Internet Congestion Control for Future High Bandwidth-Delay Product Environments

Dina Katabi — MIT-LCS — dk@mit.edu
Mark Handley — ICSI — mjh@icsi.berkeley.edu
Charlie Rohrs — Tellabs — crhors@mit.edu

Abstract

Theory and experiments show that as the per-flow product of bandwidth and latency increases, TCP becomes inefficient and prone to instability, regardless of the queuing scheme. This failing becomes increasingly important as the Internet evolves to incorporate very high-bandwidth optical links and more large-delay satellite links.

To address this problem, we develop a novel approach to Internet congestion control that outperforms TCP in conventional environments, and remains efficient, fair, scalable, and stable as the bandwidth-delay product increases. This new eXplicit Control Protocol, XCP, generalizes the Explicit Congestion Notification proposal (ECN). In addition, XCP introduces the new concept of decoupling utilization control from fairness control. This allows a more flexible and analytically tractable protocol design and opens new avenues for service differentiation.

Using a control theory framework, we model XCP and demonstrate it is stable and efficient regardless of the link capacity, the round trip delay, and the number of sources. Extensive packet-level simulations show that XCP outperforms TCP in both conventional and high bandwidth-delay environments. Further, XCP achieves fair bandwidth allocation, high utilization, small standing queue size, and near-zero packet drops, with both steady and highly varying traffic. Additionally, the new protocol does not maintain any per-flow state in routers and requires few CPU cycles per packet, which makes it implementable in high-speed routers.

1 Introduction

For the Internet to continue to thrive, its congestion control mechanism must remain effective as the network evolves. Technology trends indicate that the future Internet will have a large number of very high-bandwidth links. Less ubiquitous but still commonplace will be satellite and wireless links with high latency. These trends are problematic because TCP reacts adversely to increases in bandwidth or delay.

Mathematical analysis of current congestion control algorithms reveals that, regardless of the queuing scheme, as the delay-bandwidth product increases, TCP becomes more oscillatory and prone to instability. By casting the problem into a control theory framework, Low et al. [22] show that as capacity or delay increases, Random Early Discard (RED) [13], Random Early Marking (REM) [5], Proportional Integral Controller [15], and Virtual Queue [14] all eventually become prone to instability. They further argue that it is unlikely that *any* Active Queue Management scheme (AQM) can maintain stability over very high-capacity or large-delay links. Although their analysis uses Reno TCP, their argument is valid for all current TCP implementations where throughput is inversely proportional to the round trip time (RTT) and the square root of the drop rate. Furthermore, Katabi and Blake [19] show that Adaptive Virtual Queue (AVQ) [21] also becomes prone to instability when the link capacity is large enough (e.g., gigabit links).

In addition to these mathematical models, intuitive reasoning shows that "slow start" might also lead to instability in the future Internet. As capacity increases, the majority of flows become "short" flows which never exit slow start. Flows in slow start increase their rate exponentially, a dramatically unstable behavior. Currently, although the number of short flows is large, most of the bytes are in long-lived flows. Consequently, the dynamics of the aggregate traffic are usually dominated by the additive-increase multiplicative-decrease policy. However, as the fraction of flows in slow start grows, exponential increase may dominate the dynamics of the aggregate traffic, causing instability.

Potential instability is not the only problem facing TCP in the future Internet. As the delay-bandwidth product increases, performance degrades. TCP's additive increase policy limits its ability to acquire spare bandwidth to one packet per RTT. Since the bandwidth-delay product of a single flow over future links may be many thousands of packets, TCP might waste thousands of RTTs ramping up to full utilization following a burst of congestion.

Further, since TCP's throughput is inversely proportional to the RTT, fairness too might become an issue as more flows in the Internet traverse satellite links or wireless WANs [25]. As users with substantially different RTTs compete for the same bottleneck capacity, considerable unfairness will result.

Although the full impact of large delay-bandwidth products is yet to come, we can see the seeds of these problems in the current Internet. For example, TCP over satellite links has revealed network utilization issues and TCP's undesirable bias against long RTT flows [4]. Currently, these problems are mitigated using ad hoc mechanisms such as ack spacing, split connection [4], or performance enhancing proxies [8].

Simulation results similar to those in § 5 support the above argument showing that, regardless of the queuing scheme, TCP's performance degrades significantly as either capacity or delay increases.

This paper develops a novel protocol for congestion control that outperforms TCP in conventional environments, and further remains efficient, fair, and stable as the link bandwidth or the round-trip delay increases. This new eXplicit Control Protocol, XCP, generalizes the Explicit Congestion Notification proposal (ECN). Instead of the one bit congestion indication used by ECN, our routers inform the senders about the degree of congestion at the bottleneck. Another new concept is *the decoupling of utilization control from fairness control*. To control utilization, the new protocol adjusts its aggressiveness according to the spare bandwidth in the network and the feedback delay. This prevents oscillations, provides stability in face of high bandwidth or large delay, and ensures efficient utilization of network resources. To control fairness, the protocol reclaims bandwidth from flows whose rate is above their fair share and reallocates it to other flows.

By putting the control state in the packets, XCP needs no per-flow state in routers and can scale to any number of flows. Further, our implementation (Appendix A), requires only a few CPU cycles per packet, making it practical even for high-speed routers.

Using a control theory framework motivated by previous work [21, 15, 22], we show that a fluid model of the protocol is stable for any link capacity, feedback delay, or number of sources. In contrast to the various AQM schemes where parameter values depend on the capacity, delay, or number of sources, our analysis shows how to set the parameters of the new protocol to constant values that are effective independent of the environment.

Our extensive packet-level simulations show that the performance of TCP over RED, AVQ, REM, or CSFQ degrades substantially as capacity or delay increases. In contrast the new protocol achieves high utilization, small queues, and almost no drops, independent of capacity or delay. Even in conventional environments, the simulations show that our protocol exhibits better fairness, higher utilization, and smaller queue size, with almost no packet drops. Further, it maintains better performance in dynamic environments with many short web-like flows, and has no bias against long RTT flows.

Although we started with the goal of solving TCP's limitations in high-bandwidth large-delay environments, our design has several derivative advantages.

First, decoupling fairness control from utilization control opens new avenues for service differentiation using schemes that provide desired bandwidth apportioning, yet are too aggressive or too weak for controlling congestion. In § 6, we present a simple scheme that implements the shadow prices model [20].

Second, the protocol facilitates distinguishing error losses from congestion losses, which makes it useful for wireless environments. In XCP, drops caused by congestion are highly uncommon (e.g., less than one in a million packets in simulations). Further, since the protocol uses explicit and precise congestion feedback, a congestion drop is likely to be preceded by an explicit feedback that tells the source to decrease its congestion window. Losses that are preceded and followed by an explicit increase feedback are likely error losses.

Finally, XCP improves security by making it easier to detect attacks and isolate unresponsive flows as described in § 7.

XCP's performance provides an incentive for both end users and network providers to deploy the protocol. In § 8 we present possible deployment paths.

2 Design Rationale

Our initial objective is to step back and rethink Internet congestion control without caring about backward compatibility or deployment. If we were to build a new congestion control architecture from scratch, what might it look like?

The first observation is that loss is a poor signal of congestion. While we do not believe a cost-effective network can always avoid loss, dropping packets should be a congestion signal of last resort. As an implicit signal, loss is bad because congestion is not the only source of loss, and because a definite decision that a packet was lost cannot be made quickly. As a binary signal, loss only signals whether there is congestion (a loss) or not (no loss). Thus senders must probe the network to the point of congestion before backing off. Moreover, as the feedback is imprecise, the increase policy must be conservative and the decrease policy must be aggressive.

Tight congestion control requires explicit and precise congestion feedback. Congestion is not a binary variable, so congestion signalling should reflect the degree of congestion. We propose using precise congestion signalling, where the network explicitly tells the receiver the state of congestion and how to react to it. This allows the senders to decrease their sending windows quickly when the bottleneck is highly congested, while performing small reductions when the sending rate is close to the bottleneck capacity. The resulting protocol is both more responsive and less oscillatory.

Second, the aggressiveness of the sources should be adjusted according to the delay in the feedback-loop. The dynamics of congestion control may be abstracted as a control loop with feedback delay. A fundamental characteristic of such a system is that it becomes unstable for some large feedback delay. To counter this destabilizing effect, the system must slow down as the feedback delay increases. In the context of congestion control, this means that as delay increases, the sources should change their sending rates more slowly. This issue has been raised by other researchers [22, 27], but the important question is how exactly feedback should depend on delay to establish stability. Using tools from control theory, we conjecture that congestion feedback based on rate-mismatch should be inversely proportional to delay, and feedback based on queue-mismatch should be inversely proportional to the square of delay.

Robustness to congestion should be independent of unknown and quickly changing parameters, such as the number of flows. A fundamental principle from control theory states that a controller must react as quickly as the dynamics of the controlled signal; otherwise the controller will always lag behind the controlled system and will be ineffective. In the context of current proposals for congestion control, the controller is an Active Queue Management scheme (AQM). The controlled signal is the aggregate traffic traversing the link. The controller seeks to match input traffic to link capacity. However, this objective might be unachievable when the input traffic consists of TCP flows, because the dynamics of a TCP aggregate depend on the number of flows (N). The aggregate rate increases by N packets per RTT, or decreases proportionally to $1/N$. Since the number of flows in the aggregate is not constant and changes over time, no AQM controller with constant parameters can be fast enough to operate with an arbitrary number of TCP flows. Thus, a third objective of our system is to make the dynamics of the aggregate traffic independent from the number of flows.

This leads to the need for decoupling *efficiency control* (i.e., control of utilization or congestion) from *fairness control*. Robustness to congestion requires the behavior of aggregate traffic to be independent of the number of flows in it. However, any fair bandwidth allocation intrinsically depends on the number of flows traversing the bottleneck. Thus, the rule for dividing bandwidth among individual flows in an aggregate should be independent from the control law that governs the dynamics of the aggregate.

