PCP: Efficient Endpoint Congestion Control

Thomas Anderson, Andrew Collins, Arvind Krishnamurthy and John Zahorjan
University of Washington

Abstract

In this paper, we present the design, implementation, and evaluation of a novel endpoint congestion control system that achieves near-optimal performance in all likely circumstances. Our approach, called the Probe Control Protocol (PCP), emulates network-based control by using explicit short probes to test and temporarily acquire available bandwidth. Like TCP, PCP requires no network support beyond plain FIFO queues. Our initial experiments show that PCP, unlike TCP, achieves rapid startup, small queues, and low loss rates, and that the efficiency of our approach does not compromise eventual fairness and stability. Further, PCP is compatible with sharing links with legacy TCP hosts, making it feasible to deploy.

1 Introduction

The efficient and fair allocation of distributed resources is a longstanding problem in network research. Today, almost all operating systems use TCP congestion control [27] to manage remote network resources. TCP assumes no network support beyond that packets are dropped when the network is overloaded; it uses these packet losses as a signal to control its sending rate. Provided that all endpoints use a compatible algorithm, persistent congestion can be averted while still achieving good throughput and fair allocation of the shared resource.

However, it has long been understood that TCP is far from optimal in many circumstances. TCP managed networks perform poorly for moderate sized flows on idle links [12, 28], interactive applications [21], applications demanding minimally variable response times [13], high bandwidth-delay paths [32], and wireless networks [5]. In each case, the response in the cited papers has been to propose explicit network support. Clearly, network-based resource allocation can be designed to perform optimally. Thus, the research debate has largely focused on the appropriate knobs to place in the network, and specifically, the tradeoff between simplicity and optimality in network support for resource management.

We take a radically different approach. Our goal is to demonstrate that cooperating endpoints, without any special support from the network, can achieve near optimal resource allocation in all likely conditions. Our motivation is partly intellectual – what are the algorithmic limits to endpoint resource allocation? Our motivation is also partly practical – it is much easier to deploy an endpoint solution than to modify every router in the Internet. Since TCP congestion control was first introduced in 1988, three substantial changes to its algorithm have been introduced and widely adopted [38]; by contrast, to date no router-based changes to congestion management have achieved equally widespread use.

Our approach is simple: we directly emulate network-based control. Our algorithm, called the Probe Control Protocol (PCP), sends a short sequence of probe packets at a specific rate to detect whether the network can currently support the test rate, given its current traffic load. If so, the endpoint sends at that rate; if not, e.g., if other hosts are already using the bandwidth, the endpoint tries a new probe at a slower rate. A key element is that we use rate pacing (instead of TCP-like ack clocking) for all traffic; this allows probes to be short and precise. These short probes are "low impact" compared to the TCP approach of sending at the target rate for the full round trip time. Thus, endpoints can be aggressive with respect to testing for available bandwidth, without causing packet loss for existing flows. For example, a new arrival can use history to safely guess the currently available bandwidth. Since most links are idle most of the time, this often allow endpoints to jump (almost) immediately to full utilization of the link.

We have implemented PCP as a user-level process on Linux. Initial tests on the RON testbed show that PCP can outperform TCP by an average of a factor of two for 200KB transfers over the wide area, without having any measurable impact on competing

	Endpoint	Router Support
Try and Backoff	TCP [27], Vegas [10] RAP [45], FastTCP [31] Scalable TCP [34] HighSpeed TCP [19]	DecBit [44], ECN [18] RED [21], AQM [9]
Request and Set	PCP	ATM [4, 46], XCP [32] WFQ [14], RCP [15]

Table 1: Congestion Control Design Space

TCP traffic. We supplement these results with simulation experiments, where we show that PCP achieves our goals of near optimal response time, near zero packet loss rates, and small router queues, across a wide variety of operating environments; we further show that it has better fairness properties than TCP. A Linux kernel implementation is also currently in progress. Interested readers can obtain the source codes for the implementations and the simulation harness at http://www.cs.washington.edu/homes/arvind/pcp.

A practical limitation of our work is that, like TCP, we provide no means to counter misbehaving hosts; network based enforcement such as fair queueing [14] is still the only known means to do so. We show that PCP, unlike TCP, benefits from fair queueing, thus making it the first system to do well for both FIFO and fair-queued routers. Any future network is likely to have a mixture of FIFO and fair-queued routers, making an endpoint solution compatible with all a necessity for network evolution. In our view, TCP's poor performance over fair-queuing routers is a barrier to further deployment of router enforcement. By contrast, PCP can be seen as a stepping stone for more robust isolation mechanisms inside the network, thereby improving the overall predictability of network performance.

The rest of this paper presents our approach in more detail. Section 2 outlines the goals of our work, arguing that TCP is sub-optimal in many common network conditions found today. Section 3 describes the design of the PCP algorithm. We evaluate our approach in Section 4, discuss related work in Section 5, and summarize our results in Section 6.

2 Design Goals

Table 1 outlines the design space for congestion control mechanisms. We argue in this section that PCP explores a previously unstudied quadrant of the design space – endpoint emulation of optimal router-based control. If we were to start from scratch, the design goals for a congestion control algorithm would be clear:

- Minimum response time. The average time required for an application transfer to complete should be as small as possible. Since most transfers are relatively short, startup efficiency is particularly important [16].
- Negligible packet loss and low queue variability. Because sources in a distributed system cannot distinguish between the root causes of packet loss, whether due to media failure, destination unavailability, or congestion, it is particularly important to avoid adding to that uncertainty. Similarly, large queueing delays unnecessarily delay interactive response time and disrupt real-time traffic.
- Work conserving. In steady state, resources should not be left idle when they might be used to send data.
- Stability under extreme load. Aggregate performance should approach physical limits, and per-flow performance should degrade gracefully as load is added to the system.
- Fairness. Competing connections (or *flows*) which are not otherwise limited should receive equal shares of the bottleneck bandwidth. At the very least, no flow should starve due to competing flows.

Note that network-based congestion control can easily achieve all of these goals [4, 46]. With ATM, for example, endpoints send a special rate control message into the network to request bandwidth, enabling the bottleneck switch or router to explicitly allocate its scarce capacity among the competing demands. We call this approach "request and set" because endpoints never send faster than the network has indicated. Response time is minimized because fair share is communicated in the minimum possible time, a round trip. This also results in queues being kept empty and bandwidth being allocated fairly. Our goal is to see if we can achieve all these properties without any special support from the network [47], by emulating "request and set" mechanics from endpoints.

By contrast, TCP congestion control achieves the last three goals [27] but not always the first two. TCP carefully coordinates how the sending rate is adjusted upwards and downwards in response to successful transmissions and congestion signals. We call this approach "try and backoff" since an end host sends traffic into the network without any evidence that the network has the capacity to accept it; only when there is a problem, that is, a packet loss, does the end host reduce its rate.

Since a TCP endpoint has no knowledge of the true available bandwidth, it initially starts small and through a series of steps called slow start, drives the network to saturation and packet loss, signaling the capacity limit of the network. Although effective, this process can waste bandwidth on startup – asymptotically $O(n \log n)$ in terms of the path's bandwidth delay product [12]. (To be fair, TCP congestion control was designed at a time when links were thin and usually fully utilized; in these situations the efficiency loss of slow start is minimal.) Further, TCP's slow start inefficiency is fundamental. Several proposals have been made for methods to jump start the initial window size, but they run the risk of causing increased packet losses in situations where there is persistent congestion [19].

Once a TCP endpoint determines the available bandwidth, in theory the link will be fully utilized, amortizing the initial inefficiency for a sufficiently long connection. Of course, many flows are short. Even for long flows, TCP steady state behavior can be disrupted by the bursty traffic pattern emitted by other flows entering and exiting slow start. In practice, TCP may achieve only a fraction of the available bandwidth, because of the need to slowly increase its sending rate after a loss event to avoid persistent congestion [30, 32]. Similar problems occur with TCP in the presence of noise-induced loss, such as with wireless links [5].

