Processor Sharing Flows in the Internet

Nandita Dukkipati[1], Masayoshi Kobayashi[2], Rui Zhang-Shen[1], and Nick McKeown[1]

[1] Computer Systems Laboratory,
Stanford University,
Stanford, CA 94305-9030, USA
{nanditad, rzhang, nickm}@stanford.edu
[2] System Platforms Research Laboratories,
NEC Corporation, Japan
m-kobayashi@eo.jp.nec.com

Abstract. Most congestion control algorithms try to emulate processor sharing (PS) by giving each competing flow an equal share of a bottleneck link. This approach leads to fairness, and prevents long flows from hogging resources. For example, if a set of flows with the same round trip time share a bottleneck link, TCP's congestion control mechanism tries to achieve PS; so do most of the proposed alternatives, such as eXplicit Control Protocol (XCP). But although they emulate PS well in a static scenario when all flows are long-lived, they do not come close to PS when new flows arrive randomly and have a finite amount of data to send, as is the case in today's Internet. Typically, flows take an order of magnitude longer to complete with TCP or XCP than with PS, suggesting large room for improvement. And so in this paper, we explore how a new congestion control algorithm — Rate Control Protocol (RCP) — comes much closer to emulating PS over a broad range of operating conditions. In RCP, a router assigns a single rate to all flows that pass through it. The router does not keep flow-state, and does no per-packet calculations. Yet we are able to show that under a wide range of traffic characteristics and network conditions, RCP's performance is very close to ideal processor sharing.

1 Introduction

Congestion control algorithms try to share congested links efficiently and fairly among flows. In the absence of information such as the size, round trip time (RTT) and path of each flow, it is natural to share a congested link equally among all flows. In fact, if the routers in the Internet had unlimited buffering, and if it was simple to emulate Processor Sharing (PS), then the solution to the congestion problem would be simple: Let a source send data at maximum rate, and use a PS scheduler to share link bandwidth equally among flows.

But routers have limited buffers, and per-flow scheduling is non-trivial. And so congestion control algorithms send feedback to the source to limit the amount of traffic admitted into the network, allowing simple FIFO queuing in the routers. Most notably, TCP's congestion control mechanism provides feedback by dropping packets (or through explicit congestion notification); and is quite successful at emulating processor sharing in a static scenario when a fixed number of flows have an infinite amount of

H. de Meer and N. Bhatti (Eds.): IWQoS 2005, LNCS 3552, pp. 267–281, 2005.
© IFIP International Federation for Information Processing 2005

data to send[1]. But in practice flows arrive randomly, and transfer a finite amount of data. Simple experiments with *ns-2* indicate that with typical Internet flow sizes, TCP does not come close to emulating processor sharing. For example, Figure 1 compares TCP (as well as XCP, which we'll discuss shortly) with ideal PS, in case of Poisson flow arrivals with pareto distributed flow sizes. The left plot compares the mean duration of flows (how long they take to complete) as a function of flow size. The values for PS are computed analytically [7] [2] and show that flows would complete an order of magnitude faster than for TCP.

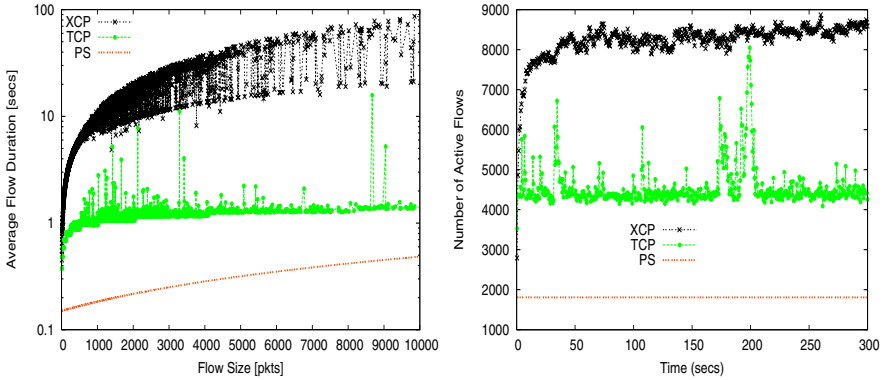

Fig. 1. The plot on left shows the average flow duration versus flow size under TCP and XCP from a simulation with Poisson flow arrivals, flow sizes are pareto distributed with mean = 30 pkts (1000 byte/pkt) and shape = 1.4, link-capacity = 2.4 Gbps, RTT = 100 ms, offered load = 0.9. The plot on right shows the the number of active flows versus time. In both plots the PS values are computed from analytical expressions