Traditionally, efficiency and fairness are coupled, as the same control law (such as AIMD in TCP) is used to obtain both fairness and efficiency simultaneously [3, 9, 17, 18, 16]. Conceptually, however, efficiency and fairness are independent. Efficiency involves only the aggregate traffic's behavior. When the input traffic rate equals the link capacity, no queue builds and 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 the same throughput irrespective of congestion.

In our new paradigm, a router has both an efficiency controller (EC) and a fairness controller (FC). This separation simplifies the design and analysis of each controller by reducing the requirements imposed. It also permits modifying one of the controllers without redesigning or re-analyzing the other. Furthermore, the decoupling opens new avenues for service differentiation using bandwidth allocation schemes that provide some controlled unfairness yet are too aggressive or too weak for controlling congestion.

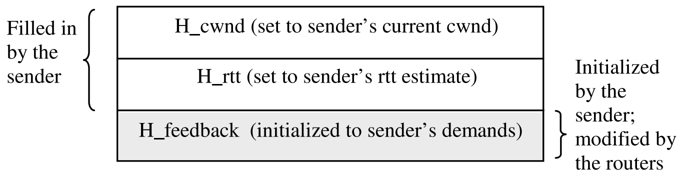
Filled in by the sender
H_cwnd (set to sender's current cwnd)
H_rtt (set to sender's rtt estimate)
H_feedback (initialized to sender's demands)
Initialized by the sender; modified by the routers

Figure 1: Congestion header.

3 Protocol

Like TCP, XCP is a window-based congestion control protocol intended for best effort traffic. However, its flexible architecture can easily support differentiated services as explained in § 6. The description of XCP in this section assumes a pure XCP network. In § 8, we show that XCP can coexist with TCP in the same Internet and be TCP-friendly.

3.1 Framework

First we give an overview of how control information flows in the network, then in § 3.5 we explain feedback computation.

Senders maintain their congestion window `cwnd` and round trip time `rtt`[1] and communicate these to the routers via a congestion header in every packet. Routers monitor the input traffic rate to each of their output queues. Based on the difference between the link bandwidth and its input traffic rate, the router tells the flows sharing that link to increase or decrease their congestion windows. It does this by annotating the congestion header of data packets. Feedback is divided between flows based on their `cwnd` and `rtt` values so that the system converges to fairness. A more congested router later in the path can further reduce the feedback in the congestion header by overwriting it. Ultimately, 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` accordingly.

3.2 The Congestion Header

Each XCP packet carries a congestion header (Figure 1), which is used to communicate a flow's state to routers and feedback from the routers on to the receivers. The field H_cwnd is the sender's current congestion window, whereas H_rtt is the sender's current RTT estimate. These are filled in by the sender and never modified in transit.

The remaining field, $H\_feedback$, takes positive or negative values and is initialized by the sender according to its requirements. Routers along the path modify this field to directly control the congestion windows of the sources.

[1]In this document, the notation RTT refers to the physical round trip time, `rtt` refers to the variable maintained by the source's software, and H_rtt refers to a field in the congestion header.

3.3 The XCP Sender

As with TCP, an XCP sender maintains a congestion window of the outstanding packets, `cwnd`, and an estimate of the round trip time `rtt`. On packet departure, the sender attaches a congestion header to the packet and sets the H_cwnd field to its current `cwnd` and H_rtt to its current `rtt`. In the first packet of a flow, H_rtt is set to zero to indicate to the routers that the source does not yet have a valid estimate of the RTT.

The sender uses the $H\_feedback$ field to request its desired window increase. For example, when the application has a desired rate r, the sender sets $H\_feedback$ to the desired increase in the congestion window ($r\cdot$ `rtt` - `cwnd`) divided by the number of packets in the current congestion window. If bandwidth is available, this initialization allows the sender to reach the desired rate after one RTT.

Whenever a new acknowledgment arrives, positive feedback increases the senders `cwnd` and negative feedback reduces it (s is the packet size):

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

In addition to direct feedback, XCP still needs to respond to losses although they are rare. It does this in a similar manner to TCP.

3.4 The XCP Receiver

An XCP receiver is similar to a TCP receiver except that when acknowledging a packet, it copies the congestion header from the data packet to its acknowledgment.

3.5 The XCP Router: The Control Laws

An XCP router uses a Drop-Tail or RED queue equipped with an *efficiency controller* and a *fairness controller*. Both of these compute estimates over the average RTT of the flows traversing the link, which smooths the burstiness of a window-based control protocol. Estimating parameters over intervals longer than the average RTT leads to sluggish response, while estimating parameters over shorter intervals leads to erroneous estimates. The average RTT is computed using the information in the congestion header.

XCP controllers make a single control decision every average RTT. This is motivated by the need to observe the results of previous control decisions before attempting a new control. For example, if the router tells the sources to increase their congestion windows, it should wait to see how much spare bandwidth remains before telling them to increase again.

The router maintains a per-link estimation-control timer that is set to the most recent estimate of the average RTT on that link. Upon timeout the router updates its estimates and its control decisions. In the remainder of this paper, we refer to the router's current estimate of the average RTT as d to emphasize this is the feedback delay.

3.5.1 The Efficiency Controller (EC)

The efficiency controller's purpose is to maximize link utilization while minimizing drop rate and persistent queues. It looks only at aggregate traffic and need not care about fairness issues, such as which flow a packet belongs to.

As XCP is window-based, the EC computes a desired increase or decrease in the number of bits that the aggregate traffic transmits in a control interval (i.e., an average RTT). This aggregate feedback ϕ is computed each control interval:

$$\phi = \alpha \cdot d \cdot S - \beta \cdot Q, \qquad (1)$$

α and β are constant parameters, whose values are set based on our stability analysis (§ 4) to 0.4 and 0.226, respectively. The term d is the average RTT, and S is the spare bandwidth defined as the difference between the input traffic rate and link capacity. Finally, Q is the persistent queue size, as opposed to a transient queue that results from the bursty nature of all window-based protocols. We compute Q by taking the minimum queue seen by an arriving packet during the last propagation delay, which we estimate by subtracting the local queuing delay from the average RTT.

Equation 1 makes the feedback proportional to the spare bandwidth because, when $S \geq 0$, the link is underutilized and we want to send positive feedback, while when $S < 0$, the link is congested and we want to send negative feedback. However this alone is insufficient because it would mean we give no feedback when the input traffic matches the capacity, and so the queue does not drain. To drain the persistent queue we make the aggregate feedback proportional to the persistent queue too. Finally, since the feedback is in bits, the spare bandwidth S is multiplied by the average RTT.

We can achieve efficiency by dividing the aggregate feedback into small chunks that we allocate to single packets as $H\_feedback$. Since the EC deals only with the aggregate behavior, it does not care which packets get the feedback and by how much each individual flow changes its congestion window. All the EC requires is that the total traffic changes by ϕ over this control interval. How exactly we divide the feedback among the packets (and hence the flows) affects only fairness, and so is the job of the fairness controller.

3.5.2 The Fairness Controller (FC)

The job of the fairness controller (FC) is to apportion the feedback to individual packets to achieve fairness. The FC relies on the same principle TCP uses to converge to fairness, namely *Additive-Increase Multiplicative-Decrease (AIMD)*. Thus, we want to compute the per-packet feedback according to the policy:

If $\phi > 0$, allocate it so that the increase in throughput of all flows is the same.

If $\phi < 0$, allocate it so that the decrease in throughput of a flow is proportional to its current throughput.

This ensures continuous convergence to fairness as long as the aggregate feedback ϕ is not zero. To prevent convergence stalling when efficiency becomes optimal ($\phi = 0$), we introduce the concept of bandwidth shuffling. This is the simultaneous allocation and deallocation of bandwidth such that the total traffic rate (and consequently the efficiency) does not change, yet the throughput of each individual flow changes gradually to approach the flow's fair share. The shuffled traffic is computed as follows:

$$h = \max(0, \gamma \cdot y - |\phi|), \tag{2}$$

where y is the input traffic in an average RTT and γ is a constant set to 0.1. This equation ensures that, every average RTT, at least 10% of the traffic is redistributed according to AIMD. The choice of 10% is a tradeoff between the time to converge to fairness and the disturbance the shuffling imposes on a system that is around optimal efficiency.

Next, we compute the per-packet feedback that allows the FC to enforce the above policies. Since the increase law is additive whereas the decrease is multiplicative, it is convenient to compute the feedback assigned to packet i as the combination of a positive feedback p_i and a negative feedback n_i.

$$H\_feedback_i = p_i - n_i. \tag{3}$$

First, we compute the case when the aggregate feedback is positive ($\phi > 0$). In this case, we want to increase the throughput of all flows by the same amount. Thus, we want the change in the throughput of any flow i to be proportional to the same constant, (i.e., $\Delta throughput_i \propto constant$). Since we are dealing with a window-based protocol, we want to compute the change in congestion window rather than the change in throughput. The change in the congestion window of flow i is the change in its throughput multiplied by its RTT. Hence, the change in the congestion window of flow i should be proportional to the flow's RTT, (i.e., $\Delta cwnd_i \propto rtt_i$).

The next step is to translate this desired change of congestion window to per-packet feedback that will be reported in the congestion header. The total change in congestion window of a flow is the sum of the per-packet feedback it receives. Thus, we obtain the per-packet feedback by dividing the change in congestion window by the expected number of packets from flow i that the router sees in a control interval d. This number is proportional to the flow's congestion window, $cwnd_i$, and inversely proportional to its round trip time, rtt_i. Thus, we find that the per-packet positive feedback is proportional to the square of the flow's RTT, and inversely proportional to its congestion window, (i.e., $p_i \propto \frac{rtt_i^2}{cwnd_i}$).

