Using Precise Feedback for Controlling Congestion in the Internet

MIT/LCS/TR-820

By

Dina Katabi, Mark Handley

MASSACHUSETTS INSTITURTE OF TECHNOLOGY
Laboratory for Computer Science
200 Technology Square, Cambridge, MA 02139

May 2001

Using Precise Feedback for Controlling Congestion in the Internet

Dina Katabi (dinaktbi@MIT.EDU), Mark Handley (mjh@ACIRI.ORG)

Abstract

This paper explores the potential of improving Internet congestion control by providing precise congestion feedback to the sources instead of the implicit and binary feedback used by TCP. We propose a window-based protocol in which each sender maintains its congestion window and round trip time and communicates this information to the routers in a congestion header attached to every packet. The routers use the congestion header to tell each sender the exact amount by which to increase or decrease its congestion window. They apportion the feedback among the flows based on their congestion windows and round trip times provided in the header. The proposed protocol does not require any per-flow state at the routers; nevertheless, its fairness is significantly superior to TCP's. Further, the queue size under the new protocol is smaller, the link utilization is near optimal and the drop rate is zero in most cases. Moreover, the high responsiveness of the protocol allows efficient operation in very high bandwidth environments and dynamic environments with many short flows. We use simulations in various settings to demonstrate these desirable properties and explore their limitations.

1 Introduction

Most Internet congestion control uses a window-based mechanism embedded in TCP. At the core of this protocol is an additive-increase/multiplicative-decrease (AIMD) control law illustrated by Equation 1.

$$I: \; w(t+rtt) = w(t) + 1; \qquad D: \; w(t+rtt) = w(t) - \frac{1}{2}w(t). \qquad (1)$$

Equation I is the increase rule: when no packet loss occurs in a round trip time (rtt) the congestion window increases by one packet. Equation D is the decrease rule: when a packet is lost the window is halved.

TCP's congestion control uses implicit and binary feedback. The feedback is implicit because the sender has to interpret a packet loss as an implicit signal of congestion to which it reacts by halving its sending window. The feedback is binary because the sender knows only whether there is congestion (a packet loss) or there is no congestion (no loss). As a result, the sender must probe the network and continuously oscillate between under- and over-loading the bottleneck.

Using precise congestion feedback, on the other hand, means the network explicitly informs the senders about congestion and how to react to it. Such a control protocol bases its control decision on the level of congestion. It can decrease its sending window quickly when the bottleneck is highly congested whilst performing small reductions when the sending rate is close to the bottleneck capacity. Thus, with precise feedback, we can simultaneously make the sources more responsive and less oscillatory. As an example of precise feedback, consider changing TCP's congestion control equations as follows:

$$I: \; w(t+rtt) = w(t) + \alpha(t); \qquad D: \; w(t+rtt) = w(t) - \beta(t) \times w(t), \qquad (2)$$

where $\alpha(t)$ and $\beta(t)$ are feedback parameters provided by the bottleneck (e.g., in the packet's header) and change over time as the congestion state of the bottleneck changes.

This example indicates that the control space available with precise feedback is significantly larger than that available with binary feedback. The challenge is to find a protocol that improves utilization and fairness, decreases losses, and removes persistent queues, while simultaneously maintaining TCP's capabilities of serving a huge number of flows in a heterogeneous network. This paper introduces PCP, a *Precise-feedback Congestion-control Protocol* that needs no per-flow state in routers, yet exhibits substantial improvement over TCP in terms of network utilization, drop rate, queue size, and fairness. The novelty of PCP comes from its decoupling of utilization control from fairness control allowing each controller to focus on a simpler problem and so use the design that best achieves its goal. PCP uses a proportional integral controller to manage utilization along with the new concept of bandwidth shuffling to achieve fairness.

The first author's research is supported by the Defense Advanced Research Projects Agency under contract N6601-98-8903 and administered by SPAWAR. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policy or endorsements, either expressed or implied, of DARPA or the US government.

Before examining PCP, we investigate the limitations of TCP stemming from its implicit binary feedback. An important limitation of implicit feedback is its inability to distinguish congestion from losses due to link errors. This causes a drop in throughput over wireless links where losses are caused by transmission errors. Proposed solutions include ECN [13], where a router may mark packets instead of dropping them. By doing so the router sends an explicit yet still binary congestion feedback to the sources.

Another limitation stems from the binary nature of the feedback and restricts the utilization of high-bandwidth links. Continuous reductions in bandwidth costs indicate that the future Internet will contain a large number of high-speed links. Their bandwidth-delay product can easily exceed 1000 packets. Consider a high-bandwidth link shared by five TCP flows. Three of these flows terminate at around the same time freeing a few gigabits/sec of capacity. As TCP increases its sending window by one packet/rtt, it takes the two remaining flows hundreds of rtts to grab the spare bandwidth. The senders only know that there is no congestion but not whether the spare bandwidth is gigabits/second or a few hundred bits per second. Thus, binary feedback makes it dangerous to increase quickly, and so bandwidth is wasted over high-speed links.

2 Design Objectives & Assumptions

We set the following design objectives for PCP.

No per-flow state at the routers: Requiring routers to hold per-flow state would limit scalability.

Good performance: Performance should be at least as good as TCP's.

Coexistence with conventional TCP: In the current Internet, any deployable protocol must be capable of coexisting peacefully with TCP. A new protocol should be TCP-friendly and its presence should not harm the network's efficiency.

Precise feedback can be used in a window- or rate-based congestion control protocol. However, PCP's initial design is window-based due to the existing experience with window-based protocols and the huge literature of TCP related research. We also wanted PCP to fall back to a TCP-like behavior whenever modifying TCP makes no performance improvement.

Our description of PCP makes the following assumptions.

End-systems are not malicious: Our description of PCP assumes hosts are not malicious and that they use the feedback returned from the network according to the protocol's rules. In this paper we do not address the problem of malicious users, but we note that PCP is not different from TCP in this regard. In both protocols, malicious hosts can obtain more bandwidth than their fair share and congest the network. Also, in PCP, malicious hosts can be detected and penalized using methods similar to those proposed for TCP such as statistical examination of the traffic or monitoring at the network edges. Indeed, precise feedback makes policing easier as a policing agent does not need to infer packet losses.

Senders are greedy: For clarity, our description of PCP assumes that senders always have data to send. However PCP works correctly with a non-greedy sender as this can be simulated by a combination of a greedy sender and a router whose output link's bandwidth is equal to the sender's demand [4]. PCP imposes a policy of "use it or lose it"; if the sender does not use all of its congestion window (cwnd) in an rtt, it reduces cwnd to the amount actually used.

Congestion is caused by data packets rather than acknowledgements: In our description, PCP does not attempt to control the rate of acknowledgements. This does not mean that PCP as described behaves incorrectly when the congestion is caused by ack traffic; in fact it behaves similarly to TCP. It is possible to extend PCP to control ack traffic in the same way as data traffic, but we do not describe such extensions in this paper.

3 PCP: A Precise-feedback Congestion-control Protocol

3.1 What should be done by the router and what should be done by the sender?

A congestion control protocol involves two components: the routers, which observe congestion, and the sources, which react to it. The first decision therefore is to decide which tasks should be performed in the routers and which in the sources.

Maintaining per-flow information: Our requirements prohibit routers from keeping per-flow state, so all such state should be maintained by the senders.

Sensing congestion: It is the routers who should sense congestion for they have full knowledge of their queue state. Requiring the senders to implicitly sense congestion via packet losses adds unnecessary delay and errors to the process.

The control law: The control law states how the sending window should be updated in response to congestion. In TCP, the control law is located at the sender and is based on the AIMD rules. The advantage of having the control decision taken by the source is that the decision can differ depending on the application (e.g., multimedia applications can be slower in reacting to congestion). However, we put the control decision at the router because this provides tighter control and increases stability; in addition to controlling each source separately, the router can control the sum of the cwnd increase/decrease performed by sources sharing a link so that the aggregate traffic stays within certain limits. It also makes sense to leave control of network resources to their owners. Locating the control law in the routers allows an ISP to control traffic dynamics at a bottleneck inside their network.

3.2 A High-Level Description of the Protocol

PCP is a window-based congestion control protocol. At a high level, PCP works as follows. Senders maintain their cwnd and rtt and communicate this information to the routers in a congestion header attached to every packet. Routers monitor the input traffic rate to each of their output links. Based on the difference between the bandwidth of a link and its input traffic rate, the router tells the sources sharing that link to increase or decrease their congestion windows. This feedback is communicated by annotating the congestion header of data packets, and the receiver then echoes it back to the sender. The feedback is apportioned among the flows based on their cwnd and rtt (which are in the congestion header) so that the system converges to fairness. A more congested router further along the path can overwrite the feedback in the congestion header to force each sender to use a smaller congestion window. As a result, eventually the packet will contain the feedback from the bottleneck along the path. When the feedback reaches the receiver, it is returned to the sender in an acknowledgment packet, and the sender updates its cwnd as appropriate. Note that PCP adapts to the bottleneck's load rather than its queue size. Hence, unlike TCP, it can distinguish between a system slightly below the bottleneck capacity and one that is considerably underutilized. The next few sections describe PCP in detail.

3.3 The Congestion Header

Each PCP packet carries a congestion header (see Figure 1), which is used to communicate a flow's state to the router and for communicating feedback from routers to the senders. *H_cwnd* is the sender's congestion window, and *H_rtt* is the sender's current RTT estimate. *H_A*, *H_B1*, and *H_B2* are single-bit fields. *H_A* distinguishes an acknowledgement from a data packet. *H_B1* is set by any router on the path when the input traffic rate exceeds the link's capacity. *H_B2* contains the value of *H_B1* in the last received acknowledgement and indicates that the flow has traversed a bottleneck in the last rtt. Apart from *H_B1*, all of the fields above are written by the sender and never modified by the routers.

