGCOMM Internet Measurement Workshop*, pages 75–80, San Francisco, CA, November 2001.

[10] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM '02*, pages 323–336, Pittsburgh, PA, August 2002.

[11] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communications Review*, 26(3):5–21, July 1996.

[12] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Proceedings of IEEE GLOBECOM '99*, pages 1859–1868, Rio de Janeiro, Brazil, December 1999.

[13] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: methodology and experience. In *Proceedings of ACM SIGCOMM '00*, pages 257–270, Stockholm, Sweden, August 2000.

[14] W.-C. 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.

[15] W.-C. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic fair blue: a queue management algorithm for enforcing fairness. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1520–1529, Anchorage, AK, April 2001.

[16] W.-C. Feng, D. Saha, D. Kandlur, and K. Shin. A self-configuring RED gateway. In *Proceedings of IEEE INFOCOM '99*, volume 3, pages 1320–1328, New York, NY, March 1999.

[17] S. Floyd. Recommendation on using the `gentle_` variant of RED, March 2000. See `http://www.icir.org/floyd/red/gentle.html`.

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

[19] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi. Packet-level traffic measurements from a tier-1 IP backbone. Technical Report TR-01-ATL-110101, Sprint ATL, November 2001.

[20] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. A control-theoretic analysis of RED. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1510–1519, Anchorage, AK, April 2001.

[21] 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.

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

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

[24] 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`.

[25] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM Computer Communication Review*, pages 75–88, July 2002.

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

[27] V. Misra, W. Gong, and D. Towsley. A fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of ACM SIGCOMM 2000*, pages 151–162, Stockholm, Sweden, August 2000.

[28] 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.

[29] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *Proceedings of ACM SIGCOMM '98*, pages 303–314, August 1998.

[30] 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*, volume 2, pages 942–951, Tel-Aviv, Israel, April 2000.

[31] V. Paxson. Automated packet trace analysis of TCP implementations. In *Proceedings of ACM SIGCOMM '97*, pages 167–179, Cannes, France, August 1997.

[32] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. IETF RFC 3168, September 2001.

[33] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Reading, MA, 1998.