Thus, positive feedback p_i is given by:

$$p_i = \xi_p \frac{rtt_i^2}{cwnd_i}, \tag{4}$$

where ξ_p is a constant. The total increase in the aggregate traffic rate is $\frac{h+\max(\phi,0)}{d}$, where $max(\phi, 0)$ ensures that we are computing the positive feedback. This is equal to the sum of the increase in the rates of all flows in the aggregate, which is the sum of the positive feedback a flow has received divided by its rtt, and so:

$$\frac{h + \max(\phi, 0)}{d} = \sum^{L} \frac{p_i}{rtt_i}, \tag{5}$$

where L is the number of packets seen by the router in an average RTT. From this, ξ_p can be derived as:

$$\xi_p = \frac{h + \max(\phi, 0)}{d \cdot \sum \frac{rtt_i}{cwnd_i}}. \tag{6}$$

Similarly, we compute the per-packet negative feedback given when the aggregate feedback is negative ($\phi < 0$). In this case, we want the decrease in the throughput of flow i to be proportional to its current throughput (i.e., Δ*throughput*$_i \propto$ *throughput*$_i$). Consequently, the desired change in the flow's congestion window is proportional to its current congestion window (i.e., Δ*cwnd*$_i \propto$ *cwnd*$_i$). Again, the desired per-packet feedback is the desired change in the congestion window divided by the expected number of packets from this flow that the router sees in an interval d. Thus, we finally find that the per-packet negative feedback should be proportional to a flow's RTT (i.e., $n_i \propto rtt_i$).

Thus negative feedback n_i is given by:

$$n_i = \xi_n \cdot rtt_i \tag{7}$$

where ξ_n is a constant. As with the increase case, the total decrease in the aggregate traffic rate is the sum of the decrease in the rates of all flows in the aggregate:

$$\frac{h + \max(-\phi, 0)}{d} = \sum^{L} \frac{n_i}{rtt_i}. \tag{8}$$

As so, ξ_n can be derived as:

$$\xi_n = \frac{h + \max(-\phi, 0)}{d \cdot L}. \tag{9}$$

3.5.3 Notes on the Efficiency and Fairness Controllers

A few points are worth noting about the design of the efficiency controller and the fairness controller.

As mentioned earlier, the efficiency and fairness controllers are decoupled. Specifically, the efficiency controller uses a Multiplicative-Increase Multiplicative-Decrease law (MIMD), whereas the fairness controller uses an Additive-Increase Multiplicative-Decrease law (AIMD). The independence is maintained by ensuring that as individual flows follow the feedback from the fairness controller, the aggregate traffic obeys the efficiency controller. XCP guarantees this independence because its equations and implementation ensure that the sum of feedback given to individual flows in a control interval adds up to the aggregate feedback computed by the efficiency controller.

The particular control laws used by the efficiency controller (MIMD) and the fairness controller (AIMD) are not the only possible choices. For example, in [26] we describe a fairness controller that uses a binomial law similar to those described in [6]. We chose the control laws above because our analysis and simulation demonstrate their good performance.

We note that the efficiency controller satisfies the requirements in § 2. The dynamics of the aggregate traffic are specified by the aggregate feedback and stay independent of the number of flows traversing the link. Additionally, in contrast to TCP where the increase/decrease rules are indifferent to the degree of congestion in the network, the aggregate feedback sent by the EC is proportional to the degree of under- or over-utilization. Furthermore, since the aggregate feedback is given over an average RTT, XCP becomes less aggressive as the round trip delay increases.[2]

Although the fairness controller uses AIMD, it is significantly fairer than TCP. Note that in AIMD, all flows increase equally regardless of their current rate. Therefore, it is the multiplicative-decrease that helps converging to fairness. In TCP, multiplicative-decrease is tied to the occurrence of a drop, which should be a rare event. In contrast, with XCP multiplicative-decrease is decoupled from drops and is performed every average RTT.

XCP is fairly robust to estimation errors. For example, we estimate the value of ξ_p every d and use it as a prediction of the value of ξ_p during the following control interval (i.e., the following d). If we underestimate ξ_p, we will fail to allocate all of the positive feedback in the current control interval. Nonetheless, the bandwidth we fail to allocate will appear in our next estimation of the input traffic as a spare bandwidth, which will be allocated (or partially allocated) in the following control interval. Thus, in every control interval, a portion of the spare bandwidth is allocated until none is left. Since our underestimation of ξ_p causes reduced allocation, the convergence to efficiency is slower than if our prediction of ξ_p had been correct. Yet the error does not stop XCP from reaching full utilization. Similarly, if we overestimate ξ_p then we will allocate more feedback to flows at the beginning of a control interval and run out of aggregate feedback quickly. This uneven spread of feedback over the allocation interval does not affect convergence to utilization but it slows down convergence to fairness. A similar argument can be made about other estimation errors; they mainly affect the convergence time rather than the correctness of the controllers.[3]

XCP congestion control is independent of the number of sources, the capacity of the bottleneck, and the delay, and so we can choose constant values for the parameters α and β that work in all environments. This is a significant improvement over previous approaches where the parameters either work only in specific environments (e.g, RED) or have to be chosen differently depending on the number of sources, the capacity, and the delay (e.g., AVQ). In § 4, we show how these constant values are chosen.

Finally, implementing the efficiency and fairness controllers is fairly simple and requires only a few lines of code as shown in Appendix A. We note that an XCP router performs only a few additions and 3 multiplications per packet, making it an attractive choice even as a backbone router.

[2]The relation between XCP's dynamics and feedback delay is hard to fully grasp from Equation 1. We refer the interested reader to Equation 16, which shows that the change in throughput based on rate-mismatch is inversely proportional to delay, and the change based on queue-mismatch is inversely proportional to the square of delay.

[3]There is one type of error that may prevent the convergence to complete efficiency, which is the unbalanced allocation and deallocation of the shuffled traffic. For example, if by the end of a control interval we deallocate all of the shuffled traffic but fail to allocate it, then the shuffling might prevent us from reaching full link utilization. Yet note that the shuffled traffic is only 10% of the input traffic. Furthermore, shuffling exists only when $|\phi| < 0.1y$.

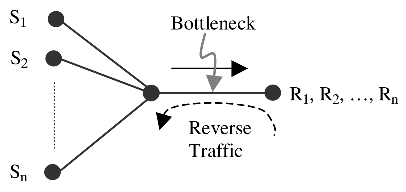

Figure 2: A single bottleneck topology.

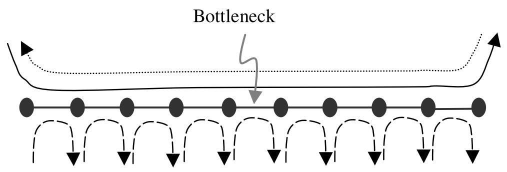

Figure 3: A parking lot topology.

4 Stability Analysis

We use a fluid model of the traffic to analyze the stability of XCP. Our analysis considers a single link traversed by multiple XCP flows. For the sake of simplicity and tractability, similarly to previous work [21, 15, 22, 24], our analysis assumes all flows have a common, finite, and positive round trip delay, and neglects boundary conditions (i.e., queues are bounded, rates cannot be negative). Later, we demonstrate through extensive simulations that even with larger topologies, different RTTs, and boundary conditions, our results still hold.

The main result can be stated as follows.

Theorem 1. *Suppose the round trip delay is d. If the parameters α and β satisfy:*

$$0 < \alpha < \frac{\pi}{4\sqrt{2}} \quad \textit{and} \quad \beta = \alpha^2\sqrt{2},$$

then the system is stable independently of delay, capacity, and number of sources.

The details of the proof are given in Appendix B. The idea underlying the stability proof is the following. Given the assumptions above, our system is a linear feedback system with delay. The stability of such systems may be studied by plotting their open-loop transfer function in a Nyquist plot. We prove that by choosing α and β as stated above, the system satisfies the Nyquist stability criterion. Further, the gain margin is greater than one and the phase margin is positive independently of delay, capacity, and number of sources. [4]

5 Performance

In this section, we demonstrate through extensive simulations that by complying with the conditions in Theorem 1, we can choose constant values for α and β that work with any capacity and delay, as well as any number of sources. Our simulations cover capacities in [1.5 Mb/s, 4 Gb/s], propagation delays in [10 ms, 3 sec], and number of sources in [1, 1000]. Further, we simulate 2-way traffic (with the resulting ack compression) and dynamic environments with arrivals and departures of short web-like flows. In all of these simulations, we set $\alpha = 0.4$ and $\beta = 0.226$ showing the robustness of our results.

Additionally, the simulations show that in contrast to TCP, the new protocol dampens oscillations and smoothly converges to high utilization, small queue size, and fair bandwidth allocation. We also demonstrate that the protocol is robust to highly varying traffic demands and high variance in flows' round trip times.

Our simulations compare XCP with various AQM schemes running under similar conditions. They show that XCP outperforms previous proposals in almost every aspect. [5]

5.1 Simulation Setup

Our simulations use the packet-level simulator *ns-2* [1], which we have extended with an XCP module. We compare XCP with TCP Reno over the following queuing disciplines:

Random Early Discard (RED [13]). Our experiments use the "gentle" mode and set the parameters according to the authors' recommendations in [2]. The minimum and the maximum thresholds are set to one third and two thirds the buffer size, respectively.

Random Early Marking (REM [5]). Our experiments set REM parameters according to the authors' recommendation provided with their code. In particular, $\phi = 1.001$, $\gamma = 0.001$, the update interval is set to the transmission time of 10 packets, and *qref* is set to one third of the buffer size.

Adaptive Virtual Queue (AVQ [21]). As recommended by the authors, our experiments use $\gamma = 0.98$ and compute α based on the equation in [21]. Yet, as shown in [19], the equation for setting α does not admit a solution for high capacities. In these cases, we use $\alpha = 0.15$ as used in [21].

[4]The gain margin is the magnitude of the transfer function at the frequency $-\pi$. The phase margin is the frequency at which the magnitude of the transfer function becomes 1. They are used to prove robust stability.

[5]We will make our code and simulation scripts publicly available.