The remaining field, *H_feedback*, takes negative or positive values and is initialized by the sender according to its demands (Section 3.4). Routers along the path modify this field to control the input traffic rate and match it to the capacity of the link.

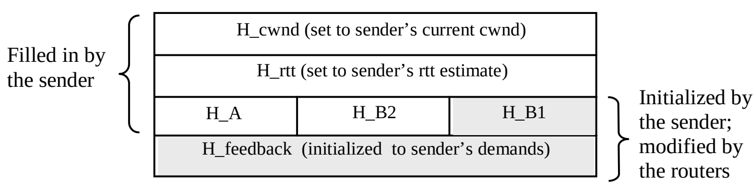

Figure 1: The congestion header.

3.4 The Role of the Sender

A PCP sender is very similar to a TCP sender except in the way it updates its congestion window. As with TCP, a PCP sender maintains a congestion window of the outstanding packets (cwnd) and an estimate of the round trip time (rtt). On packet departure, a congestion header is attached to the packet and the sender sets the *H_cwnd* field to its current cwnd and *H_rtt* to its current estimate of the round trip time. In the first packet of a flow *H_rtt* is set to zero as an indication to the routers. The *H_A* bit is set to 0 indicating this is a data packet ready for feedback collection. The *H_B1* bit is initialized to 0. If the sender does not want to increase its cwnd[1], it sets *H_B2* to 1; otherwise, it sets *H_B2* to the value of *H_B1* from the last received ack.

The sender uses the *H_feedback* field to request its desired window increase and initializes this field to the difference between its sending buffer and its cwnd.

Whenever a new acknowledgement arrives the sender updates its cwnd as follows:

(3)

[1] Either because it has reached the maximum window allowed by the receiver, or because it is the last cwnd in the transfer.

$cwnd = \max\ (s,\ cwnd + H\_feedback)$,

where *s* is the packet size, and cwnd is measured in bits. Note that a positive feedback increases the sender's cwnd and a negative feedback reduces it.

The above describes only the difference between PCP and TCP senders. The reaction of PCP to a packet loss or a timeout is similar to TCP's. [2]

3.5 The Role of the Receiver

A PCP receiver is very similar to a TCP receiver except that when acknowledging a packet, it copies the congestion header from the packet to its acknowledgment and sets *H_A* to 1.

3.6 The Role of the Router

A PCP router[3] is a drop-tail or RED queue augmented with an efficiency controller (EC) and a fairness controller (FC). Before describing the details of the two controllers, we present the three concepts underlying both EC and FC.

3.6.1 Estimation and Control

A PCP router estimates the congestion state of its output link and applies an appropriate control decision. Estimation and control are done simultaneously and in a discrete manner. The router has two timers: an *estimation timer* and a *control timer*. These timers divide time into asynchronous periods of estimation and control (see Figure 2). We define the following:

The estimation interval T_e: the interval over which the router estimates all of its parameters (e.g., average rtt, average cwnd, input traffic rate, number of flows…*etc*.). T_e should be long enough to capture the dynamics of the traffic, yet short enough to keep the system responsive. It is set to an estimate of the average rtt of the flows sharing the link. Thus, T_e is not a fixed interval; its value changes with the average rtt of the system.

The control interval T_c: a constant interval during which a single control decision is applied. When the control timer expires, the router reads its estimators and decides on the feedback to be sent during this control interval. This decision is not updated until the next T_c timeout. T_c is chosen to be the long-term average rtt of the system, i.e., the average rtt over a month. Thus, compared to the dynamics of the traffic and T_e, it appears constant. The system is fairly insensitive to the exact value of T_c.[4]

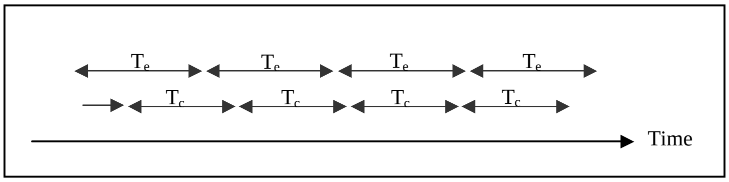

Figure 2: Illustration of the asynchrony of the estimation interval T_e and the control interval T_c. Values computed in a T_e are used in the following one or more T_c intervals.

3.6.2 Estimation

A PCP router keeps track of a set of estimators whose new values become available at T_e timeout. In this section we describe the most important estimators. Estimators not discussed in this section have intuitive computations (e.g., estimating the number of packets in a T_e by counting them) and are presented while discussing EC and FC.

Input traffic rate: The total number of bits sent to the output queue in an estimation interval divided by the interval duration.

Spare bandwidth (SBW): The difference between the capacity of the link and its input traffic rate. SBW is positive when the link is underutilized and negative when the link is congested. The objective of the control is to make SBW equal zero.

[2] The philosophy of PCP is to try to prevent losses as much as possible, a task that PCP accomplishes effectively (see Section 5). Yet when a loss happens, PCP considers this event a signal of severe congestion to which it reacts by halving its cwnd. Although this reaction might be too conservative, it ensures that in case of continuous congestion PCP is as conservative as TCP and consequently as robust to congestion collapse.

[3] We use "router" to refer to a queuing module at a router. When a router has multiple output links, each would have its own PCP queue.

[4] The experiments we conducted show that doubling T_c or halving it does not change the performance. Changing T_c further causes a smooth degradation in the performance.

Average rtt and average cwnd: The router estimates the average round trip time and congestion window of flows traversing the link. The average rtt computed in one estimation interval determines the duration of the next T_e. Although the rtt and cwnd of each flow are provided in the congestion header, the router has no per-flow state, and this complicates the computation. For example, if two flows traversing the link have cwnds of 10 and 2 packets and the same rtt, then the average cwnd is 6 packets. During an rtt, the router expects to see a cwnd of packets from each flow. Yet as it has no per-flow state, the router cannot isolate packets from any flow. For every ten packets with *H_cwnd* = 10, it sees two packets with *H_cwnd* = 2. The average *H_cwnd* in the packets is $(10\times 10+2\times 2)/12=8.6$ packets, which is not the average cwnd of the flows.

In general, a router estimating the state of a flow from the congestion header in the packets sees the state amplified by the number of packets the flow sends in an estimation interval, which is $cwnd\times(T_e/rtt)$. Thus, to counter this effect and obtain a correct average state estimate, the router should weigh the state in the packet by the round trip of the flow and the inverse of its congestion window. The correct estimates become: [5]

$$\overline{cwnd}=\frac{\sum\limits_{packets\ in\ T_e}(\mu_i\times H\_cwnd)}{\sum\limits_{packets\ in\ T_e}\mu_i} \quad\text{and}\quad \overline{rtt}=\frac{\sum\limits_{packets\ in\ T_e}(\mu_i\times H\_rtt)}{\sum\limits_{packets\ in\ T_e}\mu_i} \quad\text{where}\quad \mu_i=\frac{H\_rtt}{H\_cwnd}. \tag{4}$$

Number of flows (N): The method used to compute the average cwnd and rtt can be used to compute the number of flows. If in every estimation interval we were to receive one packet from each flow, then we would have counted the flows by counting the packets. However, in every estimation interval we receive $cwnd\times(T_e/rtt)$ packets from each flow. Thus, we can count the flows by weighing each packet by the inverse of the number of packets seen in T_e from this flow:

$$N=\frac{\sum\limits_{packet\ in\ T_e}(H\_rtt/H\_cwnd)\times 1}{T_e}. \tag{5}$$

Note that the first packet in a connection, which is marked by an *H_rtt* of 0, is ignored while computing any estimator that involves the value of *H_rtt*, (i.e., all the estimators above except the input traffic rate and SBW).

3.6.3 Decoupling Efficiency Control from Fairness Control

A congestion control protocol has two main objectives: efficiency and fairness. The efficiency goal is to maximize link utilization whilst minimizing queue size. The fairness goal, on the other hand, means that flows sharing the same bottleneck should observe the same throughput. Traditionally, these two objectives are coupled, as the same control law (such as AIMD in TCP) is used to simultaneously obtain both fairness and efficiency [1,4,8,9,10,12].

Conceptually, however, efficiency and fairness are independent. Efficiency involves only the aggregate traffic's behavior. When the input traffic rate equals the capacity of the link, no queue builds up and the utilization is optimal. Fairness on the other hand involves the relative throughput of flows sharing a link. A scheme is fair when the flows sharing a link have exactly the same throughput even if the link is completely congested.

The key characteristic of PCP is its decoupling of efficiency from fairness. In a PCP router, there is an efficiency controller (EC) and a fairness controller (FC). The two controllers work independently and have different control laws. Changing the control law in EC would change the time it takes to converge to good utilization, the drop rate, and the average queue size. However, it has minimal effect on the time to converge to fairness.

Decoupling fairness from efficiency simplifies the design and analysis of these controllers by reducing the requirements imposed on each. It also permits modifying one of the controllers without redesigning or reanalyzing the other. For example, to allow different flows to enjoy different qualities of service, we need to change the FC but the EC can be left unchanged.

3.6.4 Efficiency Controller (EC)

The efficiency controller's goal is to maximize link utilization while minimizing drop rate and persistent queues.[6] This aim is met when the input traffic rate equals the link capacity, so spare bandwidth is zero (SBW = 0). When SBW is positive, the link is underutilized, and the router should allocate more bandwidth to the flows by sending positive feedback. When SBW is negative, the link is congested, and the router should free bandwidth by sending negative feedback. Because PCP is a

[5] Although we weigh the values in the header by $cwnd\times(T_e/rtt)$, T_e is a constant and hence it cancels out.