Some researchers have studied how to modify end hosts to improve TCP performance, while keeping its basic approach. For example, TCP Vegas [10], FastTCP [31], Scalable TCP [34], and HighSpeed TCP [19] all attempt to improve TCP steady state dynamics. Vegas and FastTCP are similar to PCP in that they use packet delay to guide congestion response, but unlike PCP, they do so only after sending at the target rate for the entire round trip time. And because all of these alternate approaches leave slow start unchanged, they only help the small fraction of transfers that reach steady state.

Other researchers have explored adding TCP-specific support to routers. For example, routers can drop packets before it becomes absolutely necessary [21, 9], as a way of signalling to end hosts to reduce their sending rates. However, this trades increased packet losses in some cases for lower delay in others; most users want both low loss and low delay. Some have advocated setting a bit in a packet header to signal congestion back to the sender [44, 18], but this does nothing to address the slow start inefficiency. This has led some to advocate adding an ATM-style rate control message to the Internet to allow for more rapid TCP startup [42, 15]. And so forth.

Our approach is to explore the opposite quadrant in Table 1. Is it possible to achieve all five goals using only endpoint control, by emulating the "request and set" semantics of explicit router-based resource allocation? In doing so, we hope to design a system that is better suited to the tradeoffs we face today vs. what TCP faced fifteen years ago. In designing our system, note that we place the first two goals listed above ahead of the last three, in priority. While we want our system to to be efficient, stable and fair under high load, we also want our system to behave well in the common case.

The common case is that most network paths are idle most of the time, and are becoming more so over time [41, 2]. This was not always true! Rather, it is the natural consequence of the cumulative exponential improvement in the cost-performance of network links—at least in industrialized countries, it no longer makes sense for humans to wait for networks. By contrast, when TCP congestion control was initially designed, wide area network bandwidth cost over one thousand times more than it does today; at that price, fairness would naturally be more important than improving connection transfer time. The opposite holds today.

Second, even with HTTP persistent connections, most Internet transfers never reach TCP steady state [22, 48]. Even when they do, startup effects often dominate performance [12]. For example, a 1MB cross-country transfer over fast Ethernet can achieve an effective throughput with TCP of only a few Mbps, even with no other flows sharing the path. Home users are more frequently bandwidth-limited, but even here, TCP is not well-suited to a highly predictable environment with little multiplexing.

Third, computation and memory are becoming cheaper even faster than wide area network bandwidth. TCP was originally designed to avoid putting a multiplication operation in the packet handler [27], yet at current wide area bandwidth prices, it costs (in dollars) the same amount to send a TCP ack packet as to execute half a million CPU instructions [25]. One consequence is that hardware at aggregation points is increasingly limited by TCP mechanics: to remain TCP "friendly" the aggregation point must not send any faster than k parallel TCP connections [6], something that is only efficient if there are multiple active flows to the same destination. By contrast, PCP can benefit directly from an endpoint's excess cycles and memory by modeling the likely behavior of a network path, even if the path has not been recently used.

Finally, our approach integrates better with constant rate real-time traffic. Without router support, TCP's continual attempts to overdrive and backoff the bottleneck link can disrupt fixed-rate flows that are sharing the link, by introducing packet delay jitter and loss. To some extent, this problem can be reduced by sophisticated active queue management (AQM) [24]. Lacking widespread deployment of AQM systems, most ISP's today have abandoned the vision of integrated services—they provision logically separate networks to carry voice over IP as distinct from regular web traffic—as the only practical way to achieve quality of service goals. By contrast, in PCP, best effort traffic will normally have very little impact on background fixed-rate traffic, again without any special hardware support.

Mechanism	Description	Goal	Section
probes	Senders use short transmission bursts with limited payload to “prove” the existence of available bandwidth, while minimizing any long term effects of failed tests.	low loss	Section 3.1
direct jump	Given a successful test, senders increase their base rate to the rate that test.	min response time	Section 3.1
probabilistic accept	Accept tests taking into account the variance observed in the available bandwidth measurements.	fairness	Section 3.1
rate compensation	When existing senders detect increasing queueing, they reduce their rates to drain the queue.	low loss, low queues	Section 3.2
periodic probes	Senders periodically issue new probes to try to acquire additional bandwidth.	work-conserving	Section 3.3
binary search	Senders use binary search to allocate the available bandwidth.	min response time, work conserving	Section 3.3
exponential backoff	Senders adjust the frequency of tests to avoid test collisions and failures.	stability	Section 3.3
history	Senders use heuristics to choose the initial probe rate.	min response time	Section 3.4
tit for tat	Reduce speed of rate compensation if past compensation was ineffective.	TCP compatibility	Section 3.5

Table 2: A Summary of PCP Mechanisms

To sum, TCP is optimized for the wrong case. Like TCP, the design we describe in this paper provides robust congestion control for a wide range of operating conditions. But equally importantly, our approach is better than TCP for the common case: moderate-sized transfers over mostly idle links with near-zero loss and low delay even when there is congestion. Of course, network-based congestion control solutions have already been shown to provide these characteristics; our point is simply to demonstrate that we can achieve these goals without network support.

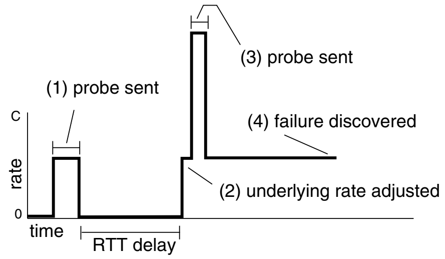

Figure 1: Example of a successful and a failed PCP probe.

3 PCP Design

In this section, we sketch the various elements of the PCP design. Table 2 provides a road map. For this discussion, we assume that PCP is used by all endpoints in the system to manage network resources; we defer to the end of this section a discussion of how to make PCP backwardly compatible with TCP end hosts. PCP represents a clean-slate redesign to endpoint congestion control; we do however retain the TCP mechanisms for connection management and flow control.

3.1 Emulating Request-and-Set

PCP is very simple at its most basic (Figure 1): endpoints send *probe packets*, which are short sequences of packets spaced at a target test rate, to determine if the network has the capacity to accommodate the request. If this *probe* is successful, the end host can immediately increase its base rate by the target rate of the probe; it then transmits the *baseline packets* in a *paced* (equally spaced) manner at the new *base rate*. The probe rate is adjusted (primarily) by changing the duration over which the probe data is sent, not the amount of data that is sent. If the initial probe is unsuccessful (e.g., the network does not have the spare capacity), the end host must try again. A key element of our approach is that endpoints only increase their base sending rates immediately after a successful probe, and at no other time; thus, modulo the round trip delay, a successful probe indicates that the network resource is unallocated.

Probe success and failure is defined by whether the probe packets induce queueing inside the network, measured by whether the delay increases during the probe. For each PCP data packet, a timestamp is recorded at the receiver and returned to the sender, akin to the TCP timestamp option [26]; measuring changes in delay at the receiver allows us to eliminate variability induced by the reverse path. Our test for available bandwidth is similar to the one suggested by Jain and Dovrolis [29], but is designed to be more robust to the noise caused by PCP's probe process. Specifically, we use least squares to fit a line through the sequence of delay measurements and accept the test rate if the measurements are consistent with flat or decreasing delay; otherwise, the probe is rejected. Furthermore, since measurements are rarely determinative of the true state of the network, we use a "probabilistic accept" rule that accepts probes nondeterministically. The least squares fit yields a probability distribution function characterized by the estimated delay slope and a standard error for the estimated value. A low error indicates a good fit, while a high value might be due to measurement noise or variability in cross traffic. We randomly choose whether to accept a probe based on this probability distribution.