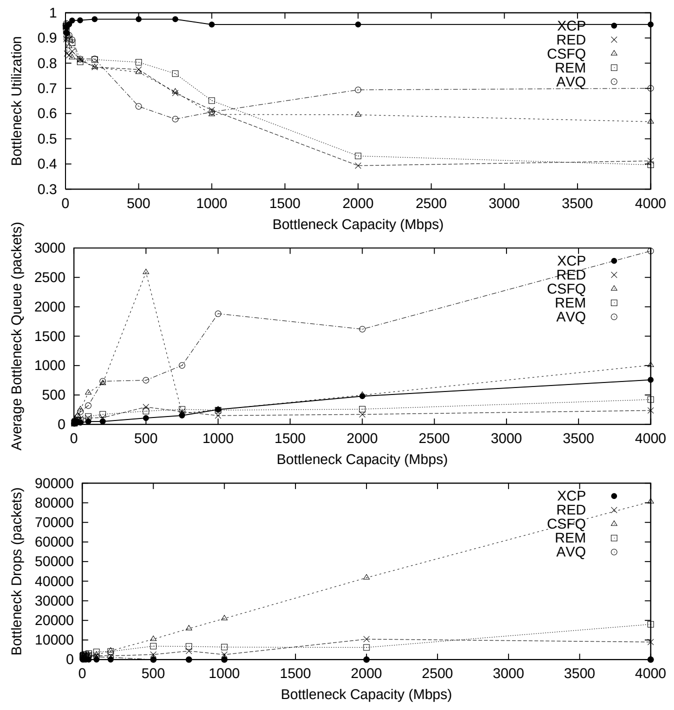

Figure 4: XCP significantly outperforms TCP in high bandwidth environments. The graphs compare the efficiency of XCP with that of TCP over RED, CSFQ, REM, and AVQ as a function of capacity.

Core Stateless Fair Queuing (CSFQ [29]). In contrast to the above AQMs, whose goal is to achieve high utilization and small queue size, CSFQ aims for providing high fairness in a network cloud with no per-flow state in core routers. We compare CSFQ with XCP to show that XCP can be used within the CSFQ framework to improve its fairness and efficiency. Again, the parameters are set to the values chosen by the authors in their *ns* implementation.

The simulator code for these AQM schemes is provided by their authors. Further, to allow these schemes to exhibit their best performance, we simulate them with ECN enabled.

In all of our simulations, the XCP parameters are set to $\alpha = 0.4$ and $\beta = 0.226$. We experimented with XCP with both Drop-Tail and RED dropping policies. There was no difference between the two cases because XCP almost never dropped packets.

Most of our simulations use the topology in Figure 2. The bottleneck capacity, the round trip delay, and the number of flows vary according to the objective of the experiment. The buffer size is always set to the delay-bandwidth product. The data packet size is 1000 bytes. Simulations over the topology in Figure 3 are used to show that our results generalize to larger and more complex topologies. Unless specified differently, the reader should assume that the simulation topology is that in Figure 2, the flows RTTs are equivalent, and the sources are long-lived FTP flows. Simulations' running times vary depending on the propagation delay but are always

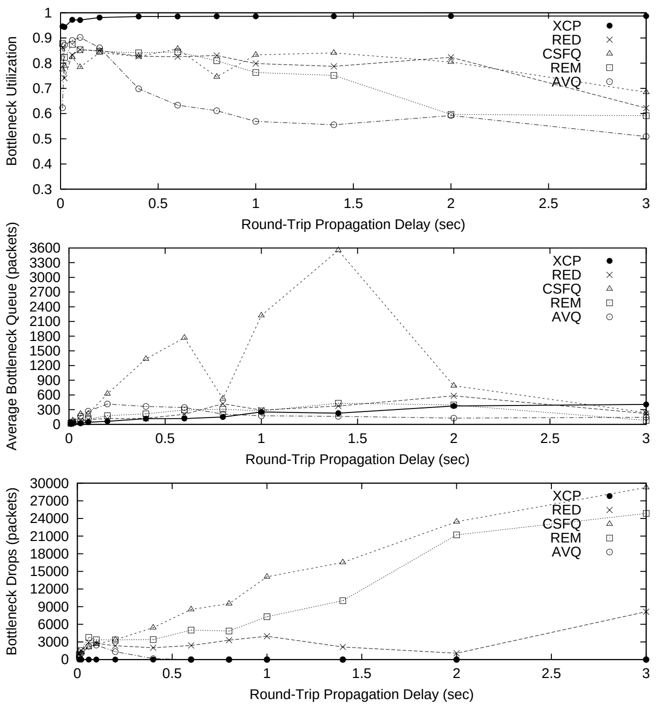

Figure 5: XCP significantly outperforms TCP in high delay environments. The graphs compare bottleneck utilization, average queue, and number of drops as round trip delay increases when flows are XCPs and when they are TCPs over RED, CSFQ, REM, and AVQ.

larger than 300 RTTs. All simulations were run long enough to ensure the system has reached a consistent behavior.

5.2 Comparison with TCP and AQM Schemes

Impact of Capacity: We show that an increase in link capacity will cause a significant degradation in TCP's performance, irrespective of the queuing scheme. In this experiment, 50 long-lived FTP flows share a bottleneck. All links have the same propagation delay of 20 ms. Additionally, there are 50 flows traversing the reverse path and used merely to create a 2-way traffic environment with the potential for ack compression. Since XCP is based on a fluid model and estimates some parameters, the existence of reverse traffic tends to stress the protocol.

Figure 4 demonstrates that as capacity increases, TCP's bottleneck utilization decreases significantly. This happens regardless of the queuing scheme. In contrast, XCP's utilization is always near optimal independent of the link capacity. Furthermore, XCP shows a considerably smaller queue size and drop rate than TCP. We particularly note that in all of the experiments in Figure 4, XCP did not drop a single packet.

Impact of Feedback Delay: We fix the bottleneck capacity at 45 Mb/s and study the impact of increased delay on the performance of congestion control. All other parameters have the same values used in the previous experiment.

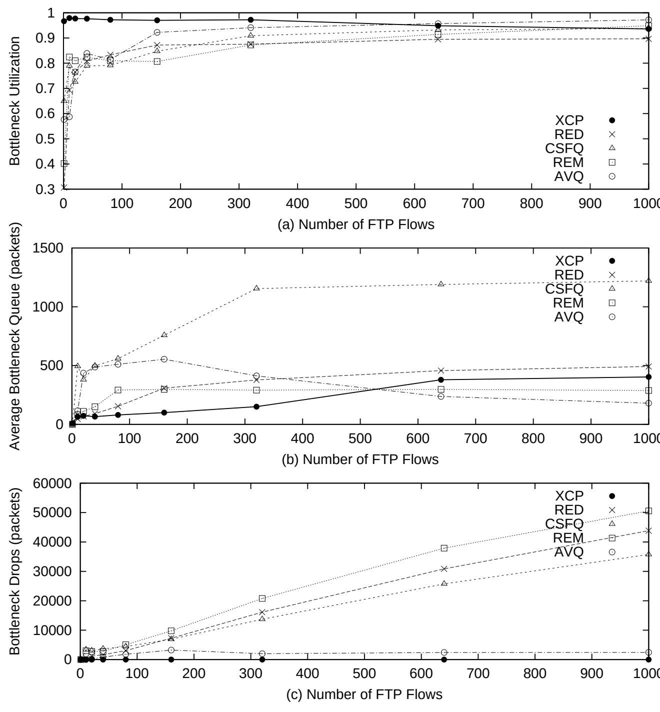

Figure 6: XCP is efficient with any number of flows. The graphs compare the efficiency of XCP and TCP with various queuing schemes as a function of the number of flows.

Figure 5 shows that as the propagation delay increases, TCP's utilization degrades considerably regardless of the queuing scheme. XCP, on the other hand, maintains high utilization independently of delay.

The adverse impact of large delay on TCP's performance has been noted over satellite links. The bursty nature of TCP has been suggested as a potential explanation and packet pacing has been proposed as a solution [4]; however, this experiment shows that burstiness is a minor factor. In particular, XCP is a bursty window-based protocol but it copes with delay much better than TCP. It does so by adjusting its aggressiveness according to round trip delay.

Impact of Number of Flows: We fix the bottleneck capacity at 100 Mb/s and round trip propagation delay at 100 ms and repeat the same experiment with a varying number of FTP sources. Other parameters have the same values used in the previous experiment.[6] Figure 6 shows that overall, XCP exhibits better utilization, reasonable queue size, and no packet losses. The increase in XCP queue as the number of flows increases is a side effect of its high fairness (see Figure 8). When the number of flows is larger than 500, the fair congestion window is between one and two packets. Since the fair congestion window is a real number but the effective congestion window is an integer number of packets, the rounding error increases causing a disturbance. Consequently, the queue

[6]We choose a capacity of 100 Mb/s and a round trip delay of 100 ms to guarantee that the pipe is large enough for all flows to send at least one packet and stay active.

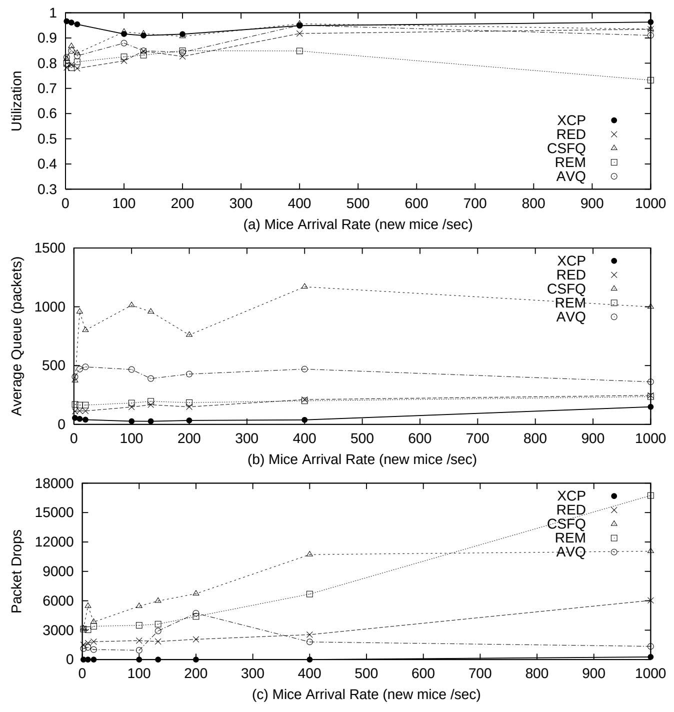

Figure 7: XCP is robust and efficient in environments with arrivals and departures of short web-like flows. The graphs compare the efficiency of XCP to that of TCP over various queuing schemes as a function of short flows arrival rate.