There are several reasons for the long duration of flows with TCP. First, it takes "slow-start" several round trip times to find the fair-share rate. In many cases, the flow has finished before TCP has found the correct rate. Second, once a flow has reached the "congestion-avoidance" mode, TCP adapts slowly because of additive increase. While this was a deliberate choice to help stabilize TCP, it has the effect of increasing flow duration. We'll see later that we can design stable congestion control algorithms that don't require additive increase. A third reason TCP flows last so long is because of buffer occupancy. TCP deliberately fills the buffer at the bottleneck, so as to obtain feedback when packets are dropped. Extra buffers mean extra delay, which add to the duration of a flow.

Our plots also show eXplicit Control Protocol (XCP) [2]. XCP is designed to work well in networks with large bandwidth-delay products. The routers provide feedback, in terms of incremental window changes, to the sources over multiple round trip times,

[1] We assume here that all flows have the same RTT. TCP approximately shares bandwidth as $\frac{K}{RTT\sqrt{p}}$ where p is loss probability and K is a constant [1].

[2] Flow duration in PS = RTT + $\frac{L}{C(1-\rho)}$, L is flow length, C is link capacity, ρ is offered load.

which works well when all flows are long-lived. But as our plots show, in a dynamic environment XCP can increase the duration of each flow even further relative to ideal PS, and so there are more flows in progress at any instant.

The goal of our work is to identify a simple and practical congestion control algorithm that emulates processor sharing irrespective of traffic characteristics and network conditions. Our approach is very different from TCP and XCP. Instead of incremental window changes in every round trip time, we want to know if there is an explicit rate that the router can give to the flows so as to emulate processor sharing. Furthermore, we would like to achieve this without per-flow state, per-flow queues, or per-packet calculations at the routers.

2 Rate Control Protocol (RCP): An Algorithm to Achieve Processor Sharing

2.1 Picking the Flow Rate

We are going to address the following question:

Is there a rate that a router can give out to all flows, so as to emulate processor sharing? If the router has perfect information on the number of ongoing flows at time t, *and* there is no feedback delay between the congested link and the source, then the rate assignment algorithm would simply be:

$$R(t) = \frac{C}{N(t)}$$

where $R(t)$ is the rate given out to the flows by the router at time t [3], C is the link capacity and $N(t)$ is the number of ongoing flows at time t. But the router does not know $N(t)$ and it is complicated to keep track of. And even if it could, there is a feedback delay and so by the time $R(t)$ reached the source, $N(t)$ would have changed. So, we propose that the routers have an adaptive algorithm that updates the rate assigned to the flows, to approximate processor sharing in the presence of feedback delay, without any knowledge of the number of ongoing flows. RCP is a particular heuristic designed to approximate PS. It has three main characteristics that makes it simple and practical:

1. The flow rate is picked by the routers based on very little information (the current queue occupancy and the aggregate input traffic rate).
2. Each router assigns a *single* rate for all flows passing through it.
3. The router requires no per-flow state or per-packet calculations.

2.2 The Algorithm

The basic RCP algorithm operates as follows.

1. Every router maintains a single fair-share rate, $R(t)$, that it offers to all flows. It updates $R(t)$ approximately once per RTT.

[3] The sources are informed at what rate to transmit. We will shortly see how this is done.

2. Every packet header carries a rate field, R_p. When transmitted by the source, $R_p = \infty$ or the desired sending rate. When a router receives a packet, if $R(t)$ at the router is smaller than R_p, then $R_p \leftarrow R(t)$; otherwise it is unchanged. The destination copies R_p into the acknowledgment packets, so as to notify the source. The packet header also carries an RTT field, RTT_p, where RTT_p is the source's current estimate of the RTT for the flow. When a router receives a packet it uses RTT_p to update its moving average of the RTT of flows passing through it, d_0.
3. The source transmits at rate R_p, which corresponds to the smallest offered rate along the path.
4. Each router periodically updates its local $R(t)$ value according to Equation (1) below.