[6] A persistent queue lasts longer than an rtt. A non-persistent queue is a queue generated by a burst of traffic but which dissolves in an rtt.

window-based protocol, allocation and freeing are done using an amount of bits. We define BTA (Bits To Allocate) as the number of bits the router wants to allocate in the current control interval T_c, and BTF (Bits To Free) as the number of bits the router wants to free during T_c. Both BTA and BTF are computed once per T_c. An intuitive approach for achieving our efficiency goal (SBW = 0) would be to allocate or free proportionally to the spare bandwidth during every control interval:

$$BTA = \max(0,\ k_1 \times SBW \times T_c); \qquad BTF = \max(0,\ -k_1 \times SBW \times T_c); \qquad 0 < k_1 \le 1 .$$

Note that when BTA is positive BTF is zero and vice versa. This proportional allocation/freeing causes the spare bandwidth to exponentially converge to zero. For example, when $k_1 = 0.5$, ignoring delay in the system, it takes three T_c intervals to allocate (or free) 87.5% of the original spare bandwidth.

In reality, there is a delay between applying the control law and observing its effects: it takes at least an rtt between setting the feedback in the congestion header and observing its effects on the spare bandwidth at the router. Because of this delay, the router might over-allocate or over-free bandwidth causing the input traffic rate to oscillate around the link's capacity, as illustrated in Figure 3. When the input rate overshoots the capacity, packets accumulate in the queue. Later as the input traffic rate under-shoots the capacity some of the packets queued during the overshoot are drained. However, because the oscillations are damped over time, some packets queued during an overshoot cannot be drained in the following undershoot, and a persistent queue gradually builds up. To avoid this, we modify our computation of BTA and BTF to:

$$BTA = \max(0,\ k_1 \times SBW \times T_c - k_2 \times queue); \qquad BTF = \max(0,\ -k_1 \times SBW \times T_c + k_2 \times queue),$$

where k_1 and k_2 are constants in the range *(0,1)* and *queue* is the queue size at T_c timeout.

The last subtlety to consider is the effect of 2-way traffic. Reverse traffic causes ack compression, and the aggregate input traffic then becomes bursty. In this case, the queue becomes a combination of a persistent queue resulting from the overshooting described above and a non-persistent queue caused by the bursty nature of the traffic. Our controller should ignore the non-persistent queue and drain only the persistent queue. Thus, we modify BTA and BTF again to:

$$BTA = \max(0,\ k_1 \times SBW \times T_c - k_2 \times \min\_queue); \quad BTF = \max(0,\ -k_1 \times SBW \times T_c + k_2 \times \min\_queue), \qquad (6)$$

where *min_queue* is the minimum size the queue has reached in the past T_c. The values of k_1 and k_2 do not depend on the characteristics of the traffic, but do depend on each other. For a certain value of k_1, k_2 must be large enough to drain the persistent queue resulting from the proportional controller. We recommend setting $k_1 = 0.4$ and $k_2 = 0.5$, which results in a good trade off between the time to reach full utilization and the damping of the oscillations.[7] Figure 4 provides pseudo-code for the efficiency controller.

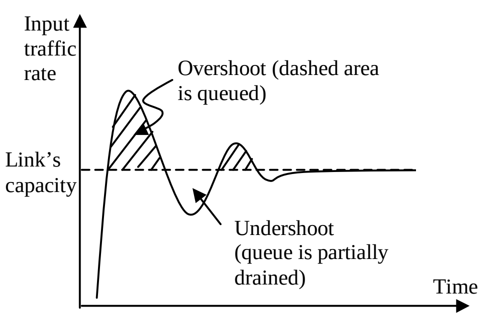

Figure 3: The dynamics of the efficiency controller

```
On packet arrival do
    input_traffic += packetsize;
    If (queue < temporary_min_queue)
        temporary_min_queue = queue

On Te timeout do
    SBW = capacity – input_traffic / Te
    input_traffic = 0
    min_queue = 0.3 × temporary_min_queue + 0.7 × min_queue
    temporary_min_queue = queue

On Tc timeout do
        BTA = max (0, k1 × SBW × Tc - k2 × min_queue)
        BTF = max(0, - k1 × SBW × Tc +k2 × min_queue)
```

Figure 4: Pseudo-code of the efficiency controller

We can achieve efficiency by dividing the difference between BTA and BTF into small chunks that we allocate to single packets as positive or negative feedback. Since EC deals only with the aggregate behavior, it doesn't care which packets get the feedback and by how much each individual flow changes its cwnd. All EC requires is that the total traffic increases by BTA per T_c or decreases by BTF per T_c. How exactly we divide the feedback among the packets (and hence the flows) only affects fairness, and so is the job of the fairness controller.

---

[7] k_1 and k_2 are independent of the number of flows because the dynamics of the aggregate under PCP are independent of the number of flows (i.e., every T_c, the aggregate increases by BTA and decreases by BTF regardless of the number of flows.) They are independent of the capacity of the link because the capacity plays the role of a reference signal in the traditional control theory framework.

3.6.5 Fairness Controller (FC)

FC's goal is to apportion positive and negative feedback to individual packets to achieve fairness. FC relies on the same principle TCP and similar protocols use to converge. In [3], Bansal and Balarishnan generalized TCP-style control to a larger class called *binomial* congestion control protocols. Such protocols increase their congestion window inversely proportional to a power *k* of the current window, and decrease by a power *l* proportional to the current window, as illustrated by Equation 7.

$$\begin{aligned} I:&\quad w(t+rtt) = w(t) + \alpha / w(t)^k; \quad \alpha > 0 \\ D:&\quad w(t+rtt) = w(t) - \beta\, w(t)^l; \quad 0 < \beta < 1 \end{aligned} \tag{7}$$

In TCP, $k = 0$ and $l = 1$. The authors of [3] have argued that any protocol with $k > 0$ and $l > 0$ converges to fairness.

FC converges to fairness by allocating BTA to flows proportionally to 1/cwnd and allocating BTF to flows proportionally to their cwnd. This corresponds to a binomial protocol with $k = l = 1$, though PCP itself is not binomial. [8]

From the perspective of the router it is easier to compute the per-packet positive and negative feedback separately and compute the per-packet feedback as: *H_feedback = positive_feedback – negative_feedback.*

First, we describe FC's computation of the per-packet positive feedback. For a flow to increase its cwnd every rtt proportionally to 1/cwnd, the sum of the positive feedback it receives in an rtt should be proportional to 1/cwnd. Every rtt the flow sends a cwnd of packets, and so positive feedback per packet should be proportional to $1/cwnd^2$. Further, we allocate positive feedback over a control interval T_c, not over at rtt. Yet, packets from small-rtt flows show up more often at the router, so to prevent them getting more than their fair share, the per-packet positive feedback should be proportional to (H_rtt / T_c).

The design so far causes the congestion window of different flows to converge. However, to be fair, it is the throughput of flows that should converge. Since throughput = cwnd /rtt, the per-packet positive feedback should be scaled by the rtt of the flow. Hence, $positive\_feedback \propto 1/H\_cwnd^2 \times H\_rtt^2 / T_c$. Moreover, the sum of the positive feedback allocated to all packets in a T_c should be BTA. From this, we see that the per-packet positive feedback should be:

$$positive\_feedback = \eta \, \frac{BTA}{T_c} \left(\frac{H\_rtt}{H\_cwnd} \right)^2; \quad \eta = \frac{T_c}{\sum\limits_{packets\ in\ T_c} \left(H\_rtt / H\_cwnd \right)^2} \approx \frac{T_e}{\sum\limits_{packets\ in\ T_e} \left(H\_rtt / H\_cwnd \right)^2},$$

where η is a scaling parameter estimated once every T_e.

Next, we describe the computation of the negative feedback. The per-rtt negative feedback is proportional to cwnd. Thus, the per-packet negative feedback is independent of its flow's cwnd. As the sum of the negative feedback in a control interval should be BTF:

$$negative\_feedback = \frac{BTF}{N_p},$$

where N_p is the number of data (not ack) packets in a control interval, computed by counting the packets in an estimation interval and scaling the count by T_c/T_e. Note that with negative feedback we don't need to compensate for bias against long rtts as we did with positive feedback, because a multiplicative decrease does not change the fairness. For example, when two flows with equal throughput and different rtts halve their congestion windows, they still have equal throughput.

The FC, as described so far, does not guarantee convergence to fairness because, when the input traffic rate equals the link capacity, BTA and BTF are both zero. The positive and negative feedback both become zero as well, and the congestion windows of the different flows traversing the link stop changing irrespective of whether their current values are fair or not.

To guarantee convergence to fairness we use a concept we call *bandwidth shuffling*. This is the simultaneous allocation and de-allocation of bandwidth such that the total input traffic rate does not change yet the throughput of each individual flow changes gradually to approach the flow's fair share. We define BTS (Bits To Shuffle) as:

$$BTS = k_3 \times (capacity - SBW) \times T_c; \quad k_3 = 0.1.$$

---

[8] A binomial with $k = l = 1$ is not TCP-friendly; however, this is not relevant because PCP is not a binomial protocol. PCP can be made TCP-friendly as we describe in Section 4.2.