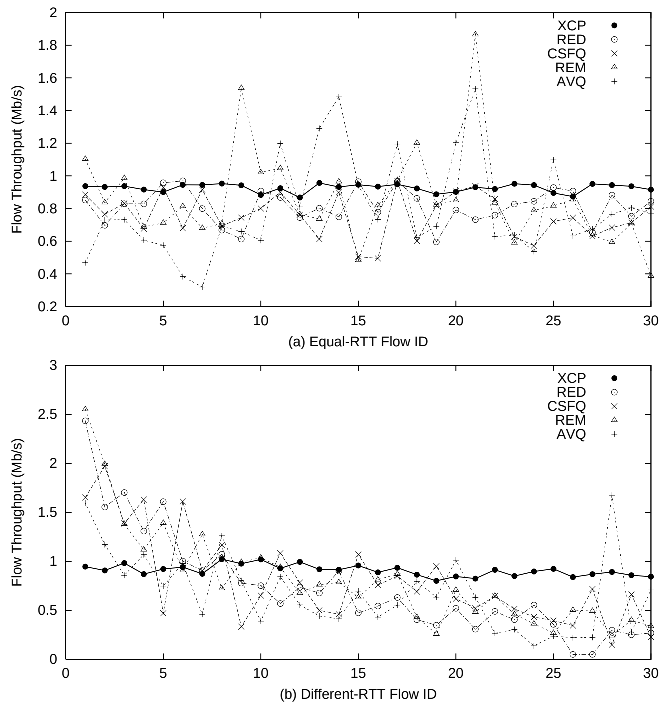

Figure 8: XCP is fair to both equal and different RTT flows. The graphs compare XCP's Fairness to that of TCP over RED, CSFQ, REM, and AVQ.

increases to absorb this disturbance.

Impact of Short Web-Like Traffic: Since a large number of flows in the Internet are short web-like flows, it is important to investigate the impact of such dynamic flows on congestion control. In this experiment, we have 50 long-lived FTP flows traversing the bottleneck link. Also, there are 50 flows traversing the reverse path whose presence emulates a 2-way traffic environment with the resulting ack compression. The bottleneck bandwidth is 100 Mb/s and the round trip delay is 80 ms. Short flows arrive according to a Poisson process. Their transfer size is derived from a Pareto distribution with an average of 30 packets (ns-implementation with shape_ $= 1.35$), which complies with real web traffic [11].

Figure 7 graphs bottleneck utilization, average queue size, and total number of drops, all as functions of the arrival rate of short flows. The figure shows that XCP outperforms the other schemes in all experiments. Further, the utilization, queue size, and drop rate obtained by XCP are very close to the optimal behavior. Most importantly, this particular experiment demonstrates XCP's robustness in dynamic environments with large numbers of flow arrivals and departures.

Fairness : This experiment shows that XCP is significantly fairer than TCP, regardless of the queuing scheme. We have 30 long-lived FTP flows sharing a single 30 Mb/s bottleneck. We conduct two sets of simulations. In the first set, all flows have a common round-trip propagation delay of 40 ms. In the second set of simulations, the flows have different RTTs in the range [40 ms, 330 ms] ($RTT_{i+1} = RTT_i + 10ms$).

Figures 8-a and 8-b demonstrate that, in contrast to other approaches, XCP provides a fair bandwidth allocation and does not have any bias against long RTT flows. Furthermore, Figure 8-b demonstrates XCP robustness to high variance in the RTT distribution. Thus, although XCP computes an estimate of the average RTT of the system, it still operates correctly in environments where the RTT varies widely from one flow to another. For further information on this point see Appendix C.

At the end of this section, it is worth noting that the average drop rate of XCP was less than 10^{-6}, which is three orders of magnitude smaller than the other schemes despite their use of ECN.

5.3 The Dynamics of XCP

While the simulations presented above focus on long term average behavior, this section shows the short term dynamics of XCP. In particular, we show that XCP's utilization, queue size, and throughput exhibit very limited oscillations. Therefore, the average behavior presented in the section above is highly representative of the general behavior of the protocol.

Convergence Dynamics: We show that XCP dampens oscillations and converges smoothly to high utilization small queues and fair bandwidth allocation. In this experiment, 5 long-lived flows share a 45 Mb/s bottleneck and have a com-

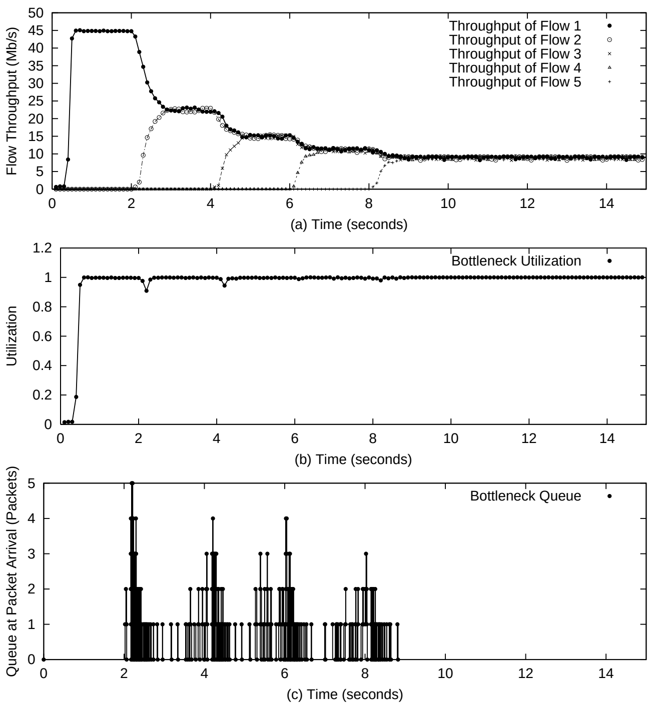

Figure 9: XCP's smooth convergence to high fairness, good utilization, and small queue size. Five XCP flows share a 45 Mb/s bottleneck. They start their transfers at times 0, 2, 4, 6, and 8 seconds.

mon RTT of 40 ms. The flows start their transfers two seconds apart at 0, 2, 4, 6, and 8 seconds.

Figure 9-a shows that whenever a new flow starts, the fairness controller reallocates bandwidth to maintain min-max fairness. Figure 9-b shows that decoupling utilization and fairness control ensures that this reallocation is achieved without disturbing the link's high utilization. Finally, Figure 9-c shows the instantaneous queue, which effectively absorbs the new traffic and drains afterwards.

Robustness to Sudden Increase or Decrease in Traffic Demands: In this experiment, we examine performance as traffic demands and dynamics vary considerably. We start the simulation with 10 long-lived FTP flows sharing a 100 Mb/s bottleneck with a round trip propagation delay of 40 ms. At $t = 4$ seconds, we start 100 new flows and let them stabilize. At $t = 8$ seconds, we stop these 100 flows leaving the original 10 flows in the system.

Figure 10 shows the utilization and queue, both for the case when the flows are XCP, and for when they are TCPs traversing RED queues. XCP absorbs the new burst of flows without dropping any packets, while maintaining high utilization. TCP on the other hand is highly disturbed by the sudden increase in the traffic and takes a long time to restabilize. When the flows are suddenly stopped at $t = 10$ seconds, XCP quickly reallocates the spare bandwidth and continues to have high utilization. In contrast, the sudden decrease in demand destabilizes TCP and causes a large sequence of oscillations.

A More Complex Topology: This experiment uses the 9-link topology in Figure 3, although results are very similar for topologies with more links. Link 5 has the lowest capacity, namely 50 Mb/s, whereas the others are 100 Mb/s links. All links have 20 ms one-way propagation delay. Fifty flows, represented by the solid arrow, traverse all links in the forward direction. Fifty cross flows, illustrated by the small dashed arrows, traverse each individual link in the forward direction. 50 flows also traverse all links along the reverse path.

Figure 11 illustrates the average utilization, queue size, and number of drops at every link. In general, all schemes maintain a reasonably high utilization at all links (note the y-scale). However, the trade off between optimal utilization and small queue size is handled differently in XCP from the various AQM schemes. XCP trades a few percent of utilization for a considerably smaller queue size. XCP's lower utilization in this experiment compared to previous ones is due to disturbance introduced by shuffling. In particular, at links 1, 2, 3, and 4 the fairness controller tries to shuffle bandwidth from the cross flows to the long-distance flows, which have lower throughput. Yet, these long-distance flows are throttled downstream at link 5, and so cannot benefit from this positive feedback. This effect is mitigated at links downstream from link 5 because they can observe the upstream throttling and correspondingly reduce the amount of negative feedback given (see implementation in Appendix A). In any event, as the total shuffled bandwidth is less than 10%, the utilization is always higher than 90%.

It is possible to modify XCP so that it maintains the queue around a target value rather than draining all of it. This would cause the disturbance induced by shuffling to appear as a fluctuation in the queue rather than as a drop in utilization. However, we believe that maintaining a small queue size is more valuable than a few percent increase in utilization when flows traverse multiple congested links. In particular, it leaves a safety margin for bursty arrivals of new flows. In contrast, the large queues maintained at all links in the TCP simulations cause every packet to wait at all of the nine queues, which considerably increases end-to-end latency.