Intuitively, to emulate processor sharing the router should offer the same rate to every flow, try to fill the outgoing link with traffic, *and* keep the queue occupancy close to zero. We want the queue backlog to be close to zero since otherwise if there is always a backlog then at any instant, only those flows which have their packets in the queue get a bandwidth share, and the other flows do not. This does not happen in ideal PS where at any instant every ongoing flow will get it's fair share. The following rate update equation is based on this intuition:

$$R(t) = R(t - d_0) + \frac{[\alpha(C - y(t)) - \beta\frac{q(t)}{d_0}]}{\hat{N}(t)} \tag{1}$$

where d_0 is a moving average of the RTT measured across all packets, $R(t - d_0)$ is the last updated rate, C is the link capacity, $y(t)$ is the measured input traffic rate during the last update interval (d_0 in this case), $q(t)$ is the instantaneous queue size, $\hat{N}(t)$ is the router's estimate of the number of ongoing flows (i.e., number of flows actively sending traffic) at time t and α, β are parameters chosen for stability and performance.

The basic idea is: If there is spare capacity available (i.e., $C - y(t) > 0$), then share it equally among all flows. On the other hand, if $C - y(t) < 0$, then the link is oversubscribed and the flow rate is decreased evenly. Finally, we should decrease the flow rate when the queue builds up. The bandwidth needed to drain the queue within an RTT is $\frac{q(t)}{d_0}$. The expression $\alpha(C - y(t)) - \beta\frac{q(t)}{d_0}$ is the desired aggregate change in traffic in the next control interval, and dividing this expression by $\hat{N}(t)$ gives the change in traffic rate needed per flow.

RCP doesn't exactly use the equation above for two reasons. First, the router cannot directly measure the number of ongoing flows, $N(t)$, and so estimates it as $\hat{N}(t) = \frac{C}{R(t-d_0)}$. Second, we would like to make the update rate interval (i.e., how often $R(t)$ is updated) a user-defined parameter, τ. This is in case we want to drain a filling queue more quickly than once per RTT. The update interval is actually $T = \min(\tau, d_0)$ since we want it to be at least equal to average RTT, d_0. The desired aggregate change in traffic over one average RTT is $\alpha(C - y(t)) - \beta\frac{q(t)}{d_0}$, and to update the rate more often

than once per RTT, we scale this aggregate change by T/d_0. And, $\hat{N}(t) = C/R(t-T)$. Then the equation becomes:

$$R(t) = R(t-T)[1 + \frac{\frac{T}{d_0}(\alpha(C - y(t)) - \beta\frac{q(t)}{d_0})}{C}] \quad (2)$$

2.3 Understanding the RCP Algorithm

How good is the estimate $\hat{N} = C/R$? When the router updates the rate, it knows precisely the spare capacity and the queue size it needs to drain. So the accuracy of the algorithm depends on how well C/R estimates $N(t)$.

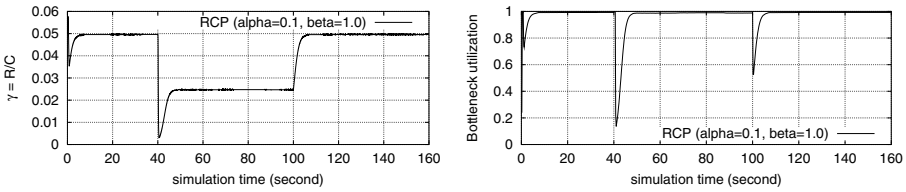

Fig. 2. The time evolution of RCP rate factor $\gamma(t) = R(t)/C$ and measured bottleneck utilization under long-lived flows. At $t = 0$, 20 flows start; at $t = 40$s, 20 more flows start; at $t = 100$s, 20 flows finish. In each case, $C/R(t)$ converges to $N(t)$

In the simplest scenario with only long-lived flows, $C/R(t)$ converges to the correct number of flows, N. An example is shown in Figure 2 where 20 flows start at time $t = 0$ and 20 more flows start at time 40, and 20 flows complete at time 100. In each case, $C/R(t)$ converges to $N(t)$. The values of α and β only affect the rate of convergence; we will examine the stability region for α and β shortly.