Every control interval we shuffle 10% of the traffic. The per-packet positive and negative feedback become:

$$positive\_feedback = \eta \; \frac{BTA + BTS}{T_c} \left(\frac{H\_rtt}{H\_cwnd} \right)^2; \qquad \eta = \frac{T_e}{\sum_{packets\ in\ T_e} \left(H\_rtt / H\_cwnd \right)^2} \tag{8}$$

$$negative\_feedback = \frac{BTF + BTS}{N_p}, \tag{9}$$

At packet departure, the router computes the appropriate positive and negative feedback and sets *H_feedback* to the difference. However, it should be emphasized that the router never allocates more than BTA+BTS and never frees more than BTF+BTS in any control interval. We define TPF and TNF as the total positive and negative feedback. At the T_c timeout, TPF is set to BTA+BTS and TNF is set to BTF+BTS. Whenever a feedback is given, the router reduces TPF by the positive feedback and TNF by the negative feedback. If the router runs out of TPF (or TNF) it stops giving positive (or negative) feedback until the next T_c timeout.

At the end of this section, we note two points. First, the exact control laws used by both EC and FC can be changed without changing the framework. The equations above for computing feedback are not the only equations that would provide efficiency and fairness. For example, as in TCP, FC might use additive-increase positive feedback and multiplicative-decrease negative feedback and still converge. Such a design was not adopted because it takes longer to reach fairness.

Second, PCP is fairly robust to estimation errors. For example, we estimate the value of η every T_e and use it as a prediction of the value of η during the following control interval. If we underestimate η, we will fail to allocate all of BTA+BTS in the current control interval. The bandwidth we fail to allocate will appear in our next estimation of SBW and will be allocated (or partially allocated) in the following T_c. Thus, in every control interval, a portion of the spare bandwidth is allocated until none is left. Since our underestimation of η causes reduced allocation, the convergence to efficiency is slower than if our prediction of η had been correct. Yet the error does not stop PCP reaching full utilization. A similar argument can be made about other estimation errors; they mainly affect the convergence time rather than the correctness of the controllers.[9]

3.6.6 Extending EC and FC to Multi-Hop Paths

On multi-hop paths, we want each flow to react to feedback sent by its bottleneck. This could be achieved if each router along the path overwrites the feedback only when the modification reduces the congestion window of the flow. That is, *H_feedback = min (H_feedback, positive_feedback – negative_feedback).* (10)

The analysis of EC and FC so far describes the behavior of PCP at the bottleneck. However, the behavior of PCP over non-bottleneck links is slightly more complex and can best be described using the example in Figure 5. Two flows, PCP_1 and PCP_2, share a non-bottleneck link L_1. In addition, PCP_2 crosses a congested link downstream, L_2, which limits its cwnd. Initially link L_1 is underutilized and Router 1 allocates the spare bandwidth to both flows. The positive feedback given to PCP_1 increases its cwnd further; but positive feedback given to PCP_2 is reduced by the downstream router and has little or no impact. This, by itself, is not problematic because any feedback not used by PCP_2 turns into a spare bandwidth that Router 1 will discover in the next T_e and reallocate in the next T_c. Gradually, PCP_1's cwnd increases and becomes much larger than PCP_2's cwnd. Yet, because the router allocates spare bandwidth to flows inversely proportional to their congestion windows, as PCP_1's cwnd becomes much larger than PCP_2's cwnd, its allocated portion of the spare bandwidth gets very small. Thus, gradual convergence to optimal utilization at non-bottleneck links slows down. In fact, if PCP_1 and PCP_2's congestion windows become very different, convergence becomes extremely slow.

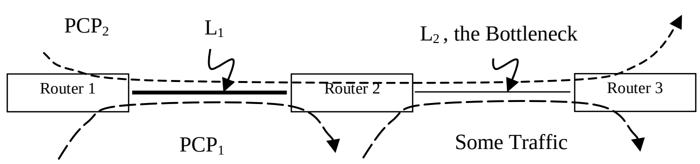

Figure 5: PCP on a multi-hop path

[9] In fact, there is one type of error that prevents convergence to complete efficiency, which is the unbalanced allocation of BTS in negative and positive feedback. If by the end of T_c, we allocate all of BTS as negative feedback but we fail to allocate all of BTS as positive feedback, then the shuffling might prevent us from reaching full link utilization. Yet, the effect of shuffling on utilization stays minor because BTS is less than 10% of the traffic.

One approach to reach utilization quickly is to indicate to Router 1 that PCP_2 crosses a bottleneck and therefore should not be given any positive feedback. To do so, we use two header bits: *H_B1* is set by any router along the path when SBW is less than 10% of the link's capacity.[10] *H_B2* is set by the sender when the last received ack had *H_B1* set, or when the sender does not want to increase its congestion window for any reason. Routers use *H_B2* to distinguish between bottlenecked traffic (BT), which includes all packets with *H_B2 = 1*, and non-bottlenecked traffic (NBT), which includes all packets with *H_B2 = 0*. Every T_e the router estimates the number of NBT flows. At T_c timeout, it checks whether all flows are bottlenecked or not. If all traffic is bottlenecked then, during this T_c, the router treats all packets according to Equations 8 and 9 above. If some traffic is not bottlenecked then the router gives the positive feedback only to that traffic. The router still shuffles all traffic and distributes its negative feedback to both NBT and BT packets. From the router's perspective, whenever there are bottlenecked and non-bottlenecked flows, it is wiser to start by giving the positive spare bandwidth to the non-bottlenecked flows and let shuffling take care of establishing fairness. Although this design does not guarantee convergence to 100% utilization at non-bottleneck links, it does converge quickly to at least 80% utilization. This bound comes from two factors. First, 10% of the bandwidth is always shuffled and will be mostly given to flows with smaller cwnds, which are usually the bottlenecked flows. Second, routers set *H_B1* in the packets when their SBW < 10% the capacity of the link. Due to delay and the non-alignment of T_e with T_c this bound is approximate, but experiments in Section 5 show it is fairly accurate in practice. Allocation of feedback to BT and NBT packets is summarized in Figure 6.

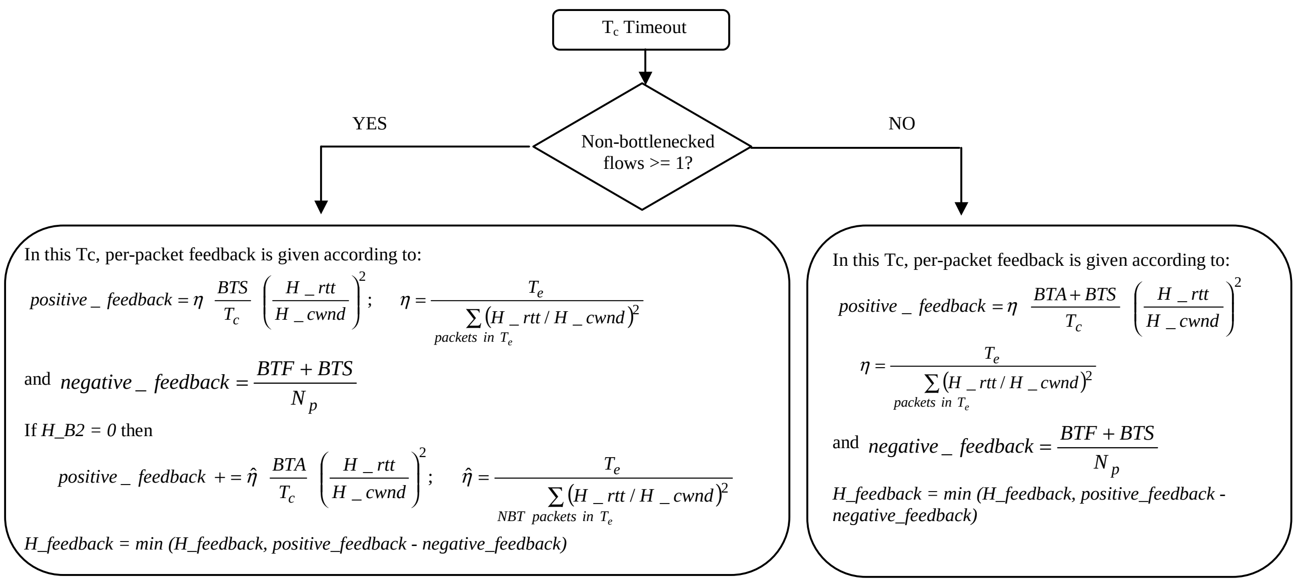

Figure 6: Computation of per-packet feedback.

4 Coexistence with TCP

While a new network design might only use PCP for congestion control, we believe PCP should also be incrementally deployable on the Internet. In this section we will demonstrate one way that PCP and TCP flows, PCP routers, and conventional routers might coexist peacefully in the same network.

4.1 Starting a PCP Connection

A PCP-enabled sender initiating a connection checks whether the receiver and the routers along the path are PCP-enabled. In case they are not, it falls back to its conventional protocol, which could be TCP, Congestion Manger [2], or UDP. [11] We describe a scheme that falls back to TCP when the receiver or the routers are not PCP-enabled.

To ensure the receiver is PCP-enabled, the sender starts the connection by sending a TCP SYN packet with a special PCP option. If the receiver is not PCP-enabled it ignores the PCP option and the connection continues using TCP. If the receiver is

[10] Specifically, *H_B1* is set when $SBW^* < 0.1 \times capacity$, where SBW^* is a smoothed SBW computed by $SBW^* = 0.7 \times SBW^* + 0.3 \times SBW$.

[11] In fact, it is potentially possible to build a PCP connection even when some of the routers along the path are not PCP-enabled. To do so, the sender updates its cwnd according to the precise feedback in the congestion header. At the same time, the sender uses a TFRC-like approach [7] to compute the sending rate required by the non-PCP routers. If this sending rate is smaller than (cwnd/rtt) the sender sets cwnd to the TFRC-like sending rate multiplied by its rtt.