6 Quality of Service

XCP renders differentiated services easier to design and implement because it decouples fairness control from efficiency control. In particular, since high utilization and small queues are guaranteed by the efficiency controller, designing differential services becomes a matter of allocating the aggregate feedback to the individual flows so that they converge to the desired rates. The designer need not worry whether the flows increase too aggressively, causing congestion, or whether they are too slow at grabbing the spare bandwidth.

Before describing our service differentiation scheme, we note that in XCP, bandwidth differentiation is the only meaningful quality of service (QoS). Since XCP provides small

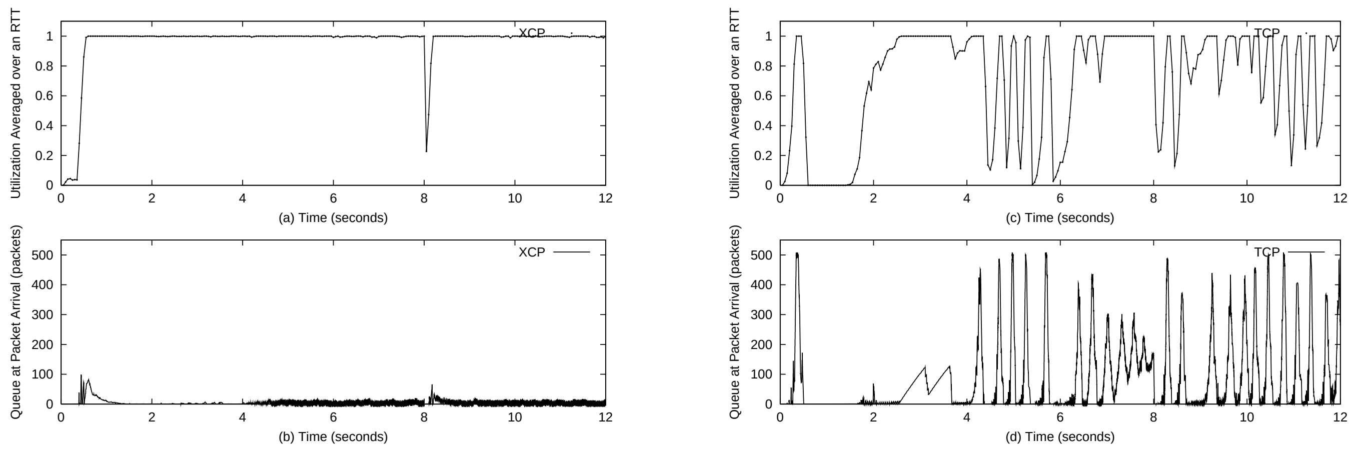

Figure 10: XCP is more robust against sudden increase or decrease in traffic demands than TCP. Ten FTP flows share a bottleneck. At time $t = 4$ seconds, we start 100 additional flows. At $t = 8$ seconds, these 100 flows are suddenly stopped and the original 10 flows are left to stabilize again.

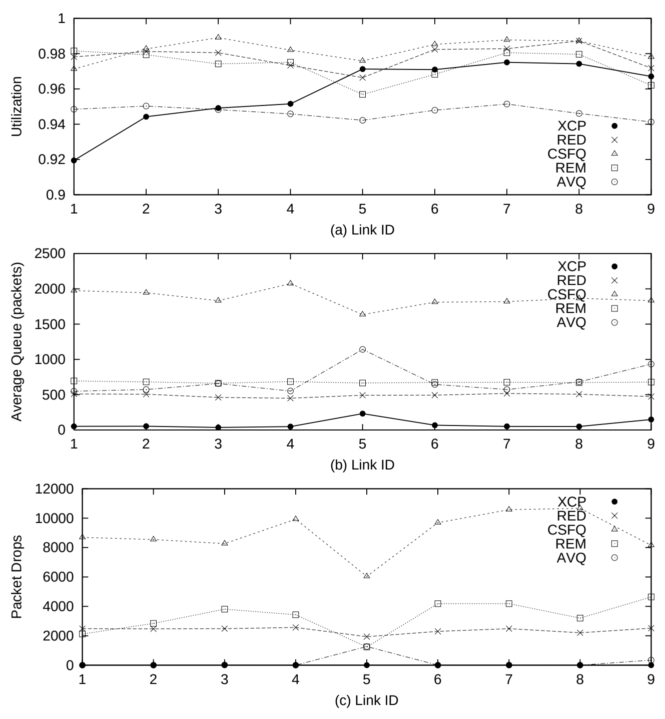

Figure 11: Simulation with multiple congested queues. Utilization, average Queue size, and number of drops at nine consecutive links (topology in Figure 3). Link 5 has the lowest capacity along the path.

queue size and near-zero drops, QoS schemes that guarantee small queuing delay or low jitter are redundant.

In this section, we describe a simple scheme that can provide differential services according to the shadow prices model defined by Kelly [20]. In this model, a user chooses the price per unit time she is willing to pay. The network allocates bandwidth so that the throughputs of users competing for the same bottleneck are proportional to their prices; (i.e.,

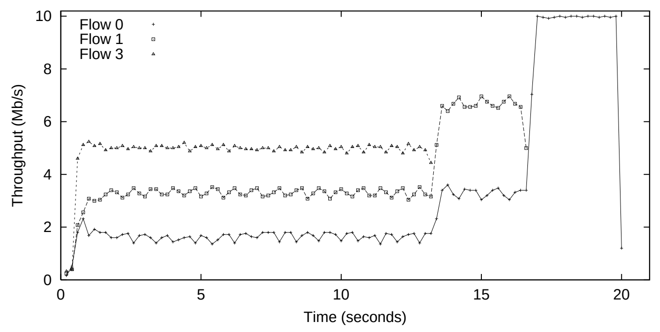

Figure 12: XCP can provide differential services by making users' throughputs proportional to the price they pay. Three XCP flows each transferring a 10 Mbytes file over a shared 10 Mb/s bottleneck. Flow 1' s price is 5, Flow 2's price is 10, and Flow 3's price is 15. Throughput is averaged over 200 ms (5 RTTs).

$\frac{throughput_i}{price_i} = \frac{throughput_j}{price_j}$).

The model allows a continuous spectrum of service differentiation because by changing the price the user can achieve the rate she desires. Note that all users sharing the same bottleneck pay the same price per unit of bandwidth. Hence, by observing the rate the user gets for a particular price, she can predict the increase in price she needs in order to obtain a certain desired rate.

Using the above model, one can easily provide end-to-end quality of service in an XCP network. To do so, the sender replaces the H_cwnd field by its current congestion window divided by the price she is willing to pay (i.e, cwnd/price). This minor modification is enough to produce a service that complies with the above model, and allows any throughput differentiation that a user desires.

Next, we show simulation results that support our claims. Three XCP sources share a 10 Mb/s bottleneck. The corresponding prices are $p_1 = 5$, $p_2 = 10$, and $p_3 = 15$. Each source wants to transfer a file of 10 Mbytes, and they all start together at $t = 0$. The results in Figure 12 show that the transfer rate depends on the price the flow pays. At the beginning, when all flows are active, their throughputs are 5 Mb/s, $3\frac{1}{3}$Mb/s, and $1\frac{2}{3}$Mb/s, which are proportional to their corresponding prices. After Flow 1 finishes its transfer, the remaining flows grab the freed bandwidth such that their throughputs continue being proportional to their prices. Note the high responsiveness of the system. In particular, when Flow 1 finishes its transfer freeing half of the link capacity, the other flows' sending rates adapt in a few RTTs.

7 Security

In this section, we show that XCP improves network security against attacks that target the network infrastructure by attempting to congest one or more links and deny bandwidth to other legitimate flows. Attacks that target a server or a particular software bug at a router module, though highly important, are outside the scope of this paper.

XCP facilitates detecting network attacks and isolating unresponsive sources because of two features: its operation with near-zero drops; and the explicit feedback.

The simulation results demonstrate that it is unlikely for XCP to drop packets. Hence, a sustained high drop rate is a strong sign of unresponsive flows. Thus, in XCP it is easy to detect flows that do not respond to congestion signals and attempt to deny other flows their legitimate share of the bandwidth. This is in contrast to a TCP network where drops are normal events and the drop rate depends on the number of flows rather than on the flows being unresponsive. Hence, while detecting attacks in a TCP network requires continuous scrutiny of at least a subset of the sources [23], and imposes an extra load on an already congested router, in an XCP network such detection happens naturally.

Additionally, isolating the misbehaving source becomes faster and easier because the the router can use the explicit feedback to test a source. More precisely, in TCP isolating an unresponsive source requires the router to monitor the average rate of a suspect source over a fairly long interval to decide whether the source is reacting according to AIMD. Also, since the source's RTT is unknown, its correct sending rate is not specified, which complicates the task even further. In contrast, in XCP, isolating a suspect flow is easy. The router can send the flow a test feedback requiring it to decrease its congestion window to a particular value. If the flow does not react in a single RTT then it is unresponsive. The fact that the flow specifies its RTT in the packet makes the monitoring easier. Since the flow cannot tell when a router is monitoring its behavior, it has to always follow the explicit feedback.[7]

[7] As an example, consider a flow that tries to increase its sending rate by claiming its RTT is larger than it really is. As a result the input traffic will exceed the link capacity and there will be drops. The router notices these drops and raises one bit in all of the packets it forwards during this high drop period. The exit border routers notice this bit and monitor the suspect flows. In particular, they send these flows a negative feedback and monitor the resulting traffic rate. The misbehaving source will show a higher rate than it is claiming and will be discovered in about one RTT.

8 Gradual Deployment

XCP is amenable to gradual deployment, which could follow one of two paths.