When flows are not long-lived, $C/R(t)$ can still be a good estimate of the number of active flows. In particular, when flows correspond to current Internet conditions (Poisson flow arrivals, pareto flow size distributions, and mean flow size $E[L]$ is close

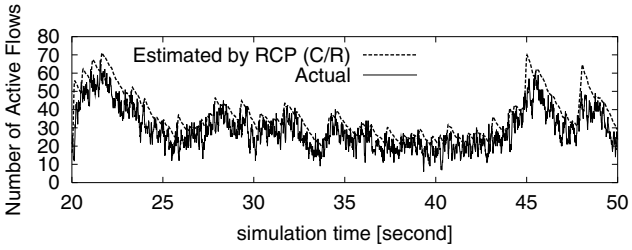

Fig. 3. Comparison of the number of measured active flows and the estimate (C/R). Bottleneck capacity, C = 10Mb/s, RTT = 50ms, flow arrival rate = 400 flows/sec, and flow sizes are pareto with mean = 25 pkts (1000 byte/pkt) and shape parameter is 1.2

to or greater than bandwidth×RTT), then $C/R(t)$ is a good estimate. It is a smoothing estimate of $N(t)$ since flows arrive and depart quickly and $N(t)$ changes rapidly. An example of this case is shown in Figure 3.

When $E[L] \ll$ bandwidth $\times$ RTT, most of the flows fit in the bandwidth-delay "pipe" and most do not have sufficient data to send for an entire round trip. In this case $C/R(t)$ represents an "effective" number of flows, $N_e(t) < N(t)$, where each flow has at least a round trip time worth of data to send. Underestimating the flows (and hence increasing the rate for each flow) is actually the right thing to do because when each flow has less than an RTT of data to send, giving exactly $C/N(t)$ to each flow means the pipe will never be filled.

Stability and Convergence: Stability of RCP depends on it's parameters α and β. We can think about RCP stability under the following two very different regimes:

1) Deterministic scenario of long-lived flows: In this scenario with N long-lived flows the equilibrium state of the RCP system is: R_e (equilibrium rate) $= C/N$ and q_e (equilibrium queue) $= 0$. We find that, if perturbed, the system will return to stability so long as α, β are within the stable region shown in Figure 4. There are two regions shown in the figure: a) The stable region obtained by Bode and Nyquist analysis of the linearized system. Details of the linear stability analysis are given in the technical report [3]. b) Stable region of the non-linear system. The real system is non-linear in nature, the most important one being the queue saturation at $q = 0$. In general while doing the stability analysis of congestion control protocols such as TCP, this non-linearity is ignored since the equilibrium point is away from it. The same is not true of the RCP system. Shown in Figure 4 is also the stable region obtained by simulations and phase portraits of this nonlinear system. Using tools from non-linear control theory, we obtained a precise characterization of the non-linear stable region and it matches well with our simulated region. The details of the non-linear analysis can be found at [4].

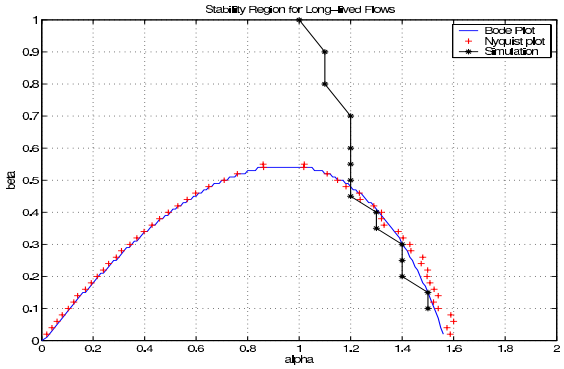

Fig. 4. Region enclosed by the solid curve is the stability region obtained via Bode and Nyquist analysis. The region to the left of the $'-*'$ line is the stability region of the non-linear system obtained from simulations and phase plane method

The two key points of the stability analysis are: First, the derived stability conditions for (α, β) guarantee global stability in the sense that irrespective of the initial conditions, (R_0, q_0), the system always converges to the equilibrium point; and second, we can make the system stable by choosing α and β *independent* of link RTT, capacity and number of flows. Although these results are proved to hold true incase of a single bottleneck link, our simulations in [3] indicate that they also hold true in a network scenario with multiple bottlenecks.