PCP-enabled, it sets the PCP option in its acknowledgement of the SYN packet, in which case the connection continues as a PCP connection if the path is PCP-enabled. Yet, because a TCP packet has no congestion header, the sender would not get any precise feedback in the first rtt even if the connection continues as a PCP connection. We fix this problem by having the sender follow the TCP SYN packet by a PCP control packet, which is a small packet that contains only an IP and a congestion header, has no data and no sequence number, and is used to collect the precise feedback that the TCP SYN packet misses.

To ensure the path is PCP-enabled, we augment the congestion header with a PCP Time to Live field (called the PTL field). PTL is decremented by every PCP-enabled router along the path. Conventional routers (not PCP-enabled) do not modify PTL because they cannot distinguish the congestion header from data or higher level protocols. When the packet reaches the receiver (or the ack reaches the sender) it checks whether PTL equals TTL, in which case the path is PCP-enabled. If the two fields are not equal then the path is not PCP-enabled and the connection falls back to TCP. Note that this approach allows the sender to ensure that the path is initially PCP-enabled and that it continues to be so.

4.2 A PCP-Router Managing a Mixture of TCP and PCP flows

This section extends the design of a PCP router to handle a mixture of PCP and TCP flows while ensuring that PCP flows are TCP-friendly. The router distinguishes PCP traffic from non-PCP traffic and queues them differently.[12] TCP packets are queued in a conventional RED queue, which we call the T-queue. PCP flows are queued in a PCP-enabled queue (described in Section 3.6) called the P-queue. To be TCP fair, the router should process packets from the two queues such that the average throughput observed by TCP flows in the T-queue equals the average throughput observed by PCP flows in the P-queue. This is achieved by processing packets from the two queues using weighted-fair queuing where the weights are dynamically updated and converge to the fair share of TCP and PCP. The weight update mechanism uses the T-queue drop rate *(p)* to compute the average congestion window of the TCP flows in the T-queue. The computation uses a TFRC-like approach and is based on TCP's throughput equation below [7].

$$\overline{cwnd}_{TCP} = \frac{s}{\sqrt{\frac{2p}{3}} + 12p\sqrt{\frac{3p}{8}} \times (1 + 32p^2)} \ ,$$

where s is the average packet size. Every estimation interval, the weights are updated as follows:

$$w_T = w_T + k_4 \times \frac{\overline{cwnd}_{PCP} - \overline{cwnd}_{TCP}}{\overline{cwnd}_{PCP} + \overline{cwnd}_{TCP}}; \qquad w_P = w_P + k_4 \times \frac{\overline{cwnd}_{TCP} - \overline{cwnd}_{PCP}}{\overline{cwnd}_{PCP} + \overline{cwnd}_{TCP}} \ , \qquad (11)$$

where k_4 is a small constant in the range (0,1), and w_T and w_p are the T-queue and the P-queue weights. Equation 11 ensures that the weights are updated to decrease the difference between TCP's and PCP's average congestion windows. When the difference becomes zero the weights stop changing and stabilize.

The final step in designing the PCP-extended router is to prevent TCP flows from starving PCP flows. PCP starvation could happen because, as long as there is no drop, TCP keeps dumping new packets into the system even when SBW < 0, a behavior that drives SBW to an even larger negative value. PCP reacts to this by giving up its bandwidth in an attempt to bring SBW back to zero. Yet TCP, which keeps increasing its window until there is a drop, would grab any bandwidth that PCP frees and would keep driving SBW below zero. The solution to this problem is to prevent PCP from giving up its fair share of the spare bandwidth to TCP, which could be done by modifying BTA and BTF as follows:

$$PSBW = w_p \times capacity - (PCP\ input\ traffic / T_c)$$

$$BTA = \max\ (0,\ k_1 \times T_c \times \max\ (SBW,\ PSBW) - k_2 \times \min\_queue)$$

$$BTF = \max\ (0,\ -k_1 \times T_c \times \min\ (SBW,\ PSBW) + k_2 \times \min\_queue)\ ,$$

where PSBW is PCP's fair share of the spare bandwidth. The above equation would allow PCP to grab any positive spare bandwidth in the system; however, when SBW becomes negative PCP frees bandwidth only until it reaches its fair share. Although we allow PCP to grab the spare bandwidth not used by TCP this should not create unfairness because PCP will give up this bandwidth as soon as TCP has enough traffic to use it. Moreover, our implementation of fair queuing remembers that PCP got slightly more than its fair share in the recent past and gives slightly more to TCP.

[12] Either using a bit in the IP header or using the transport protocol field in the IP header.

Finally, note that the above design gives PCP and TCP flows the same average congestion window yet their average throughput might still be different if the round trip time is different. However, given that TCP itself is not fair to flows with different round trip times, we think that providing fairness of the average congestion window is acceptable.

5 Performance

We evaluate PCP using simulation in various settings. Our simulations are run in the *ns* simulator, which we have extended with a PCP module. In all of our simulations we use $k_1 = 0.4$, $k_2 = 0.5$, $k_3 = 0.1$, and $T_c = 50$msec. By fixing the values of the control parameters, we demonstrate that these values perform well regardless of the link bandwidth, the number of flows, the rtts, and the simulated topology.

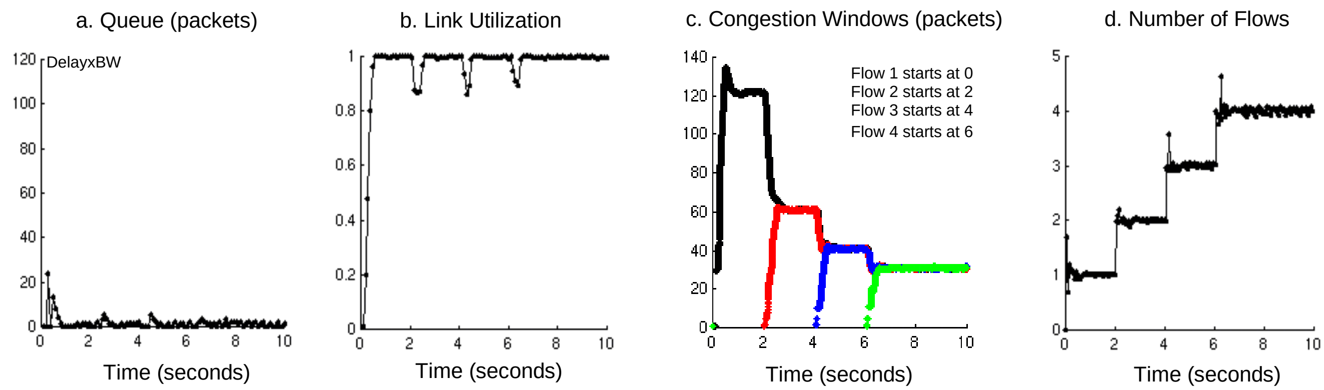

Figure 7: Four flows sharing a bottleneck; the flows' starting times are spaced out by 2 seconds; Delay-BW product = 120 packets; simulation showed 0 drops;

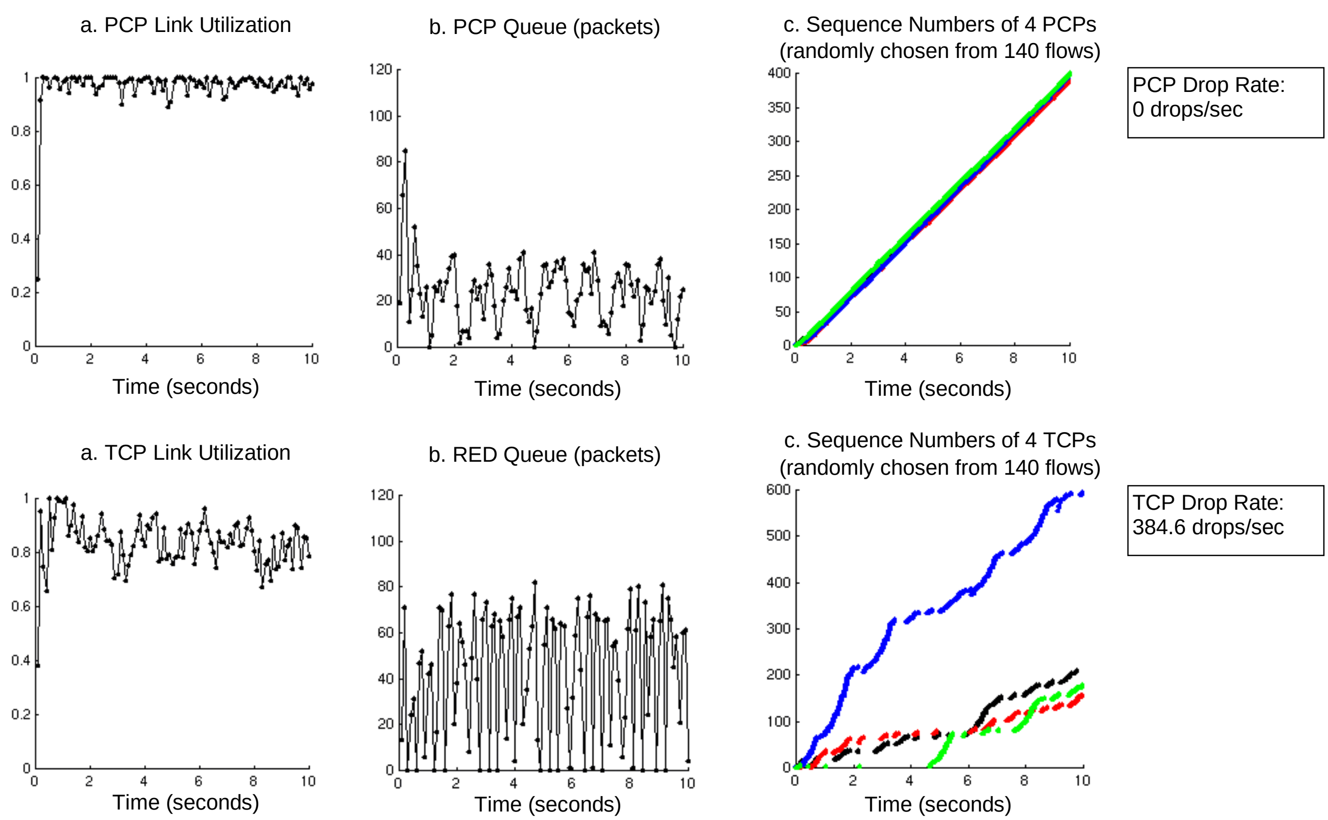

Figure 8: Comparison between PCP and TCP in an environment with many flows and 2-way traffic:
70 flows in each direction; one bottleneck; delay-bandwidth product = 120 packets