We do not assume a hard real-time operating system implementation; some jitter is acceptable in the scheduling of packets as the fitting process is robust to small variations in packet spacing. We however assume the availability of fine-grained clocks; nanosecond clocks and timers have become commonplace on modern processors [40], sufficient for scaling to gigabit speeds. Generic timestamp and packet pacing logic is also becoming increasingly common on network interface hardware.

To enable an apples to apples comparison, we set the initial probe size to match the initial TCP packet. After an initial packet exchange to verify the receiver is willing to accept packets (a mirror of the TCP SYN exchange), PCP sends, as its first probe, data equal to a maximum size packet, but divided into k separate chunks (currently, 5). Probe packets may carry live data, so that if the data to be transferred is sufficiently small, it may complete within the probe packets – that is, whether or not the network can accept packets indefinitely at the probe rate. Although some have proposed using a larger initial window size in TCP to speed its discovery of the available bandwidth for high capacity paths, this would come at the potential cost of added congestion on low capacity paths. By separating the probe *rate* from the *size* of the probe, PCP avoids having to make this trade-off; as we describe below, we can safely use history to select an aggressive rate for the initial test.

If the probe is successful, PCP immediately jumps to the requested rate. As a minor optimization, once an end host is successful in obtaining sufficient bandwidth, we convert to using k maximum sized packets as probes, again, paced at the target bit rate. This provides better accuracy for high bandwidth paths.

3.2 Rate compensation

A key challenge for PCP is to gracefully eliminate queues that might build up at a bottleneck router. Queues can build up for several reasons. One obvious cause is failed probes. If all of the bottleneck bandwidth has been allocated, any additional probe will induce queueing that will not disappear until some flow reduces its bandwidth. As more failed probes accumulate, the queues could slowly build to the point where packet loss is inevitable. A more severe cause is due to the time lag between when an end host makes a probe and when it can allocate the bandwidth. In PCP, the request and set operations are not atomic. If two or more hosts send a probe at approximately the same time, both probes may succeed, resulting in duplicate allocation of the same bandwidth. In this case, the link may be over committed, and unless one or both hosts reduce their rates, queues will quickly build up to cause packet loss.

Fortunately, it is straightforward for PCP to detect queueing at the bottleneck. Recall that once an end host allocates bandwidth, it sends its data as paced packets at the base rate. If there is no queueing at the bottleneck, the delays for these data packets will be regular. Any increase in the delay indicates a queue, requiring a rate reduction to eliminate. Note that complex link layer arbitration, as in 802.11, is perfectly compatible with PCP; any added delay in those systems is an indication of queueing – that the endpoints are sending faster than the underlying network can handle.

Eliminating queues caused by PCP endpoints is also easy. Whenever a queue is detected, all existing senders proportionately reduce their rate sufficiently to eliminate the queue over the next round trip. We call this process, *rate compensation*. Eliminating the queue over one round trip is more aggressive than is strictly required by control theory [32], but in our system, any queueing is an indication of resource contention. Under contention, proportionate decrease among existing senders, and uniform competition for newly available bandwidth among all senders, helps achieve eventual fairness.

We use two mechanisms to detect over-committed network links. First, we monitor the gap between baseline PCP packets as they enter and exit a network path. If the time gap observed at the receiver (Δ_{out}) is greater than the spacing Δ_{in} used by the sender, the bottleneck link is likely to be overloaded. To avoid making matters worse, we reduce the base sending rate by a factor of $(\Delta_{out} - \Delta_{in})/\Delta_{out}$. Second, in order to drain the queue, we monitor the one-way delays experienced by PCP packets. If the maximum one-way delay (*max-delay*) observed in the previous round trip time is greater than the minimum observed one-way delay to the destination (*min-delay*), then there is persistent queueing at the bottleneck link. To eliminate the queue build-up, we reduce the sending rate by a factor of (*max-delay* - *min-delay*)/*max-delay*. In both cases, we bound the proportionate decrease to the TCP backoff rate – no more than half of the prior rate during any round trip. (We concern ourselves only with the measurements during the previous round trip as prior rate compensation is presumed to have eliminated the overloads during prior round trips.) If all senders detect queueing and reduce their rate by this proportion, and no further probes are launched, it is easy to show that the queue will disappear within one round trip time. Senders with much shorter round trip times will reduce their rate more quickly, shouldering more of the burden of keeping queues small, but they will also be able to acquire bandwidth more quickly by probing for new bandwidth at a faster rate.

Once the base rate is reduced, probes may successfully re-acquire the bandwidth. These probes may be launched either by other nodes, or even by the reducing node itself. This is done to foster additive increase, multiplicative decrease behavior when there is contention. If the queues do not dissipate, the existing senders will continue to proportionally decrease their rates. Some combination of flows will acquire the released bandwidth.

A detail is that we apply the rate compensation incrementally, after every acknowledged packet, by comparing the required rate compensation to the rate reductions that have already been applied over the previous round trip time. If the new rate compensation is larger, we reduce the sending rate by the difference. This is similar to a single rate adjustment made once per round trip time, but operates at a finer granularity. Further, to reduce the impact of the noise on the system, we discard outliers represented by either the lowest 10% or the highest 10% of the measured values.

Another detail is that Internet routing changes can transparently alter the baseline one-way packet delay. Although some have called for providing endpoints the ability to detect when their packets have been rerouted [36], that facility is not available on the Internet today. There are two cases. If the routing change decreases the baseline delay, the node will update its *min-delay*, observe there is a difference between the new *min-delay* and the previous *max-delay*, and proceed to reduce its rate by at most one half. Behavior will then revert to normal after one round trip, and the end host will be free to probe to acquire bandwidth on the new path. If the routing change increases the baseline delay, the node will see an increase in its *max-delay* and likewise reduce its rate in an attempt to compensate. This reduction will dissipate after *min-delay* has timed out. Note that the probe process is independent of rate compensation; probe success is based on the measured increase in delay during the probe, and not on the long term estimation of the queue. Thus, as long as the new path has spare capacity, the end host will be able to quickly re-acquire its released bandwidth. Both types of routing changes result in temporarily reduced efficiency, but with the positive side effect that the affected endpoints are less aggressive exactly when the state of the network is in flux.

3.3 Probe control

Because probes carry limited payload and rate compensation corrects for any mistakes that occur, an end host can be aggressive in selecting its probe rates. For example, it can pick the probe rate that maximizes its expected yield – the likelihood of the probe's success times the requested rate. Unless there are long periods of full utilization of the network, it is better for all concerned for an arriving flow to quickly grab all available bandwidth, complete the transfer, and exit the system, leaving future resources for future arrivals. Of course, we do want the system to be work-conserving, stable and fair under persistent congestion, and thus we need to introduce additional mechanism in PCP to accomplish those goals. That is the topic of this sub-section. Note however that if high load behavior is your only concern (e.g., response time and loss rate are unimportant), TCP's current behavior is adequate, and our design would offer few benefits.

In the absence of any other information, we set the initial target probe rate to be one maximum sized packet in half of the round trip time, as measured by the initial connection establishment (TCP SYN) packet exchange. If successful, during the next round trip the end host can send its base rate packets at that probe rate – two maximum sized packets per round trip time. It may also continue probing.

In our initial prototype, we use exponential increase and decrease to guide the search process, doubling the attempted rate increase after each successful probe, and halving the rate increase after each unsuccessful one. This guarantees that, regardless of the starting point or the capacity of the link, an end host fully allocates the available bandwidth in $O(\log n)$ steps. (A further optimization, which we have not yet implemented, is to use the slope of the response times from a failed probe to guess the next probe rate – in essence, the slope tells us by how much we have overestimated the available bandwidth. This will be particularly important for capacity-constrained paths, as the initial round trip time as measured by the small connection establishment packet, may vastly underestimate the round trip time for a maximally sized packet.) Note that while we never conduct more than one probe per measured round trip time, if the available bandwidth is small enough, we may stretch a single sequence of probe packets across multiple round trips. Thus, PCP gracefully scales down to very low bandwidth and/or oversubscribed paths. By contrast, TCP has a minimum window size of one packet; this can result in very high packet loss rates for paths where each flow's share is less than a single packet per round trip [39, 37].