2) Stochastic scenario with random flow arrival times and flow sizes: In this case, convergence of $R(t)$ in the same sense as for long-lived flows is less meaningful because the input conditions are changing. Further, as discussed before we do not always want $R(t)$ to be equal to $C/N(t)$ exactly: If $N(t)$ is very large but each of the flows has very little traffic to send (less than a RTT) then we actually want to underestimate $N(t)$ and thereby give a higher rate to each flow, since if we give $C/N(t)$ exactly to each flow we will never fill up the link.

What would be more meaningful would be convergence in the stochastic sense like $E[N(t)]$ (mean number of flows) and $E[D(l)]$ (mean flow completion time for flow of length l) converge to finite equilibrium values. Proving such a result rigorously is a notoriously hard problem specially for non-linear delayed feedback systems such as RCP. The same is true for TCP, XCP and other algorithms. A large number of simulations indicate that under a variety of dynamic situations (like different flow arrival distributions, different flow size distributions, offered load, link capacities, round trip times...) RCP's performance in terms of $E[N(t)]$ and $E[D(l)]$ converges to that under ideal processor sharing for a wide range of $(\alpha, \beta) > 0$. These simulations are shown in section 3 and a more exhaustive set is in [3].

The convergence of RCP's performance measures $E[D(l)]$ (and $E[N(t)]$) to that of processor sharing is independent of the initial value of $R(t)$ chosen. The simulations support this. For any particular simulation we observe that $R(t)$ sweeps over the entire space (min-rate, link-capacity), depending on the conditions on the link. Any point could have been the starting point of the experiment. An example to illustrate this is shown in Figure 5. Notice that $R(t)$ takes a wide range of values depending on the input conditions. Starting with different initial values of $R(t)$ will give different sample

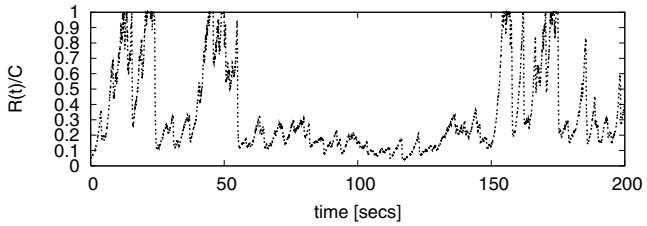

Fig. 5. The figure shows the normalized rate, $R(t)/C$ versus time for Poisson flow arrivals with pareto distributed flow sizes. Bottleneck capacity, C = 150 Mb/s, RTT = 100ms, offered load = 0.7, mean flow size = 30 pkts (1000 byte/pkt) and shape parameter is 1.2. Initial rate, $R(0) = 0.05C$. $R(t)$ sweeps over it's entire range depending on the input conditions

paths of the stochastic processes $N(t)$ and $D(l)$, but the key point is the underlying statistical properties $E[N(t)]$ and $E[D(l)]$ converge to that in processor sharing.

Given that the algorithm is stable for a wide range of $(\alpha, \beta) > 0$, we picked those values for the RCP system to maximize performance for a wide range of traffic and network conditions.

Round Trip Time Estimation: Every packet passing through the router carries the source's estimate of it's RTT. The router uses this to update the moving average, d_0, as follows:

$$d_0 = gain \times RTT_{packet} + (1 - gain) \times d_0^{last}$$

where gain $= 0.02$. The running average gives an estimate of the average RTT across all packets passing through the router. This skews the RTT estimate towards flows which have a larger number of packets. This is what is desired since the flows with a large number of packets will last many RTTs and will determine the stability of the control loop. The control loop stability depends less on the short flows which finish within one or just a few RTTs.

We find from our large number of simulations that RCP is robust to the RTT distribution of the flows. An example is shown in section 3.3 where flows with RTT ratios up to two orders of magnitude co-exist on a single link and RCP successfully emulates processor sharing.

Handling Packet Losses: RCP retransmits lost packets just like TCP. Losses were rare events for RCP in all our simulations, which is not surprising since RCP drives the queue towards empty. A queue only builds up because of the short term "mistakes" in rate estimation, resulting from the feedback delay and the small amount of information the algorithm is working with. Although the current form of RCP in Equation (2) does not explicitly account for losses, we note that it can easily do so by replacing $q(t)$ with $q(t)+$ (number of packet losses in interval T) – i.e. this would have been the queue we wanted to drain if we had enough buffering to accept the lost packets.