Basic Behavior: First, we examine the dynamics of PCP in a simple setting and investigate the accuracy of its estimators. In this simulation, four flows share one 24Mbits/sec link whose round trip delay is 40msec. The flows start at times 0, 2, 4, and 6 seconds. Figures 7-a and 7-b show that the efficiency controller reaches full utilization in only a few rtts, while maintaining a minimal queue and with no drops. Figure 7-c shows how the congestion window of the active flows rapidly reduces as new flows start so that all flows have almost the same cwnd at any point in time. Finally, 7-d shows the high accuracy of our estimator of the number of flows. Other PCP estimators have similar accuracy.

Comparison with TCP: In this simulation, we compare PCP's behavior, in environments with many flows and 2-way traffic, to TCP's behavior in separate simulations with similar conditions. We use TCP Reno over a RED queue whose parameters we tuned to $min_{th} = 36$, $max_{th} = 72$, $max_p = 0.33$, and $w_p = 0.002$ in order to achieve the best performance. The setting involves a single bottleneck (24Mbits/sec, 40msec) shared by 70 flows in each direction. Figure 8 shows that PCP achieves near perfect utilization (95%-100%) while maintaining a small queue and a zero drop rate. TCP, on the other hand, exhibits worse utilization and poor fairness.

Impulse Response: Here we examine the impact of a sudden change in the traffic on both PCP's and TCP's performance. The simulation uses one 80Mbits/sec link with round trip delay of 40 msec. Four flows are started at time 0 and left to stabilize. At t=5 sec, 40 additional flows are started simultaneously, left to stabilize, then stopped simultaneously at t=10. The top part of Figure 9 shows how PCP absorbs this burst of 40 flows without dropping any packets. Moreover, when these flows terminate their transfers, PCP grabs the spare bandwidth quickly and maintains its high utilization. In particular, 9-c shows that as the new flows start, the congestion window of the permanent PCP flows decreases rapidly to the new fair share (around 9 packets) and when these 40 flows terminate their transfers, cwnd ramps up to its previous value. On the other hand, the introduction of 40 new flows causes TCP's utilization to drop below 5%; furthermore, after the 40 flows terminate their transfers, the remaining TCP flows waste the bandwidth by taking almost 100 rtts to ramp up to full utilization.

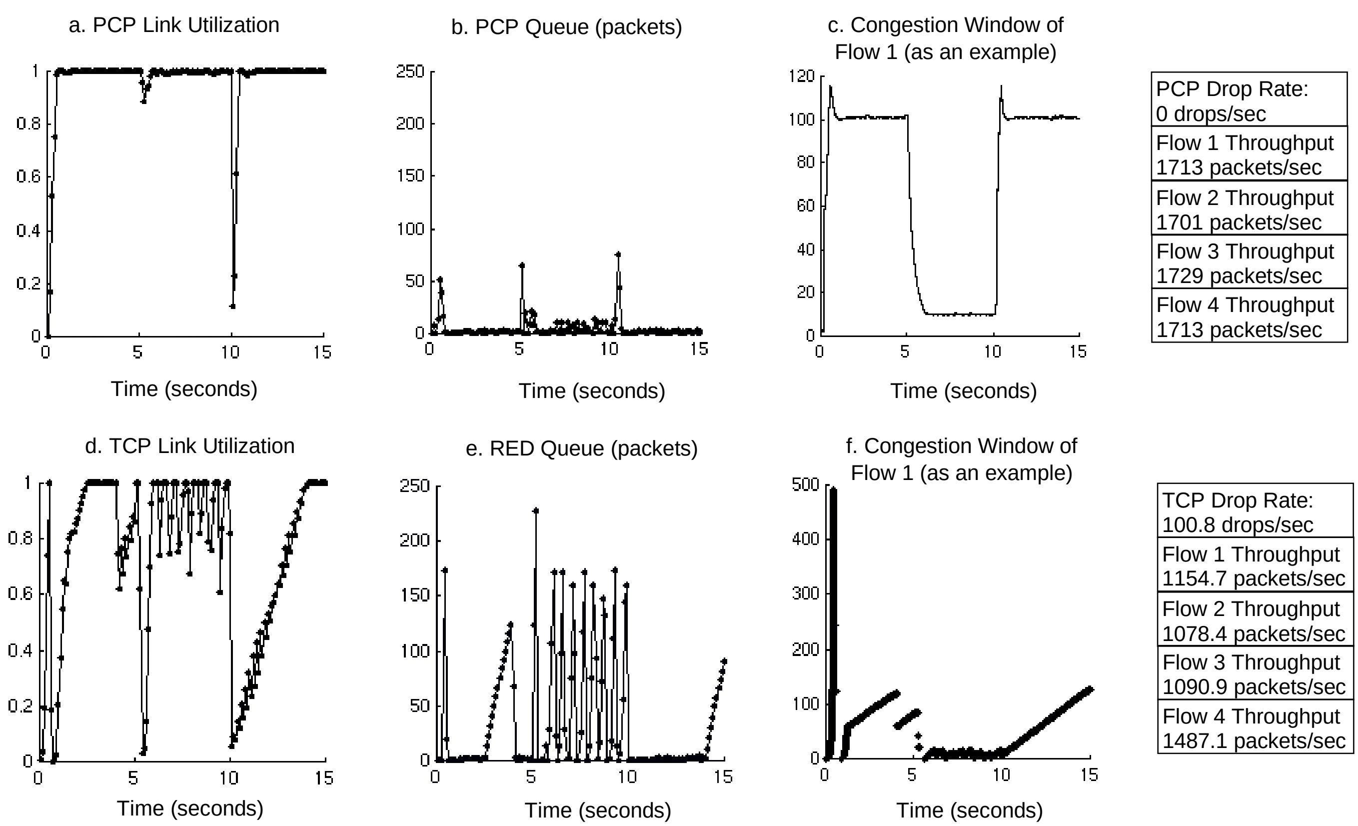

Figure 9: Comparison between PCP and TCP reaction to a sudden increase/decrease in the traffic: 4 long flows are started at t=0 and remain until the end of the simulation. At t=5, 40 additional flows are started, left to stabilize, and then stopped at t=10.

Dynamic Environment: In this simulation, we model the effects of competing web-like short flows and ftp-like longer flows. 35 short flows of 30 packets duration share a common bottleneck (24Mbits/sec, 40msec) with 35 long flows. There are also some flows along the reverse path. Whenever, a short flow finishes its transfer, a new short flow is started so that the number of short flows is the same at any time during the simulation. Figures 10-a and 10-b show that even in such a highly dynamic environment, PCP maintains near optimal utilization, small queues, and a zero drop rate. Figure 10-c, a histogram of the ratio of throughput of the short flows to the average throughput of all traffic, shows that despite their short transfers, web-like flows get around 90% of their fair share of the bandwidth.

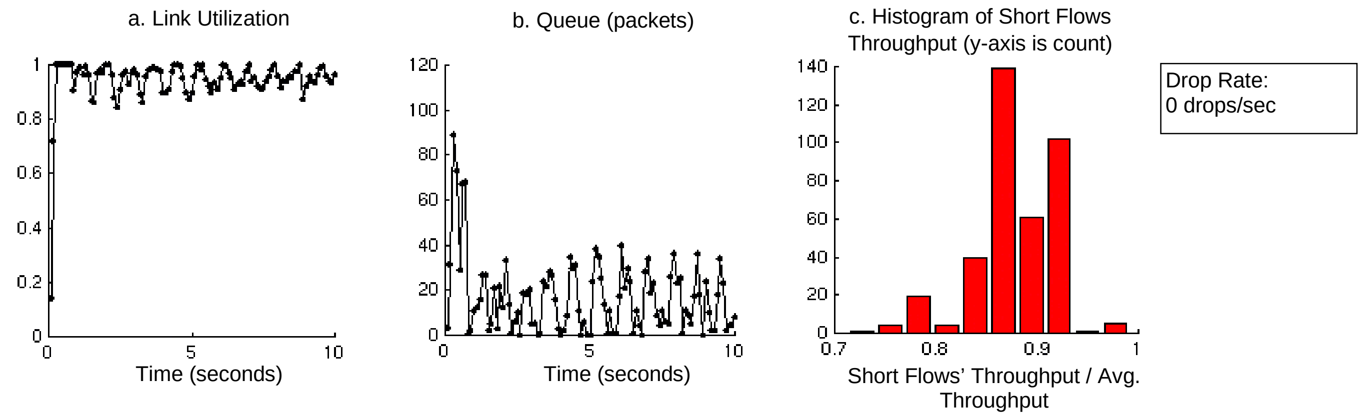

Figure 10: PCP's performance in dynamic environments with many short flows;
35 short flows of 30 packets each are sharing a common bottleneck with 35 long flows.