8.1 XCP-based Core Stateless Fair Queuing

XCP can be deployed in a cloud-based approach similar to that proposed by Core Stateless Fair Queuing (CSFQ). Such an approach would have several benefits. It would force unresponsive or UDP flows to use a fair share without needing per-flow state in the network core. It would improve the efficiency of the network because an XCP core allows higher utilization, smaller queue sizes, and minimal packet drops. It also would allow an ISP to provide differential bandwidth allocation internally in their network. CSFQ obviously shares these objectives, but our simulations indicate that XCP might give better fairness, higher utilization, and lower delay.

To use XCP in this way, we map TCP or UDP flows across a network cloud onto XCP flows between the ingress and egress border routes. Each XCP flow is associated with a queue at the ingress router. Arriving TCP or UDP packets enter the relevant queue, and the corresponding XCP flow across the core determines when they can leave. For this purpose, H_rtt is the measured propagation delay between ingress and egress routers, and H_cwnd is set to the XCP congestion window maintained by the ingress router (not the TCP congestion window).

Maintaining an XCP core can be simplified further. First, there is no need to attach a congestion header to the packets, as feedback can be collected using a small control packet exchanged between border routers every RTT. Second, multiple micro flows that share the same pair of ingress and egress border routers can be mapped to a single XCP flow. The weighted fairness scheme, described in § 6, allows each XCP macro-flow to obtain a throughput proportional to the number of micro-flows in it. The router will forward packets from the queue according to the XCP macro-flow rate. TCP will naturally cause the micro-flows to converge to share the XCP macro-flow fairly, although care should be taken not to mix responsive and unresponsive flows in the same macro-flow.

8.2 A TCP-friendly XCP

In this section, we describe a mechanism allowing end-to-end XCP to compete fairly with TCP in the same network. This design can be used to allow XCP to exist in a multi-protocol network, or as a mechanism for incremental deployment.

To start an XCP connection, the sender must check whether the receiver and the routers along the path are XCP-enabled. If they are not, the sender reverts to TCP or another conventional protocol. These checks can be done using simple TCP and IP options.

We then extend the design of an XCP router to handle a mixture of XCP and TCP flows while ensuring that XCP flows are TCP-friendly. The router distinguishes XCP traffic from non-XCP traffic and queues it separately. TCP packets are queued in a conventional RED queue (the *T-queue*). XCP flows are queued in an XCP-enabled queue (the *X-queue*, described in § 3.5). To be fair, the router should process packets from the two queues such that the average throughput observed by XCP flows equals the average throughput observed by TCP flows, irrespective of the number of flows. This is done using weighted-fair queuing with two queues where the weights are dynamically updated and converge to the fair shares of XCP and TCP. The weight update mechanism uses the T-queue drop rate p to compute the average congestion window of the TCP flows. The computation uses a TFRC-like [12] approach, based on TCP's throughput equation:

$$\overline{cwnd}_{TCP} = \frac{s}{\sqrt{\frac{2p}{3}} + 12p\sqrt{\frac{3p}{8}} \times (1 + 32p^2)}, \qquad (10)$$

where s is the average packet size. When the estimation-control timer fires, the weights are updated as follows:

$$w_T = w_T + \kappa \frac{\overline{cwnd}_{XCP} - \overline{cwnd}_{TCP}}{\overline{cwnd}_{XCP} + \overline{cwnd}_{TCP}}, \qquad (11)$$

$$w_X = w_X + \kappa \frac{\overline{cwnd}_{TCP} - \overline{cwnd}_{XCP}}{\overline{cwnd}_{XCP} + \overline{cwnd}_{TCP}}, \qquad (12)$$

where κ is a small constant in the range (0,1), and w_T and w_X are the T-queue and the X-queue weights. This updates the weights to decrease the difference between TCP's and XCP's average congestion windows. When the difference becomes zero, the weights stop changing and stabilize.

Finally, the aggregate feedback is modified to cause the XCP traffic to converge to its fair share of the link bandwidth:

$$\phi = \alpha \cdot d \cdot S_X - \beta \cdot Q_X, \qquad (13)$$

where α and β are constant parameters, d the average round trip time, Q_X is the size of the X-queue, and S_X is XCP's fair share of the spare bandwidth computed as:

$$S_X = w_X \cdot c - y_X, \qquad (14)$$

where w_X is the XCP weight, c is the capacity of the link, and y_X is the total rate of the XCP traffic traversing the link.

Figure 13 shows the throughputs of various combinations of competing TCP and XCP flows normalized by the fair share. The bottleneck capacity is 45 Mb/s and the common delay is 40 ms. The simulations results demonstrate that XCP is as TCP-friendly as other protocols that are currently under consideration for deployment in the Internet [12].

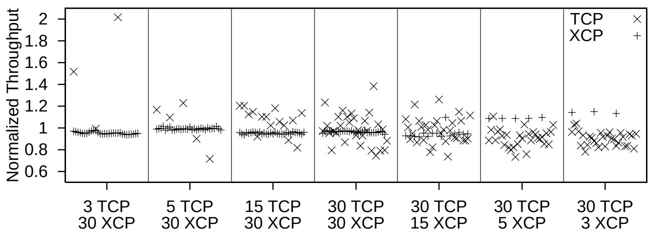

Figure 13: XCP is TCP-friendly.

9 Related Work

XCP builds on the experience learned from TCP and previous research in congestion control [6, 10, 13, 16]. In particular, the use of explicit congestion feedback has been proposed by the authors of Explicit Congestion Notification (ECN) [28]. XCP generalizes this approach so as to send more information about the degree of congestion in the network. Also, explicit congestion feedback has been used for controlling Available Bit Rate (ABR) in ATM networks [3, 9, 17, 18]. However, in contrast to ABR flow control protocols, which usually maintain per-flow state at switches [3, 9, 17, 18], XCP does not keep any per-flow state in routers. Further, ABR control protocols are usually rate-based, while XCP is a window-based protocol and enjoys self-clocking, a characteristic that considerably improves stability [7].

Additionally, XCP builds on Core Stateless Fair Queuing (CSFQ) [29], which by putting a flow's state in the packets can provide fairness with no per-flow state in the core routers.

Our work is also related to Active Queue Management disciplines [13, 5, 21, 15], which detect anticipated congestion and attempt to prevent it by taking active counter measures. However, in contrast to these schemes, XCP uses constant parameters whose effective values are independent of capacity, delay, and number of sources.

Finally, Our analysis is motivated by previous work that used a control theory framework for analyzing the stability of congestion control protocols [22, 27, 15, 21, 24].

10 Conclusions and Future Work

Theory and simulations suggest that current Internet congestion control mechanisms are likely to run into difficulty in the long term if current technology trends continue. This motivated us to step back and re-evaluate both control law and signalling for congestion control.

Motivated by CSFQ, we chose to convey control information between the end-systems and the routers using a few bytes in the packet header. The most important consequence of this explicit control is that it permits a decoupling of *congestion control* from *fairness control*. In turn, this decoupling allows more efficient use of network resources and more flexible bandwidth allocation schemes.

Based on these ideas, we devised XCP, an explicit congestion control protocol and architecture that can control the dynamics of the aggregate traffic independently from the relative throughput of the individual flows in the aggregate. Controlling congestion is done using an analytically tractable method that matches the aggregate traffic rate to the link capacity, while preventing persistent queues from forming. The decoupling then permits XCP to reallocate bandwidth between individual flows without worrying about being too aggressive in dropping packets or too slow in utilizing spare bandwidth. We demonstrated a fairness mechanism based on *bandwidth shuffling* that converges much faster than TCP does, and showed how to use this to implement both min-max fairness and the shadow prices model.

Our extensive simulations demonstrate that XCP maintains good utilization and fairness, has low queuing delay, and drops very few packets. We evaluated XCP in comparison with TCP over RED, REM, AVQ, and CSFQ queues, in both steady-state and dynamic environments with web-like traffic and with impulse loads. We found no case where XCP performs significantly worse than TCP. In fact when the per-flow delay-bandwidth product becomes large, XCP's performance remains excellent whereas TCP suffers significantly.

We believe that XCP is viable and practical as a congestion control scheme. It appears to make defense against denial-of-service attacks easier than in the current Internet. We have also proposed two possible strategies whereby XCP and TCP could gracefully co-exist in the same network.

Our future work focuses on two directions. First, XCP was motivated by our experience with TCP's degraded performance over gigabit links. We are currently implementing XCP in a FreeBSD kernel to use it over our gigabit testbed. Second, in this paper we developed a simple approach to differential services. We think that the decoupling can also make the design of guaranteed services easier and we are pursuing this idea.

References

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

[2] Red parameters. http://www.icir.org/floyd/red.html#parameters.

[3] Y. Afek, Y. Mansour, and Z. Ostfeld. Phantom: A simple and effective flow control scheme. In *Proc. of ACM SIGCOMM*, 1996.

[4] M. Allman, D. Glover, and L. Sanchez. Enhancing tcp over satellite channels using standard mechanisms, Jan. 1999.

[5] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. Rem: Active queue management. *IEEE Network*, 2001.

[6] D. Bansal and H. Balakrishnan. Binomial congestion control algorithms. In *Proc. of IEEE INFOCOM '01*, Apr. 2001.

[7] D. Bansal, H. Balakrishnan, and S. S. S. Floyd. Dynamic behavior of slowly-responsive congestion control algorithms. In *Proc. of ACM SIGCOMM*, 2001.

[8] J. Border, M. Kojo, J. Griner, and G. Montenegro. Performance enhancing proxies, Nov. 2000.

[9] A. Charny. An algorithm for rate allocation in a packet-switching network with feedback, 1994.

[10] 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.

[11] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *IEEE/ACM Transactions on Networking, 5(6):835–846*, Dec. 1997.

[12] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of ACM SIGCOMM*, Aug. 2000.

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

[14] R. Gibbens and F. Kelly. Distributed connection acceptance control for a connectionless network. In *Proc. of the 16th Intl. Telegraffic Congress*, June 1999.

[15] C. Hollot, V. Misra, D. Towsley, , and W. Gong. On designing improved controllers for aqm routers supporting tcp flows. In *Proc. of IEEE INFOCOM*, Apr. 2001.