Comparison with XCP: Both XCP and RCP try to emulate processor sharing among flows, which is why their control equations are similar. The manner in which they converge to PS is quite different; the main difference between XCP and RCP is in the kind of feedback that flows receive. XCP gives a window increment or decrement over the current window size of the flow (which is small for all newly starting flows). At any time XCP flows could have different window sizes and RTTs and therefore different rates. XCP continuously tries to converge to the point where all flows have the fair-share rate, by slowly reducing the window sizes of the flows with rates greater than fair-share and increasing windows of the flows with rates less than fair-share (while avoiding over-subscription). New flows start with a small window, and the convergence could take several RTTs especially if there is little or no spare capacity. If the flows arrive as a Poisson process with heavy-tailed flow sizes, then most of the flows finish by the time they reach their fair share. In RCP, all flows (new and old) receive the same rate feedback which is their *equilibrium* rate. This helps flows finish quickly. We will see in Section 3 that this difference between RCP and XCP contributes to a big difference in their performance.

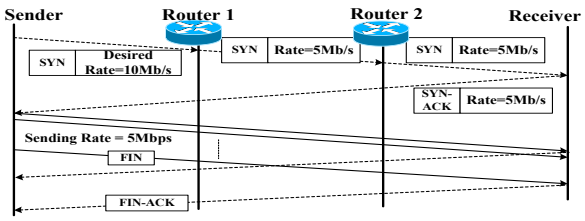

Fig. 6. The SYN message sent by the source indicates the rate at which it wants to send the flow (which could be infinite). As detailed in the last section, each router maintains a single rate, $R(t)$, that it assigns to all flows. As the message passes through the network, if the current rate $R(t)$ at a router is lower than the value in the SYN packet, the router overwrites it. When the SYN packet reaches its destination, it has the lowest rate corresponding to the most congested link along the path. This value is sent back to the source in the SYN-ACK message to set the starting rate. When the flows last longer than an RTT then they are periodically and explicitly told a new rate by the network. This rate is piggy-backed on the data and the ACK messages

XCP is computationally more complex than RCP since it gives different feedback values to each flow, and involves multiplications and additions for every packet. RCP maintains a single rate for all flows and involves no per-packet computation. [4]

2.4 RCP for the Internet

This is an outline of how RCP can be implemented in the Internet. We assume that – as with TCP – flows continue to have the connection set-up phase to establish state at both ends of the connection. This allows the initial rate to be calculated during the initial handshake by piggy-backing on the SYN and SYN-ACK messages. This is very important for short-lived flows, which could last less than one RTT. Current feedback-based algorithms do not work well for short-lived flows, yet most flows in the Internet are of this type [5]. An example of the RCP startup mechanism is illustrated in Figure 6.

3 RCP Performance

3.1 Simulation Setup

In this section we study RCP's performance using ns-2 [6] (Version 2.26) augmented with RCP end-host and queue modules.

We compare the performance of RCP with processor sharing, TCP and XCP. We are primarily interested in the average flow completion time (AFCT) [5]. Flow completion

[4] The router uses the RTT information in the packets to update its RTT estimate – our stability analysis and simulations indicate that it is sufficient for the router to have a "rough" estimate of the feedback delay, and so it can even just sample a few packets and update its estimate of RTT.

[5] We will use the term "flow" here to represent the packets corresponding to a particular application flow.

time (FCT) is defined as the time from when the sender sends a SYN packet until the receiver receives the last packet of the flow, i.e. FCT = 1 RTT for the connection set-up plus the duration of the data transfer. For elastic flows this is arguably the most important performance metric. We will use RTPD to abbreviate *round-trip propagation delay*. AFCT is the average of FCT over all flows for the simulation run. Note that $AFCT \geq 1.5RTPD + \frac{E[L]}{C}$. This is because (ignoring queuing delay) the minimum FCT for any flow of size L is: 1 RTPD for SYN/SYN-ACK and (1/2 RTPD + L/C) for the data transfer. The analytical expression for FCT of a flow of size L under processor sharing is [7]:

$$FCT_{PS} = 1.5\ RTPD + \frac{L}{C(1-\rho)} \tag{3}$$

where ρ is the offered load and C is the link capacity. We will use Equation (3) to compute the PS values for our simulation setups. As secondary measures, in [3] we are also interested in the link utilization, and the average number of ongoing or active flows – which in PS can be simply computed by Little's Law: $E[N] = \lambda \times FCT_{PS}$ where λ is the flow arrival rate.