Since we do not want nodes to continuously probe unsuccessfully for bandwidth, we place a lower bound on the rate that a flow can request from the network, at 1% of its current base rate. Once an existing sender has failed to acquire its minimum rate, it exponentially reduces its frequency of probing, up to a limit of 100 round trips. Within this interval, the probe is placed randomly. This is analogous to Ethernet, for a similar purpose. Removing the limit would improve theoretical scalability, but at a cost of reduced efficiency in acquiring recently released bandwidth.

We further note that our probabilistic accept rule for probes allows a probe to succeed even if it causes a small amount of queueing, thereby allowing new connections to succeed at receiving small amounts of bandwidth and triggering incumbent flows to release their bandwidth due to rate compensation. As a side effect, the rule also improves efficiency under heavy load by keeping the queue non-empty [10]. This behaves much like router-based active queue management (AQM), but controlled from the endpoint. A side effect, discussed below, is that the rule also makes PCP more robust when competing with legacy TCP flows.

3.4 History Information

As described so far, PCP seems merely to have replicated the delays caused by TCP slow start – $O(\log n)$ steps to determine the bandwidth. However, we note that the impact of a PCP probe is independent of its test rate, in contrast to TCP's approach of sending at its target rate for a full round trip time. Thus, it is easy to test aggressively in PCP, without fear of disrupting existing connections. We use this flexibility to reduce the startup transient to a constant number of round trips in the common case.

We achieve this by keeping *history information* about the base rates previously used to each Internet address. When this history information is available, we set the initial probe rate to be $1/3$ of the previous base rate, and then use binary search from that point forward. This allows the end host to usually identify the optimal rate within two round trip times after the initial connection establishment. We also keep track of the variance in base rates and the accuracy of predictions made based on the history information; if the history provides inaccurate estimate, we halve/double the initial probe rate after each inaccurate/accurate prediction, up to 1/3 of the base rate. Note that we do not set the initial rate to be the entire previous base rate, to avoid the chance that multiple hosts will simultaneously probe for, and seemingly acquire, the full link bandwidth. Our use of history is similar to that of the MIT Congestion Manager (CM) [6] but more general; because CM uses TCP, it can only reuse the congestion window if multiple transfers to the same location happen almost simultaneously. A mistakenly large congestion window in TCP could cause massive packet loss. Because the cost of making a mistake with history in PCP is only a wasted probe, we can make more aggressive use of potentially stale data of the likely available bandwidth of a path.

3.5 TCP Compatibility

To be feasible to deploy, PCP must be able to share network resources with existing TCP connections. Naively, as TCP increases its window size, queues will build up, and any PCP endpoints will reduce their sending rate to compensate. The TCP host will proceed unimpeded, continuing to increase its window size, causing further rate reductions from the PCP hosts.

We do not believe it is essential for PCP to be strictly fair with respect to TCP hosts, since that might well require simulating TCP's precise semantics. Rather, our goal is for PCP to be incentive compatible when sharing resources with TCP, so that it outperforms TCP while avoiding starvation for TCP hosts. Since PCP does much better than TCP for the common case of short to moderate transfers, there is substantial room for PCP to outperform TCP without needing to actively penalize TCP senders.

Our design walks this delicate balancing act. First, we recognize when a bottleneck is being shared with TCP, by observing when PCP's rate compensation is ineffective at reducing the queue size over several consecutive round trips. Normally PCP would continue to decrease its rate in the hope of eliminating the queue, but instead we apply a "tit for tat" rule, *decreasing* the rate compensation by a factor of ten for as long as rate compensation is ineffective. While this might seem counter-intuitive – increasing aggressiveness precisely at the point when congestion is building – "tit for tat" is needed to counter TCP's overly aggressive behavior. Eventually, the TCP connection will over-drive the link, causing loss and backoff for the TCP sender; PCP can then increase its rate during these periods. Our measurements and simulation results indicate that the TCP backoff and reduced PCP rate compensation balance out in most cases. When the TCP flow completes, any remaining PCP flows will find that rate compensation again becomes effective, enabling them to revert to their normal behavior.

In all other respects, PCP is backwardly compatible with legacy TCP hosts. We re-use the TCP packet header, indicating whether PCP should be used as an option in the TCP SYN packet. If the PCP receiver acknowledges the option, the sender uses PCP; otherwise we use traditional TCP congestion control.

There is no fundamental reason we cannot design a PCP sender to interoperate with an unmodified TCP re-

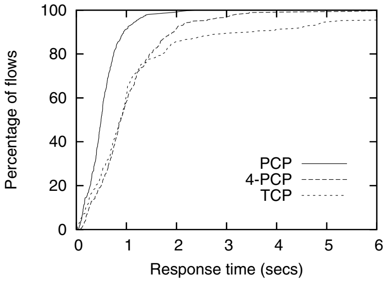

Figure 2: Cumulative distribution function of the average transfer time for the 380 wide-area paths in the RON testbed.

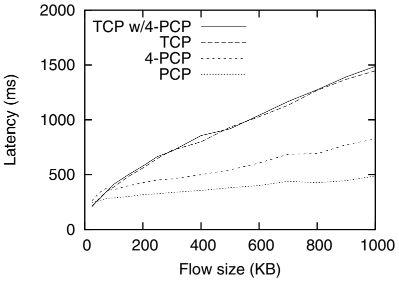

Figure 3: Transfer times for the user-level implementation.

ceiver. The principal difference between a PCP and a TCP receiver are in the precise semantics of timestamps and delayed acknowledgments. TCP timestamps are similar in theory to those used in PCP to measure one-way delay, but the TCP specification is tightly bound to the TCP retransmit timer logic. The TCP timestamp reflects the time that the *data* being acknowledged was received, not the time that the packet causing the acknowledgment was received. Instead, we plan to use round trip measurements to approximate one-way delay when interoperating with a TCP receiver. Similarly, PCP assumes that delayed acknowledgments are turned off; a PCP sender can disable the receiver's delayed acknowledgment logic by simply reordering every other packet or by sending all probe packets as doublets.

An interesting, and future, research question is whether we can design a PCP receiver to induce a TCP sender to use PCP congestion control. Savage et al. [49] have shown that a malicious receiver can abuse a sender's TCP control logic to cause it to send at an arbitrary rate; we believe we can leverage those ideas for inducing PCP compatibility with legacy TCP senders.

4 Evaluation

In this section, we first present data from a user-level implementation of PCP; we then use simulation to examine the behavior of PCP in more detail. Our results are both preliminary and incomplete; for example, we provide no study of the sensitivity of our results to the choice of PCP's internal parameters.

4.1 Performance Results from a User Level Implementation

This section presents data for PCP and TCP transfers over the Internet between twenty North American nodes selected from the RON testbed [3].

We implemented the PCP protocol in a user-level process; this is a conservative measure of PCP's effectiveness, since timestamps and packet pacing are less accurate when done outside the operating system kernel. To enable an apples-to-apples comparison, we also implemented TCP-SACK in the same user-level process. With TCP-SACK, selective acknowledgments give the sender a complete picture of which segments have been received without loss. The sender uses fast retransmit whenever it receives three out-of-order acknowledgments and then enters fast recovery. Our TCP implementation assumes delayed acknowledgments, where every other packet arriving within a short time interval of 200 ms is acknowledged. As in most other implementations [38], acknowledgments are sent out immediately for the very first data packet (in order to avoid the initial delayed ACK timeout when the congestion window is simply one) and for all packets that are received out-of-order. Removing delayed ACKs would improve TCP response time, but potentially at a cost of worse packet loss rates by making TCP's slow start phase more aggressive and overshooting the available network resources to a greater extent. We used RON to validate that our user-level implementation of TCP yielded similar results to native kernel TCP transfers for our measured paths.