Fairness and Utilization as Functions of Bottleneck's Bandwidth: In this experiment we show that PCP's efficiency and fairness are independent of the bottleneck's bandwidth. We simulate 10 flows sharing a bottleneck whose capacity varies from 1 to 128Mbits/sec. In all of these runs, as in all of the simulations presented in this section, we use $k_1 = 0.4$, $k_2 = 0.5$, and $k_3 = 0.1$. Figure 11 shows that PCP maintains almost optimal utilization and fairness across a wide range of link bandwidths without requiring careful tuning of parameters.

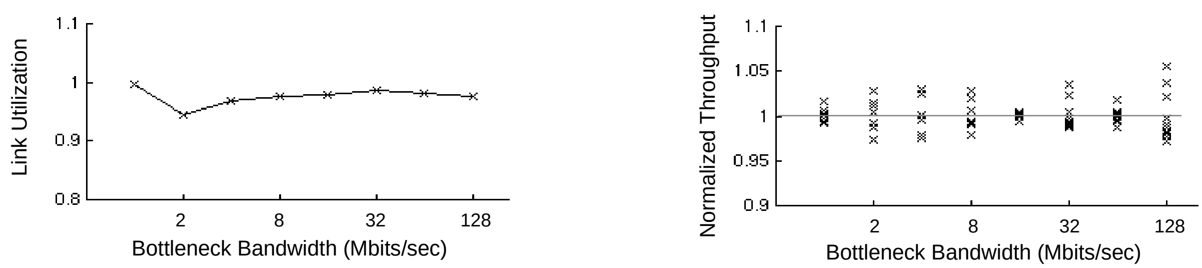

Figure 11: PCP's efficiency and fairness are fairly independent of the bottleneck bandwidth. Left: utilization as a function of link bandwidth. Right: throughput of ten flows divided by the average throughput and plotted as a function of link bandwidth.

Utilization and Queue Size as Functions of Burstiness (Number of 2-Way Flows): When the aggregate input traffic is extremely bursty it becomes difficult for EC to distinguish between persistent and transient queues. In this case, EC might drain some of the transient queue causing reduced utilization. The burstiness of the aggregate is caused by ack compression and depends on the number of 2-way flows. It is worst when there is a single flow in each direction with a cwnd of half the delay-bandwidth, which the sender transmits in one burst. On the other hand, when there are many 2-way flows, though each flow could be extremely bursty, the aggregate's burstiness is reduced. To examine this, we simulate a single link (24Mbits/sec, 40msec) shared by n flows in each direction, where n varies between one and 85. Figure 12 shows that, as expected, the worst utilization (76%) happens with one flow in each direction; however, as soon as there are five flows in each direction the utilization becomes above 90%.

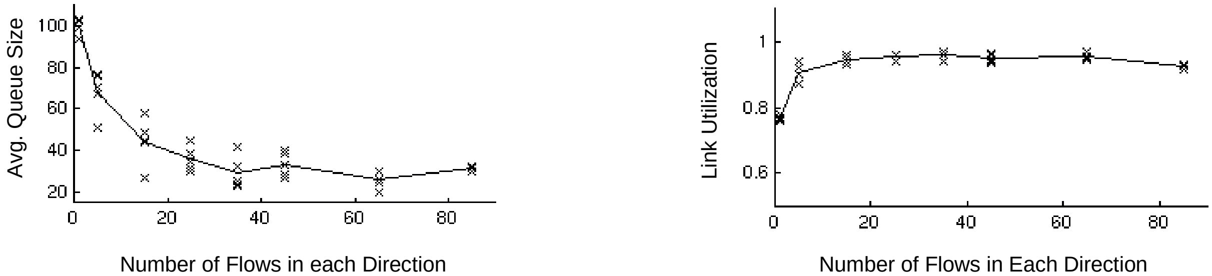

Figure 12: EC's performance as a function of the aggregate traffic burstiness.

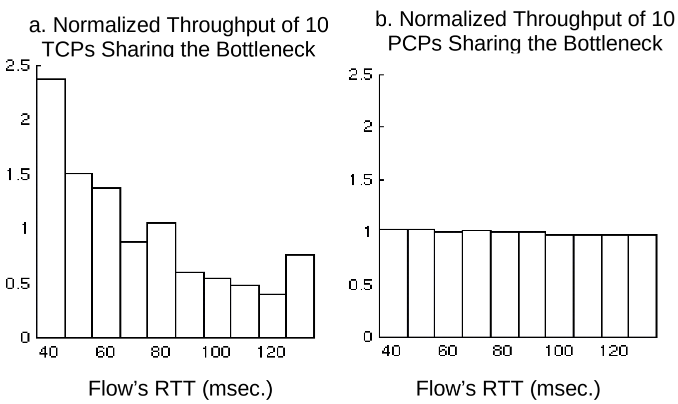

Figure 13: Comparison of PCP and TCP Fairness to different RTTs; (Throughputs are normalized to the average.)

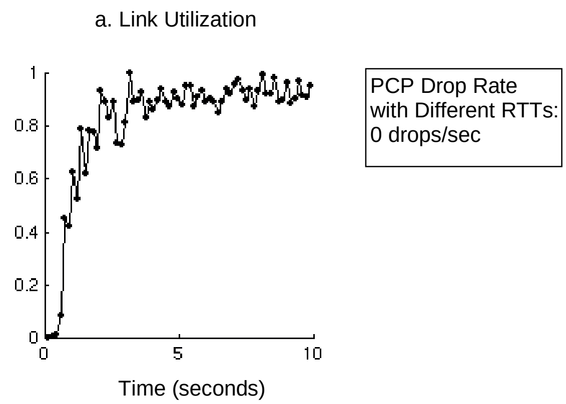

Figure 14: The effect of flows with different RTTs on PCP performance.

Fairness to Longer RTTs: In this simulation, ten flows whose round trip times are 40, 50, 60, 70, 80, 90, 100, 110, 120, 130msec share a common bottleneck of 24Mbits/sec. Figure 13 compares the throughput these flows obtain under TCP and PCP. It shows that while TCP is unfair to flows with longer rtt, PCP gives them almost their fair share (90%-95%). Figure 14 examines the effect of different rtts on PCP's efficiency. It shows that although the utilization is slightly reduced, it stays near optimal. Note that in this experiment (as in all the experiments in this section), T_c is set to 50msec. This shows that PCP is robust to values of T_c that are significantly less than the average rtt of the system. Other experiments (not shown here), indicate that setting T_c to twice the average rtt has little impact on the performance.

Multi-hop Paths: In Section 3.6.6, we noted that when some of the flows traversing a non-bottleneck link are restricted by a downstream bottleneck, the utilization of the non-bottleneck link might become less than optimal. We also noted that as the ratio of cwnd of the non-bottlenecked flows to cwnd of the bottlenecked flows becomes larger, the utilization of the non-bottleneck link decreases, yet stays above 80%. The next simulation uses the topology in Figure 15, which shows two 24Mbits/sec links. There are *n* flows that cross only the upstream hop, *m* flows that cross only the downstream hop, and *r* flows that cross both hops. We always choose $n < m$ so that the downstream link is the bottleneck for flows that cross both links. At t = 0, we start both the flows traversing the 2-hop path and the ones traversing only the first hop. At t = 5, we start the *m* flows that cross only the downstream link. Figure 15 illustrates the utilization of the upstream non-bottleneck link for different values of *n, m, and r*. It shows that even when cwnd of the non-bottlenecked flows is two orders of magnitudes larger than cwnd of the bottlenecked flows, the utilization of the non-bottleneck link stays above 80%. Although it would be beneficial to simulate longer and more complex paths, we believe that the complexity of the path does not affect the performance.[13]

[13] Note that longer paths do not cause error accumulation because both the feedback and *H_B1* are overwritten rather than modified by a congested router.

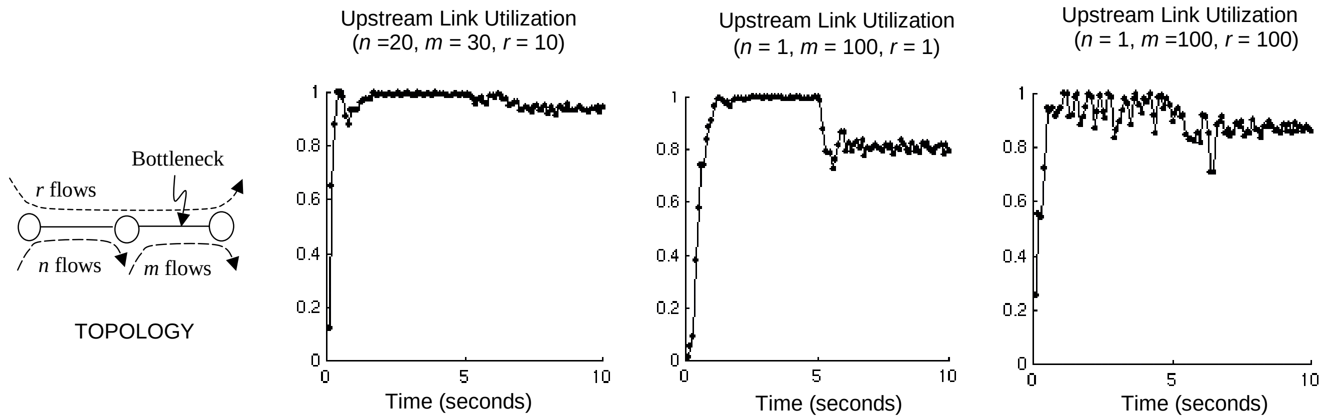

Figure 15: PCP's utilization at non-bottleneck links; traffic at the downstream bottleneck is started at t = 5.