We assume the usual rule-of-thumb that a router's queue equals the bandwidth-delay product, i.e., link capacity multiplied by maximum RTPD of flows passing through it. We also assume that packets are dropped from the tail of the queue. Our simulations are run until the performance measures converge. In all simulations so far, we have not seen any packet drops with RCP and XCP. There are packet drops with TCP.

Equation (2) is the rate update equation used in the RCP router. The RCP parameters are: Control period, $T = \min(10\text{ms}, d_0)$ and $\alpha = 0.1, \beta = 1.0$. For TCP, we used TCP Reno module in *ns-2* with an initial window size of two packets. The *ns-2* implementation of XCP (Version 1.1) is publicly available [8], and the parameters are set as in the paper [2].

All data packets are 1000 bytes and the control packets (SYN, SYN-ACK, FIN) are 40 bytes. Unless otherwise mentioned we will assume that flows arrive as a Poisson process with rate λ and flow sizes are pareto distributed [5, 9]. The offered load on a link is $\rho = \lambda E[L]/C$. In our simulations we vary the network and traffic parameters from one extreme to the other and observe how RCP, TCP and XCP compare with PS.

3.2 When Traffic Characteristics Vary

In this section our goal is to find out if RCP's performance is close to PS under different traffic characteristics. All simulations in this section are done with a single bottleneck link in the network.

Average Flow Completion Time vs. Flow Size: In this section we will observe the AFCT of RCP, XCP and TCP for an entire range of flow sizes in a particular simulation setup chosen to represent high bandwidth-delay product ($C \times \text{RTPD}$) environment. This is the scenario that often differentiates the performance of protocols.

• Setup: C = 2.4 Gbps, RTPD = 100 ms, ρ = 0.9, pareto distributed flow sizes

AFCT is plotted against flow size in the top two graphs of Figure 7. The AFCT of RCP is close to that of PS and it is always lower than that of XCP and TCP. For flows

up to 2000 pkts, TCP delay is 4 times higher than in RCP, and XCP delay is as much as 30 times higher for flows around 2000 pkts. Note the logscale of the y-axis.

With longer flows ($>$ 2000 pkts), the ratio of XCP and RCP delay still remains around 30, while TCP and RCP are similar. For any fixed simulation time, not only was RCP better for the flows that completed, but it also finished more flows (and more work) than TCP and XCP.

The third graph in Figure 7 shows the maximum delay for a given flow size. Note that in RCP the maximum delay experienced by the flows is also very close to the average PS delay. With all flow sizes, the maximum delay for RCP is smaller than for TCP and XCP. TCP delays have high variance, often ten times the mean.

The results above are representative of the large number of simulations we performed. Now let's see why these protocols have such different delays.

RCP vs. TCP: In figure, 7 the TCP delay for most flows follows the *Slow-start* curve. The delay in TCP slow-start for a flow of size L is $[log_2(L+1)+1/2] \times RTPD + L/C$ (excluding the queuing delay). With RCP the same flows get a jump-start because the routers set a higher initial rate close to what they would have gotten with PS. Hence their delay is close to PS. This is clear from the time evolution of a typical flow, as shown in Figure 8 (left plot).

Next, consider the TCP flows which deviate from the Slow-start curve. These flows experienced at least one packet drop in their lifetime and entered the additive increase, multiplicative decrease (AIMD) phase. Once a flow is in the AIMD phase, it is slow in