For each pair of the twenty RON nodes, and in each direction, we ran three experiments: a single PCP transfer, a single TCP-SACK transfer, and four parallel PCP transfers. Each transfer was 250KB, repeated one hundred times and averaged. Figure 2 presents the cumulative distribution function of the transfer times for these 380 paths. PCP outperforms TCP in the common case because of its better startup behavior. The average PCP response time is 0.52 seconds; the average TCP response time is 1.33 seconds. To put this in perspective, four parallel PCP 250KB transfers complete on average in roughly the same time as a single 250KB TCP transfer. Further, worst case performance is much worse for TCP; while all PCP transfers complete within 2 seconds, over 10% of TCP transfers take longer than 5 seconds. While this could be explained by PCP being too aggressive relative to competing TCP traffic, we will see in the next graph that this is not the case. Rather, for congested links, TCP's steady state behavior is easily disrupted by background packet losses induced by shorter flows entering and exiting slow start.

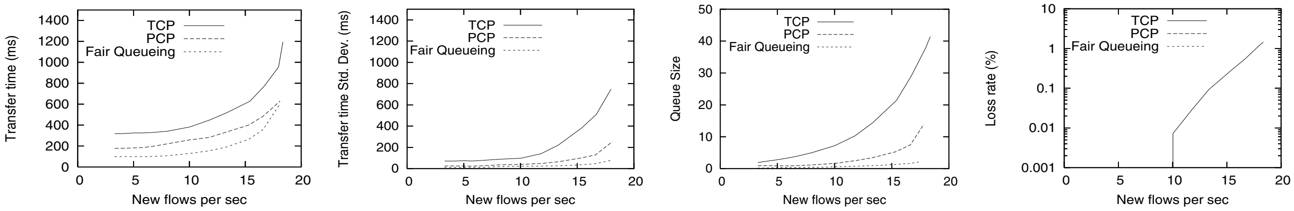

Figure 4: Performance of TCP, PCP and fair queueing. File transfer time includes connection setup. Bottleneck bandwidth is 40 Mb/s, average RTT is 25 ms, fix ed length flows of 250 KB. Fair queueing and PCP suffer no packet loss; the corresponding lines in the loss rate graph overlap with the x-axis.

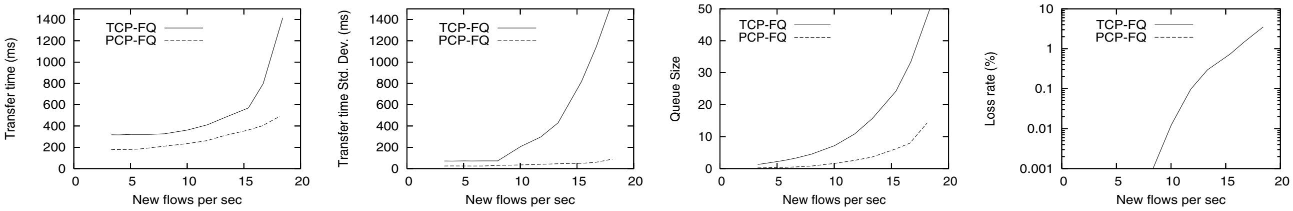

Figure 5: Performance of TCP and PCP through a bottleneck router that implements fair queueing. Bottleneck bandwidth is 40 Mb/s, average RTT is 25 ms, fix ed length flows of 250 KB. Fair queueing and PCP suffer no packet loss.

In Figure 3, we examine the behavior of PCP and TCP as a function of flow size, for a single pair of nodes. The TCP behavior is dominated by slow start for small transfers, and steady state behavior for large transfers; since this path has a significant background loss rate caused by other TCP flows, TCP is unable to achieve more than 5Mb/s in steady state. By contrast, PCP is able to transfer large files at over 15Mb/s, without increasing the background loss rate seen by TCP flows. To show this, we ran four parallel PCP transfers simultaneously with the TCP transfer; the TCP transfer was unaffected by the PCP traffic. In other words, for this path, there is a significant background loss rate, limiting TCP performance, despite the fact that the congested link has room for substantial additional bandwidth.

While some researchers have suggested specific modifications to TCP's additive increase rule to improve its performance for high bandwidth paths, we believe these changes would have made little difference for our tests, as most of our paths have moderate bandwidth and most of our tests use moderate transfer sizes. Quantitatively evaluating these alternatives against PCP is future work.

4.2 Simulation Results

We next use simulation to examine PCP's behavior in more detail. We chose not to use ns-2 for our simulations as it tends to be slow and scales poorly with the number of nodes or flows in the system. Using our own simulator also enabled us to reuse the same code for PCP and TCP that we ran on RON. Recall that we validated our TCP implementation against the TCP in the RON kernel.

For comparison, we also implemented centralized fair queueing [14] in our simulator. This mechanism achieves near-perfect isolation and fairness by scheduling packets from active flows in a bit-sliced round robin fashion. Given such a centralized router mechanism, it is possible for endpoints to infer their fair share by sending a pair of back to back packets, and observing their separation upon reception [35]. We model a fair-queueing system where the router also assists the endpoint in determining the optimal sending rate in the following manner. The endpoint transmits a control packet every RTT, and the router tags this packet with the flow's fair bandwidth allocation and the current queue for the flow. The endpoint simply sends at its fair share rate while compensating for queue buildups resulting from dynamic changes in the number of flows. While the resulting design might be difficult to realize in practice as it requires the router to communicate multiple bits of congestion information to the endpoint, it is intended as a bound on what is possible.

4.2.1 Impact of Varying Offered Load

We begin by evaluating the effect of varying load on the performance of our proposed algorithm and its alternatives. We consider a simple topology with a single bottleneck shared by many competing flows. The bottleneck bandwidth is 40 Mb/s and is shared by a hundred source-destination pairs. We model the bottleneck router as a FIFO drop-tail router for TCP and PCP flows. The buffering at the bottleneck is set to the bandwidth delay product. The round trip times for the source-destination pairs are uniformly distributed from 15 ms to 35 ms. We simulate fixed-length flows of 200 packets of size 1250

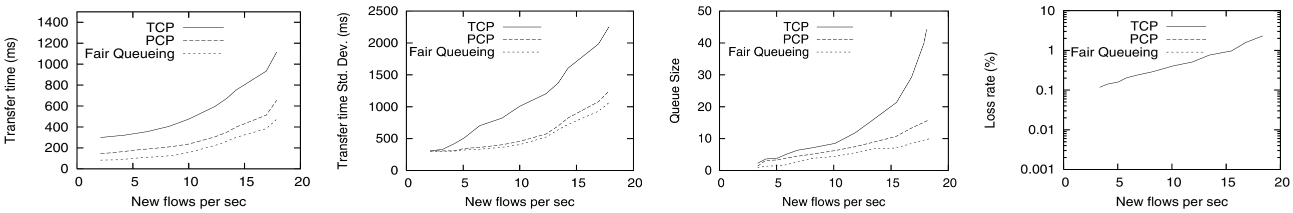

Figure 6: Performance when flow lengths and inter-arrival times are Pareto distributed. Mean flow length is 250 KB, bottleneck bandwidth is 40 Mb/s, and average RTT is 25 ms. Note that the standard deviation plot is depicted on a different scale due to the higher variances caused by Pareto-distributed flow lengths. Fair queueing and PCP suffer no packet loss.