[16] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium*, 18, 4:314–329, Aug. 1988.

[17] 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. In *The Ohio State University, Department of CIS*, Jan. 1997.

[18] 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*, Dec. 1997.

[19] D. Katabi and C. Blake. A note on the stability requirements of adaptive virtual queue. MIT Technichal Memo, 2002.

[20] F. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability.

[21] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue. In *Proc. of ACM SIGCOMM*, 2001.

[22] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle. Dynamics of tcp/aqm and a scalable control. In *Proc. of IEEE INFOCOM*, June 2002.

[23] R. Manajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. under submission to CCR, July 2001.

[24] 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. Aug. 2000.

[25] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. Long thin networks, Jan. 2000.

[26] Names and affiliations are removed for anonymity. Precise feedback for congestion control in the internet. Technical Report, 2001.

[27] F. Paganini, J. C. Doyle, and S. H. Low. Scalable laws for stable network congestion control. In *IEEE CDC*, 2001.

[28] K. K. Ramakrishnan and S. Floyd. Proposal to add explicit congestion notification (ecn) to ip. RFC 2481, Jan. 1999.

[29] 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. of ACM SIGCOMM*, Aug. 1998.

A Implementation

Implementing an XCP router is fairly simple and is best described using the following pseudo code. There are three relevant blocks of code. The first block is executed at the arrival of a packet and involves updating the estimates maintained by the router.

```
On packet arrival do:
    input_traffic += 1
    sum_rtt_by_cwnd += H_rtt / H_cwnd
    sum_rtt_square_by_cwnd += H_rtt × H_rtt / H_cwnd
```

The second block is executed when the estimation-control timer fires. It involves updating our control variables, reinitializing the estimation variables, and rescheduling the timer.

```
On estimation-control timeout do:
    avg_rtt = sum_rtt_square_by_cwnd / sum_rtt_by_cwnd[8]
    $\phi = \alpha\times$ avg_rtt × (capacity - input_traffic) - $\beta\times$ Queue
    shuffled_traffic = $0.1\times$ input_traffic
    $\xi_p$ = ((max($\phi$,0) + shuffled_traffic) / (avg_rtt × sum_rtt_by_cwnd)
    $\xi_n$ = ((max($-\phi$,0) + shuffled_traffic) / (avg_rtt × input_traffic)
    residue_pos_fbk = (max($\phi$,0)+ shuffled_traffic) /avg_rtt
    residue_neg_fbk = (max($-\phi$,0)+ shuffled_traffic) /avg_rtt
    input_traffic = 0
    sum_rtt_by_cwnd = 0
    sum_rtt_square_by_cwnd = 0
    timer.schedule(avg_rtt)
```

The third block of code involves computing the feedback and is executed at packets' departure.

```
On packet departure do:
    pos_fbk = $\xi_p\times$ H_rtt × H_rtt / H_cwnd
    neg_fbk = $\xi_n\times$ H_rtt
    feedback = pos_fbk - neg_fbk
    if (H_feedback $\geq$ feedback) then
        H_feedback = feedback
        residue_pos_fbk -= pos_fbk / H_rtt
        residue_neg_fbk -= neg_fbk / H_rtt
    else
        if (H_feedback $\geq$ 0)
            residue_pos_fbk -= H_feedback / H_rtt
            residue_neg_fbk -= (feedback - H_feedback) / H_rtt
        else
            residue_neg_fbk += H_feedback / H_rtt
            if (feedback $\geq$ 0) then residue_neg_fbk -= feedback/H_rtt
    if (residue_pos_fbk $\leq$ 0) then $\xi_p = 0$
    if (residue_neg_fbk $\leq$ 0) then $\xi_n = 0$
```

Note that the code executed on timeout does not fall on the critical path. The per-packet code can be made substantially faster by replacing `cwnd` in the congestion header by `rtt/cwnd`, and by having the routers return $feedback \times H\_rtt$ in $H\_feedback$ and the sender dividing this value by its `rtt`. This modification spares the router any division operation, **in which case, the router does only a few additions and 3 multiplications per packet.**

[8]This is the average RTT over the flows (not the packets).

B Proof of Theorem 1

Model: Consider a single link of capacity c traversed by N XCP flows. Let d be the common round trip delay of all users, and $r_i(t)$ be the sending rate of user i at time t. The aggregate traffic rate is $y(t) = \sum r_i(t)$. The shuffled traffic *rate* is $h(t) = 0.1 \cdot y(t)$.[9]

The router sends some aggregate feedback every control interval d. The feedback reaches the sources after a round trip delay. It changes the sum of their congestion windows (i.e., $\sum w(t)$). Thus, the aggregate feedback sent per time unit is the sum of the derivatives of the congestion windows:

$$\sum \frac{dw}{dt} = \frac{1}{d}\Big(-\alpha \cdot d \cdot (y(t-d) - c) - \beta \cdot q(t-d)\Big).$$

Since the input traffic rate is $y(t) = \sum \frac{w_i(t)}{d}$, the derivative of the traffic rate $\dot{y}(t)$ is:

$$\dot{y}(t) = \frac{1}{d^2}\Big(-\alpha \cdot d \cdot (y(t-d) - c) - \beta \cdot q(t-d)\Big).$$

Ignoring the boundary conditions, the whole system can be expressed using the following delay differential equations.

$$\dot{q}(t) = y(t) - c \tag{15}$$

$$\dot{y}(t) = -\frac{\alpha}{d}(y(t-d) - c) - \frac{\beta}{d^2}q(t-d) \tag{16}$$

$$\dot{r}_i(t) = \frac{1}{N}([\dot{y}(t-d)]^+ + h(t-d)) - \frac{r_i(t-d)}{y(t-d)}([-\dot{y}(t-d)]^+ + h(t-d)) \tag{17}$$

The notation $[\dot{y}(t-d)]^+$ is equivalent to $\max(0, \dot{y}(t-d))$. Equation 17 expresses the AIMD policy used by the FC; namely, the positive feedback allocated to flows is equal, while the negative feedback allocated to flows is proportional to their current throughputs.

Stability: Let us change variable to $x(t) = y(t) - c$.
Proposition: The Linear system:

$$\dot{q}(t) = x(t)$$

$$\dot{x}(t) = -K_1 x(t-d) - K_2 q(t-d)$$

is stable for any constant delay $d > 0$ if

$$K_1 = \frac{\alpha}{d} \quad \text{and} \quad K_2 = \frac{\beta}{d^2},$$

where α and β are any constants satisfying:

$$0 < \alpha < \frac{\pi}{4\sqrt{2}} \quad \text{and} \quad \beta = \alpha^2\sqrt{2}.$$

Proof. The system can be expressed using a delayed feedback (see Figure 14. The open loop transfer function is:

$$G(s) = \frac{K_1 \cdot s + K_2}{s^2} e^{-ds}$$

For very small $d > 0$, the closed-loop system is stable. The shape of its Nyquist plot, which is given in Figure 15, does not encircle -1.

[9]We are slightly modifying our notations. While $y(t)$ in § 3.5 refers to the input traffic in an average RTT, we use it here as the input traffic rate (i.e., input traffic in a unit of time). The same is true for $h(t)$.

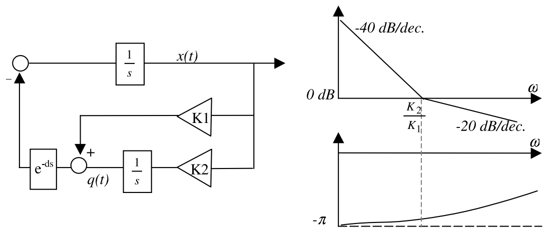

Figure 14: The feedback loop and the Bode plot of its open loop transfer function.

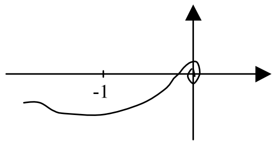

Figure 15: The Nyquist plot of the open-loop transfer function with a very small delay.

Next, we can prove that the phase margin remains positive independent of the delay. The magnitude and angle of the open-loop transfer function are:

$$|G| = \frac{\sqrt{K_1^2 \cdot w^2 + K_2^2}}{w^2}$$

$$\angle G = -\pi + \arctan \frac{wK_1}{K_2} - w \cdot d$$

The break frequency of the zero occurs at: $w_z = \frac{K_2}{K_1}$.

To simplify the system, we decided to choose α and β such that the break frequency of the zero w_z is the same as the crossover frequency w_c (frequency for which $|G(w_c)| = 1$). Substituting $w_c = w_z = \frac{K_2}{K_1}$ in $|G(w_c)| = 1$ leads to $\beta = \alpha^2\sqrt{2}$.

To maintain stability for any delay, we need to make sure that the phase margin is independent of delay and always remains positive. This means that we need $\angle G(w_c) = -\pi + \frac{\pi}{4} - \frac{\beta}{\alpha} > -\pi \Rightarrow \frac{\beta}{\alpha} < \frac{\pi}{4}$. Substituting β from the previous paragraph, we find that we need $\alpha < \frac{\pi}{4\sqrt{2}}$, in which case, the gain margin is larger than one and the phase margin is always positive (see the Bode plot in Figure 14). This is true for any delay, capacity, and number of sources. □

C XCP robustness to High Variance in the Round Trip Time

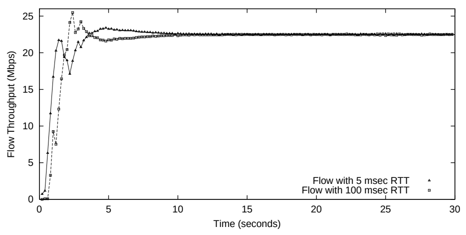

Figure 16: XCP robustness to high RTT variance. Two XCP flows each transferring a 10 Mbytes file over a shared 45 Mb/s bottleneck. Although the first flow has an RTT of 20 ms and the second flow has an RTT of 200 ms both flows converge to the same throughput. Throughput is averaged over 200 ms intervals.