TCP-friendliness: Figure 16 illustrates the throughput of *n* TCP flows and *m* PCP flows sharing a bottleneck averaged over 30sec. The graph shows that in most cases the difference between the average throughput of competing TCP and PCP flows is less than 20%. It also shows that when there were three TCP flows competing with 30 PCP flows, two TCPs got significantly more than their fair share. A close inspection of the sequence numbers of these two TCPs revealed that they opened their cwnds to very large values during slow start. After slow start, they continued using a large window for a while because the RED drop rate with only 3 TCPs in the system is very low, which causes the weight update to be slow. This doesn't show that our mechanism is incorrect but shows that TCP itself is hardly fair when the drop rate is small, such as when a small number of flows competes for large bandwidth. This is demonstrated by the fact that the third TCP flow in this run achieved exactly the same throughput as the PCP flows.

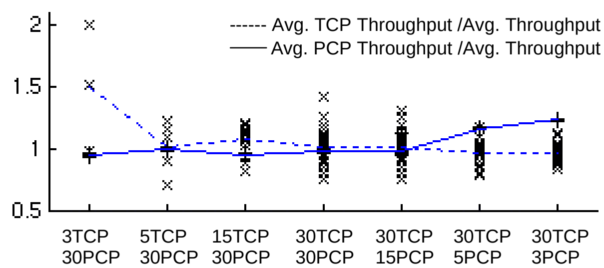

Figure 16: Normalized throughput of competing PCP and TCP flows .
The measurements are for different numbers of PCP and TCP flows.

6 Related Work

Previous related work lies in two areas.

ABR Congestion Control: The ATM Forum has adopted an explicit end-to-end rate-based flow-control protocol (EERC) for controlling Available Bit Rate (ABR) traffic [11]. In EERC, each source periodically sends special resource management (RM) cells. These cells include a field called the explicit rate field (ER) which the source initializes to its desired sending rate. Each switch along the path computes the source's min-max fair rate. If the min-max fair rate is smaller than the value in the ER field, the switch sets ER to the min-max fair rate. The receiver relays the RM cells back to the sender, which then adjusts its rate to that indicated in the RM cell. Note that EERC does not specify how the min-max fair rate is computed. A few algorithms have been proposed for computing the min-max fair sending rate [1,4,8,9,12].

PCP resembles ABR congestion control in several ways. First, they both send explicit and precise feedback. Second, in both cases, the framework is independent from the exact control law used so that the control law can be changed while maintaining the framework. The differences between the two approaches are crucial. First, in contrast to ABR control protocols where the system indirectly converges to good efficiency by allocating to each flow its fair share, a PCP router explicitly computes the traffic increase needed to achieve efficiency or the decrease needed to prevent congestion, and lets the system converge to fairness through bandwidth shuffling. Thus, PCP provides tighter control and a greater stability. Second, most ABR flow control protocols maintain per-flow state at the switches [4,8,9,12] whereas PCP does not keep any per-flow state in routers. Finally, PCP is built on top of TCP. Its window adjustments have finer granularity, but when a drop occurs it falls back to a TCP-like behavior.

Internet Congestion Control: PCP builds on the experience learned from TCP and other congestion control related research [3,5,6,7,10,13,14]. However, the design of PCP is most closely related to Explicit Congestion Notification (ECN) and Core Stateless Fair Queuing (CSFQ).

With ECN [13], a RED queue signals congestion to the sources by marking their packets rather than dropping them. Thus, ECN changes the congestion feedback from implicit binary feedback to explicit binary feedback. PCP can be regarded as an extension of the ECN proposal since it makes the feedback both explicit and precise. PCP is similar to ECN in that it does not maintain any per-flow state at the routers. Further, both ECN and PCP require modifying the end systems and the routers. However, PCP has completely different dynamics and performance characteristics than TCP with ECN.

In CSFQ [14], edge routers estimate the flows' rates and insert the estimates in the packets' header. Core routers estimate the fair rate and drop packets preferentially based on the difference between the estimated rate and the fair rate. PCP builds on CSFQ's idea of pushing the per-flow state to the edges of the network, but by pushing the state all the way to the sender, PCP can use simpler and more accurate estimators. CSFQ has the advantage of being able to force malicious and UDP flows to use almost their fair rate. PCP can be extended to police malicious and UDP senders but this extension is left for future work. Another advantage of PCP over CSFQ is its better performance especially over short time scales.

7 Discussion

This paper introduces PCP, a congestion control protocol in which the routers tell the senders the precise amount by which to increase or decrease their congestion windows. The key idea of PCP is decoupling utilization control from fairness control. This allows us to reformulate the problem of controlling utilization in the context of traditional control theory, where the signal needing control is the input traffic to a link, and the reference signal is the link's capacity. The first step in this reformulation is to hide the burstiness of Internet traffic to allow the utilization controller to perceive it as a smooth signal. To achieve this, the router estimates all of its parameters over the mean rtt of the system and looks at the persistent queue instead of the instantaneous queue length. The second step is to discretize the system, which we do by introducing the concept of a control interval. At the beginning of each control interval, the router decides on the feedback that should be sent to the sources sharing the link and does not update this decision until the next control interval. Given the above, the control problem can be solved by feeding the error to the system's input, which translates directly to the router computing the difference between the input traffic rate and the link's capacity and asking the sources to increase or decrease proportionally to this difference. Finally, we extend this system to drain any queue that might build up, and to work properly over multi-hop paths.

The idea we use to control fairness is also conceptually simple. It comes from recognizing that binomial protocols [3] such as TCP converge asymptotically (as $t \to \infty$) to fairness as was demonstrated by Chiu and Jain in their classic paper [5]. However, this convergence is too slow and expensive because it is tied to the occurrence of drops, which are - and should be- rare events. Our idea of *bandwidth shuffling* allows faster convergence, independently of packet drops. The decoupling allows us to use different control laws for controlling fairness and utilization. Furthermore, the congestion header communicates each flow's state to the routers and returns the feedback to the sources allowing the above to be done without any per-flow state at the router.

Simulation results show that PCP systematically succeeds in achieving near optimal utilization, queue size, and drop rate. One reason why PCP significantly improves over TCP and RED is precise feedback, which enables a PCP router to fine tune the sources' sending rate to home in on the capacity of the link. Yet, equally important for PCP's success is that in addition to controlling the window adjustments of each flow, a PCP router controls the dynamics of the aggregate traffic by controlling the sum of window increase/decrease performed by sources sharing a link. A RED router, on the other hand, has to cope with the fact that the dynamics of a TCP aggregate depend on unknown and uncontrollable factors such as the number of flows. As a result, it is difficult to tune the RED parameters so that the controller is neither too slow nor too aggressive regardless of the number of flows. In contrast, the choice of good values for PCP's control parameters is stable and independent of the number of flows.

The ability to directly control the dynamics of the aggregate traffic is also an essential difference between PCP and ABR-like congestion control [4,8,12], where a router allocates its computed fair bandwidth share to each flow, ignoring that the computation errors could accumulate in the aggregate and cause it to behave badly.

Examining PCP limitations, we find that PCP attaches a congestion header to every packet generating an additional load on the system. However, this extra load could be justified given PCP's improved utilization. Second, a PCP router is more complex than a drop-tail or a RED router. However, all of the PCP-specific operations have a complexity of *O(1)* and could be performed quickly in specialized hardware. Finally, although the large number of experiments we have conducted show substantial robustness and systematic good performance, our experience with PCP is still small compared to the extensive experience with TCP. Our future research will focus on conducting more extensive experiments over complex topologies.

8 References

[1] Y. Afek, Y. Mansour, and Z. Ostfeld, "Phantom: A Simple and Effective Flow Control Scheme," In Proc. of ACM SIGCOMM, 1996.

[2] H. Balakrishnan, H. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," In Proc. of ACM SIGCOMM, 1999.

[3] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," In Proc. of IEEE INFOCOM, April 2001.

[4] A. Charny, "An Algorithm for Rate Allocation in a Packet-Switching Network with feedback, " Masters thesis, MIT 1994.

[5] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," In Computer Networks and ISDN Systems 17, page 1-14, 1989.

[6] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," In IEEE/ACM Transactions on Networking, 1(4):397--413, August 1993.

[7] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," In Proc. of ACM SIGCOMM, 2000.

[8] R. Jain, S. Fahmy, S. Kalyanaraman, and R. Goyal, "The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks: Part II: Requirements and Performance Evaluation," The Ohio State University, Department of CIS, January 1997.

[9] R. Jain, S. Kalyanaraman and R. Viswanathan, "The OSU Scheme for Congestion Avoidance in ATM Networks: Lessons Learnt and Extensions," In Performance Evaluation Journal, Vol. 31/1-2, December, 1997.

[10] V. Jacobsen and M. Karels, "Congestion Avoidance and Control," In Proc. of ACM SIGCOMM, Stanford, August 1988.

[11] S. Keshav, "An Engineering Approach to Computer Networks," Addison Wesley, 1997.

[12] W. Kin, F. Wong, and D. H. K. Tsang, "A Rate-Based Switch Algorithm with Delay Adjustment for ABR Traffic to Achieve Max-Min Fairness," In IEEE ATM Workshop, October 30 - November 1, 1995.

[13] K. K. Ramakrishnan, S. Floyd, "Proposal to add Explicit Congestion Notification (ECN) to IP," RFC 2481, 1999.

[14] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless Fair Queuing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks," In Proc. ACM SIGCOMM, Vancouver, CA, August 1998.