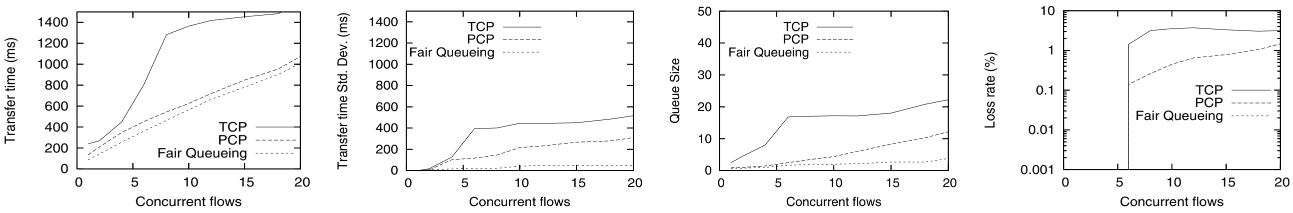

Figure 7: Performance with synchronized start of flows. Flow length is 250 KB, bottleneck bandwidth is 40 Mb/s, and average RTT is 25 ms. Fair queueing suffers no packet loss.

bytes each (resulting in an overall flow size of 250 KB). We vary the arrival rate of new flows to control the offered load to the system. The flow arrivals are randomized, using a Poisson arrival process, to avoid synchronization artifacts and to simulate varying load conditions, but the mean arrival rate is fixed based on the desired level of load.

The simulation parameters for this and the following experiments were chosen to be illustrative; they are not intended to be representative. For instance, instead of using a mixture of flow sizes as with real workloads, we typically use fixed size flows so that we can compare mean response times.

Figure 4 presents the variation in key performance characteristics as we increase the offered load from 15% to about 90% of bottleneck capacity. Since each flow offers 250 KB or 2 Mb of load, three new flows per second implies an offered load of 6 Mb/s or 15% of bottleneck capacity. We measure the response time to complete each transfer, including the cost of connection establishment. We also report the variability in response time (fairness), the queue size at the bottleneck, and the average loss rate at the bottleneck.

The results show that PCP has better response time than TCP and exhibits smaller variations in response time. PCP's response time is close to that of fair queueing. For low load conditions, PCP adds about two round-trip delays to fair queueing as it probes for available bandwidth before ramping up to the sending rate. At high load, PCP continues to perform well, ensuring that the bottleneck is kept busy; PCP keeps queues small while holding down average response time. Across the spectrum of load conditions, PCP keeps queue sizes lower than TCP, and has no packet loss even at high loads. Even at moderate loads, packet losses can dramatically penalize specific TCP flows [12]; PCP avoids this effect.

4.2.2 TCP and PCP over Fair Queueing Routers

We next study the performance of TCP and PCP flows when they are transmitted through routers that implement fair queueing. Our goal is to show that PCP is compatible with and benefits from general-purpose network enforcement. Others have argued for TCP-specific rules for penalizing misbehaving endpoints [20], but we argue that this unnecessarily constrains the choice of end host algorithm.

In theory, fair queueing is neutral to the choice of endpoint congestion control algorithm. An endpoint that sends faster than its fair share will build up queues and cause packet loss, but only to its own packets. However, the inefficiency imposed by TCP slow start has no benefit if the bottleneck resource is fair queued, but the endpoint has no way in general of knowing how the network is being managed.

We use the same simulation parameters as those used in the previous experiment, but substitute the FIFO drop-tail router with a fair-queued router. Figure 5 shows that PCP benefits from fair queueing, with lower response time variance even at high loads. Bandwidth probes initiated by arriving PCP flows are allowed to succeed due to the isolation of flows by the fair-queued router. TCP, on the other hand, performs slightly worse with fair queueing. When a TCP flow exceeds its fair share of router

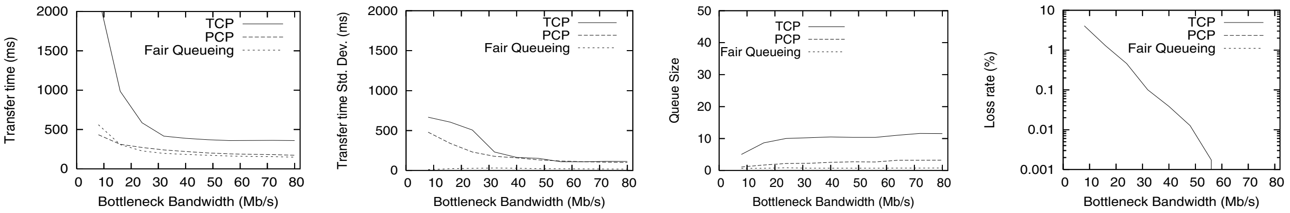

Figure 8: Effect of varying bottleneck bandwidth. Average RTT is 25 ms, flow lengths are 250 KB, and interarrival times are set to operate the system at 60% load. Fair queueing and PCP suffer no packet loss.

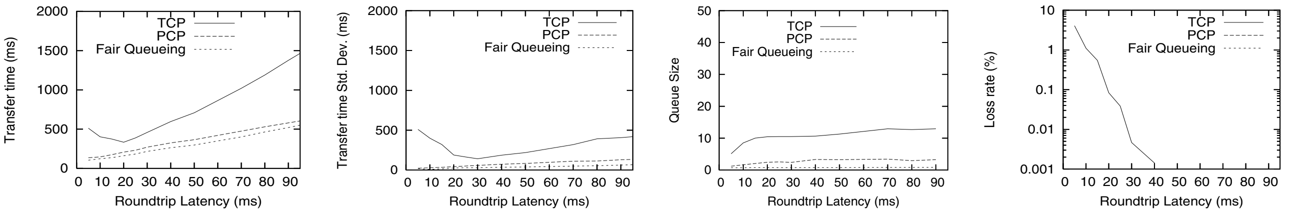

Figure 9: Effect of varying round-trip time. Bottleneck bandwidth is 40 Mb/s, flow lengths are 250 KB, and interarrival times are set to operate the system at 60% load. Fair queueing and PCP suffer no packet loss.

buffer space under loaded conditions, it suffers multiple losses, causing its performance to drop severely; in contrast, a drop-tail router spreads the packet losses more smoothly across all of its flows.

4.2.3 Bursty Traffic Patterns

We now evaluate the performance of PCP under bursty settings, where the history information might potentially yield incorrect predictions. First, we repeat the experiment from the previous section using a mixture of flow lengths instead of fixed length flows. We retain the mean length of flow to be 250 KB, but vary flow lengths according to a Pareto distribution with shape 1.25. We also set inter-arrival times to follow a Pareto distribution with shape 1.2. These parameters model self-similar traffic observed in real traces. Figure 6 plots the results. PCP provides performance close to router-based control in this case.

We next evaluate the different protocols when a number of concurrent flows are repeatedly started together resulting in synchronized transfers. In addition, we artificially rig PCP's history mechanism to always provide the estimate that all of the bottleneck bandwidth is available to each flow. By initiating multiple flows simultaneously with this estimate, we intend to evaluate PCP's performance when the history mechanism provides grossly inaccurate estimates. Figure 7 depicts the results as we vary the number of synchronized flows, each transferring 250 KB through our base configuration comprising of a 40 Mb/s bottleneck and 25 ms average RTT. Since the flow start times are synchronized, the TCP flows enter slow-start phase together. When the number of concurrent TCP flows is six or more, their combined ramp-up during slow-start results in filling up the router queue and causes a large number of packet losses. PCP also suffers from packet loss due to inaccurate history information, most of which occurs during the first RTT after transitioning to a high sending rate. But the sending rate is throttled in the subsequent round-trip as the PCP sender performs rate compensation in response to increased inter-packet gaps and one-way delays.

4.2.4 Impact of Varying Simulation Parameters

We then perform sensitivity analysis to study the effect of varying the parameters of the topology. Figure 8 presents the various metrics as the bottleneck bandwidth is varied. The rate of flow arrivals is set such that the offered load to the system is 60% of the bottleneck capacity for the various runs of the experiment. At lower capacities, TCP's slow-start phase overruns the available bandwidth resources, causing packet loss storms, resulting in substantial back-off and increased transfer times. TCP's transfer time performance levels out with increasing bandwidth, but never approaches the performance of PCP due to the $O(\log n)$ overhead associated with the startup phase.

Figure 9 illustrates the performance of various flows through our base configuration of a 40 Mb/s bottleneck router as we vary the round-trip latency of the flows. We again consider fixed-size flows of length 250 KB, and we also fix the offered load at 60% (twelve new flows per second for this configuration). The average round-trip latency is varied from 5ms to 100ms, and the buffer space is set to the corresponding bandwidth-delay product for each run. At small RTTs, TCP flows tend to blow out the small router queues rather quickly, while at high RTTs, the $O(\log n)$ slow-start overhead translates to much higher transfer times. PCP flows track the performance of fair queueing under all RTT conditions.

We also study performance as we vary the mean flow length. Figure 10 graphs the various performance metrics as we vary the flow size and correspondingly vary the arrival rate in order to fix the offered load at 60%. As we study the performance of TCP flows, we observe a tradeoff between two competing phenomena. As we increase the flow lengths, the initial slow-start overhead is amortized over a larger transfer. The resulting efficiency is however annulled by increased loss rates as there are a sufficient number of packets per flow for TCP to overrun buffer resources during the slow-start phase.

4.2.5 Impact of Transmission Loss

Finally, we evaluate the impact of transmission loss on the response time for the different protocols. We consider transmissions with an average RTT of 25 ms and subject the packets to a constant loss rate, independent of the load in the system. Figure 11 graphs the results. The response time for TCP blows up with increased loss rates, since TCP interprets losses as signals of congestion. PCP and fair queueing can tolerate losses without suffering a substantial increase in response times. In PCP, when a loss is detected, either through a timeout or by the presence of acknowledgments for subsequent messages, the packet is scheduled for retransmission for the next available time slot based on the current paced transmission rate.

5 Related Work

As we have noted, many of the elements of PCP have been proposed elsewhere; our principal contribution is to assemble these ideas into a system that can emulate the efficiency of network-based congestion control.

Our work on PCP is inspired in many ways by Ethernet arbitration [8]. PCP, like Ethernet, is designed to perform well in the common case of low load, with high load stability and fairness an important but secondary concern. Ethernet's lack of stability and fairness in certain extreme cases yielded much followup work within the academic community, but Ethernet's common case performance and simplicity was sufficient for it to be wildly successful in practice.

Our work also closely parallels the effort to define algorithms for endpoint admission control [11, 23, 33, 17, 7]. In these systems, endpoints probe the network to determine if a fixed-rate real-time connection can be admitted into the system with reasonable QoS guarantees and without disrupting previously admitted connections. As with our work, this research demonstrated that endpoints can effectively emulate centralized resource management. Nevertheless, there are significant differences with our work. First, real-time connections have fixed bandwidth demands and are relatively long-running; hence, probes were run only at connection setup, and it was not necessary to make them particularly efficient. For example, Breslau et al. suggest that probes should use TCP-like slow start to determine if there is sufficient capacity for the connection [11]. Our system is designed to allow probes to be short and precise. With endpoint admission control, once the connection is started, no further adaptation is needed; by contrast, dynamic adaptation is clearly required for efficient and fair congestion control.

Another major area of related work is the various efforts to short-circuit TCP's slow start delay for moderate-sized connections. Typically, these systems use some form of rate pacing for the initial (large) window, but revert to TCP-like behavior for steady state. This allows these systems to be mostly backwardly compatible with existing TCP implementations. As we have argued, however, determining the available bandwidth along a network path is easiest when network traffic is designed to be smooth and well-conditioned. For example, TCP Swift Start [43] uses an initial burst of four packets to measure the physical (not available) capacity of the network path. The hosts then set their initial window to be a fixed fraction (e.g. 1/8th) of the physical capacity. If the bottleneck has significant unused bandwidth, this works great, but it can theoretically create persistent congestion if the rate of arriving flows is greater than the fixed fraction can support. TCP Fast Start [42] and the MIT Congestion Manager [6] use history to guide the selection of the initial congestion window and other TCP parameters; however, their approach only works for nearly simultaneous connections to the same destination.

Similarly, several previous efforts have proposed using delay information, rather than packet loss, to guide congestion control. An early example of this was the packet pair algorithm, using the delay spread from back to back packets to measure the bottleneck bandwidth through a fair-queued router [35]. Packet pair does not perform well with FIFO queues, however, as all endpoints would send at the maximum capacity of the link. Two more recent examples are TCP Vegas [10] and FastTCP [31]. The motivation in each case was to improve steady state TCP performance. As we have argued, many TCP transfers never reach steady state on today's networks.

Finally, we use many individual pieces of technology developed in other contexts. XCP was the first to show that separate mechanisms could be used to provide efficiency and eventual fairness in a congestion control algorithm [32]. In XCP, routers allocate resources to flows without keeping per-flow state. If there is idle capacity, flow rates are rapidly increased without regard to fairness; eventual fairness is provided as a background additive increase/multiplicative decrease process applied

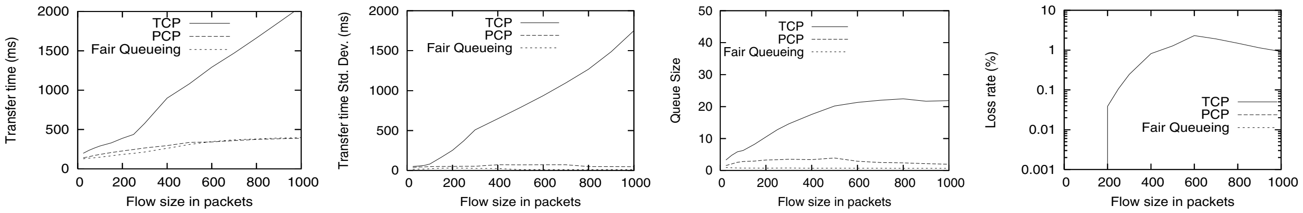

Figure 10: Effect of varying flow size. Bottleneck bandwidth is 40 Mb/s, average RTT is 25 ms, and interarrival times are set to operate the system at 60% load. Fair queueing and PCP suffer no packet loss.

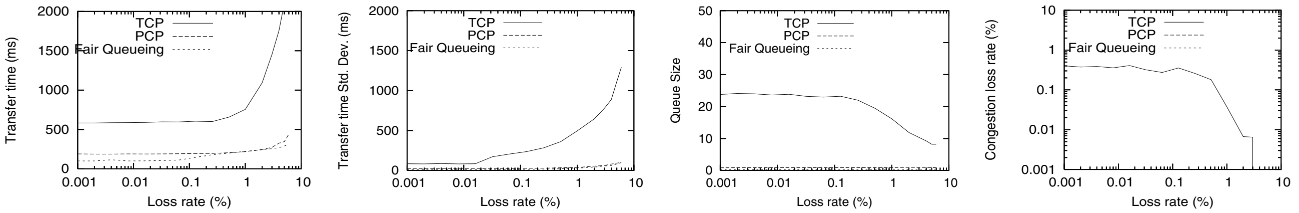

Figure 11: Performance over a lossy channel. Bottleneck bandwidth is 40 Mb/s, average RTT is 25 ms, flow lengths are 250 KB, and interarrival times are set to operate the system at 60% load. Fair queueing and PCP suffer no congestion packet loss.

to all flows. We directly leverage their approach in our work; in fact, we initially started our effort to investigate whether we could achieve the efficiency of XCP without router support. Similarly, Jain and Dovrolis show how to efficiently measure the available bandwidth (capacity less utilization) of an Internet path, by successively probing the path with faster and faster rates until measured delays start increasing [29]. We use a version of their technique, with two principal differences. Their motivation was to measure Internet paths; ours is congestion control. Thus, we carefully meter our probe rates, to ensure that they usually succeed; Jain and Dovrolis attempt to drive the network to the point where the probe starts causing congestion. Their work is also complicated by the fact that TCP traffic is particularly bursty at short time-scales, requiring much longer measurements than in our system. By ensuring that all traffic is paced, we can use shorter and more precise probes. Finally, we note that rate pacing has been proposed as a way to speed TCP startup [42, 43, 28]; if the TCP congestion control variables can be measured or predicted, pacing can provide a way to smooth the initial flight of traffic. We build on this work by using rate pacing throughout the lifetime of a connection, to provide high resource utilization with low delay variance. Earlier work has shown that this form of complete rate pacing interacts poorly with TCP dynamics, by delaying the onset of congestion [1]. In our system, however, the fact that competing flows use rate pacing allows an endpoint to quickly and accurately find if there is available capacity, avoiding the need to drive the resource to overload to determine its resource limits.

6 Conclusion

We have presented the design, implementation and evaluation of PCP, a novel architecture for distributed congestion control with minimal router support. By using short, paced, high-rate bursts, PCP is able to quickly converge to the desired bandwidth in the common case of lightly loaded links with good history. Although our results are somewhat preliminary, we have demonstrated that our approach can significantly outperform TCP, and approach optimal, for response time, loss rate, queue occupancy, and fairness. We believe PCP's combination of techniques shows great promise as a distributed resource allocation mechanism for modern networks.

Acknowledgments

We thank our shepherd Bryan Lyles and other NSDI reviewers for their comments on this paper. We also thank Dave Andersen for providing us access to the wide-area RON testbed.

References

[1] A. Aggarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. In *Proc. IEEE INFOCOM 2000*, Mar. 2000.

[2] A. Akella, S. Seshan, and A. Shaikh. An Empirical Evaluation of Wide Area Internet Bottlenecks. In *Proc. Internet Measurement Conference*, pages 101–114, Miami Beach, FL, Oct. 2003.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th SOSP*, 2001.

[4] The ATM Forum Traffic Management Specification Version 4.0, 1996.

[5] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A Comparison of Mechanisms for Improving TCP Performance

over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[6] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. ACM SIGCOMM '99*, pages 175–187, Boston, MA, Aug. 1999.

[7] G. Bianchi, A. Capone, and C. Petrioli. Throughput analysis of end-to-end measurement-based admission control in IP. In *Proc. IEEE INFOCOM 2000*, Mar. 2000.

[8] D. Boggs, J. Mogul, and C. Kent. Measured Capacity of an Ethernet: Myths and Reality. In *Proc. ACM SIGCOMM '88*, Stanford, CA, Aug. 1988.

[9] B. Braden and alia. Recommendations on Queue Management and Congestion Avoidance in the Internet. IETF RFC-2309, Apr. 1998.

[10] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. In *IEEE Journal on Selected Areas in Communications*, volume 13(8), pages 1465–1480, 1995.

[11] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang. Endpoint Admission Control: Architectural Issues and Performance. In *Proc. ACM SIGCOMM 2000*, Stockholm, Sweden, Aug. 2000.

[12] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proc. IEEE INFOCOM 2000*, Mar. 2000.

[13] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM '92*, pages 14–26, 1992.

[14] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair-Queueing Algorithm. In *Proc. ACM SIGCOMM '89*, 1989.

[15] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor Sharing Flows in the Internet. In *Proc. of IWQoS*, 2005.

[16] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36(1), 2006.

[17] V. Elek, G. Karlsson, and R. Ronngren. Admission control based on end-to-end measurements. In *Proc. of IEEE INFOCOM*, 2000.

[18] S. Floyd. TCP and Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review*, 24(5):10–23, Oct. 1994.

[19] S. Floyd. HighSpeed TCP for Large Congestion Windows. IETF RFC-3649, Dec. 2003.

[20] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, Aug. 1999.

[21] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[22] C. Fraleigh and alia. Packet-level Traffic Measurement from the Sprint IP Backbone. *IEEE Network Magazine*, Nov. 2003.

[23] R. Gibbens and F. Kelly. Distributed connection acceptance control for a connectionless network. In *Proc. 16th International Teletraffic Congress*, Edinburgh, UK, June 1999.

[24] R. Gibbens and F. Kelly. On packet marking at priority queues. *IEEE Transactions on Automatic Control*, 47:1016–1020, 2002.

[25] J. Gray. Distributed Computing Economics. Technical Report MSR-TR-2003-24, Microsoft Research, Mar. 2003.

[26] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. IETF RFC-1323, May 1992.

[27] V. Jacobson and M. Karels. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM '88*, 1988.

[28] A. Jain and S. Floyd. Quick Start for TCP and IP. IETF Draft, Feb. 2005.

[29] M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proc. ACM SIGCOMM 2002*, Pittsburgh, PA, Aug. 2002.

[30] C. Jin, D. Wei, and S. Low. The case for delay-based congestion control. In *Proc. IEEE Computer Communication Workshop*, Laguna Beach, CA, Oct. 2003.

[31] C. Jin, D. Wei, and S. Low. Fast TCP: Motivation, Architecture, Algorithms, Performance. In *Proc. IEEE INFOCOM 2004*, 2004.

[32] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proc. ACM SIGCOMM 2002*, Pittsburgh, PA, Aug. 2002.

[33] F. Kelly, P. Key, and S. Zachary. Distributed Admission Control. In *IEEE Journal on Selected Areas in Communications*, 2000.

[34] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *Computer Communication Review*, 2003.

[35] S. Keshav. A Control-Theoretic Approach to Flow Control. In *Proc. ACM SIGCOMM '91*, Aug. 1991.

[36] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level internet path diagnosis. In *Proc. of 19th SOSP*, 2003.

[37] L. Massoulie and J. Roberts. Arguments in favour of admission control for tcp flows. In *Proc. of ITC 16*, 1999.

[38] A. Medina, M. Allman, and S. Floyd. Measuring the Evolution of Transport Protocols in the Internet. http://www.icir.org/tbit/TCPevolution-Dec2004.pdf, Dec. 2004.

[39] R. Morris. TCP Behavior with Many Flows. In *Proc. of ICNP*, 1997.

[40] V. Oberle and U. Walter. Micro-second precision timer support for the Linux kernel. Technical report, IBM Linux Challenge, Nov. 2001.

[41] A. Odlyzko. Data Networks are Lightly Utilized, and will Stay that Way. *Review of Network Economics*, 2(3), Sept. 2003.

[42] V. Padmanabhan and R. Katz. TCP Fast Start: A Technique for Speeding Up Web Transfers. In *Proc. of Globecom Internet Mini-Conf.*, 1998.

[43] C. Partridge, D. Rockwell, M. Allman, R. Krishnan, and J. Sterbenz. A Swifter Start of TCP. Technical Report 8339, BBN Tech., 2002.

[44] K. K. Ramakrishnan and R. Jain. Congestion Avoidance in Computer Networks. Technical Report TR-510, DEC, Aug. 1987.

[45] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proc. IEEE INFOCOM 1999*, pages 1337–1345, 1999.

[46] L. Roberts. Performance of Explicit Rate Flow Control in ATM Networks. In *Proc. COMPCON Spring '96*, 1996.

[47] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), Nov. 1984.

[48] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An Analysis of Internet Content Delivery Systems. In *Proc. 5th OSDI*, 2002.

[49] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *ACM SIGCOMM Computer Communication Review*, 29(5):71–78, Oct. 1999.