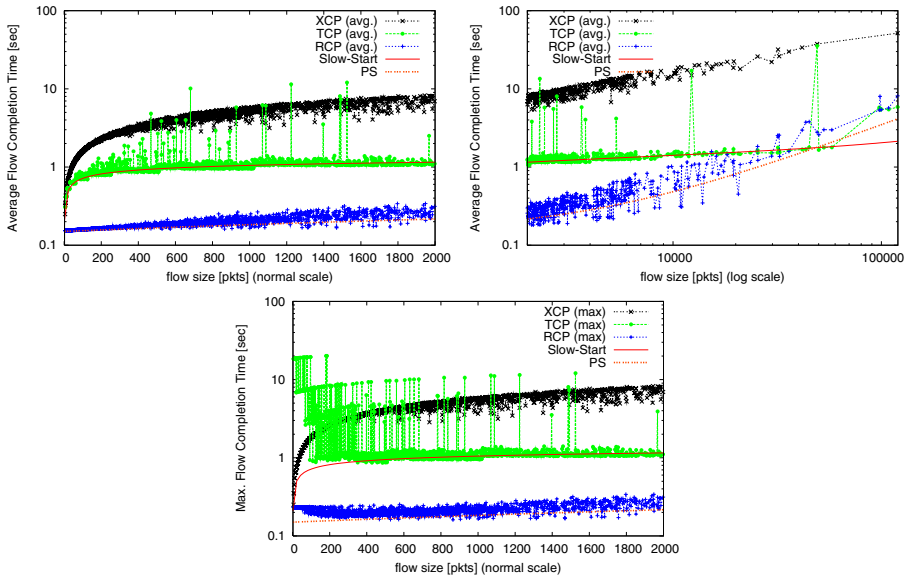

Fig. 7. AFCT for different flow sizes when $C = 2.4$ Gb/s, RTPD=0.1s, and $\rho = 0.9$. Flows are pareto distributed with $E[L] = 25$ pkts, shape = 1.2. The top left plot shows the AFCT for flow sizes 0 to 2000 pkts; the top right plot shows the AFCT for flow sizes 2000 to 10^5 pkts; the bottom plot shows the maximum flow completion time among all flows of the particular size

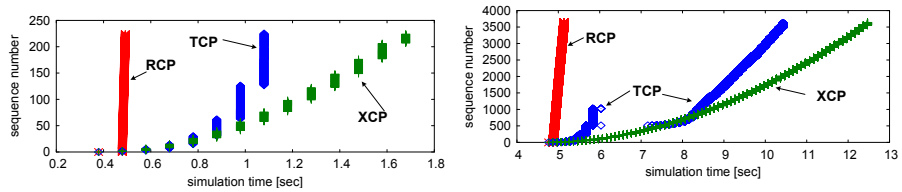

Fig. 8. Time evolution of the sequence numbers of two flows under TCP, XCP and RCP, chosen from the simulation set up of Figure 7. The flow size in the left plot is 230 pkts, and in the right plot is 3600 pkts

catching up with any spare capacity and therefore lasts longer than it needs to. RCP on the hand is quick to catch up with any spare capacity available and flows finish sooner. An example of the time evolution of a flow is shown in Figure 8 (right plot).

RCP vs. XCP: The time evolution of XCP for two sample flows is shown in Figure 8. XCP is slow in giving bandwidth to the flows, giving a small rate to newly starting flows. It gradually reduces the window sizes of existing flows and increases the window sizes of the new flows, making sure there is no bandwidth over-subscription. It takes multiple RTTs for most flows to reach their fair share rate (which is changing as new flows arrive). Many flows complete before they reach their fair share rate. In general, XCP stretches the flows over multiple RTPDs, to avoid over-subscribing the link, and so keep buffer occupancy low. On the other hand, RCP tries to give the equilibrium rate to every flow based on the information it has so far, at the expense of temporary bandwidth over-subscription.

When mean flow size increases: Figure 9 compares AFCT when mean flow size gets longer. Flow sizes are pareto distributed and the mean flow size is varied from 30 pkts (equals $\frac{1}{1000} \cdot C \cdot RTPD$) to 30,000 pkts (equals $C \cdot RTPD$). The left plot shows the AFCT averaged over flows with $< 7,000$ pkts and the right one is for flows $\geq 7,000$ pkts. [6]

There are two points to take away from the graph:

1. The AFCT of RCP is close to PS irrespective of the mean flow size
2. The performance of XCP and TCP is reversed as the mean flow size increases: when the mean flow size is small, XCP performs far worse than TCP (for flows with > 7000 pkts) and as the mean flow size gets larger, XCP's performance gets closer to PS while TCP deviates further from it – see right plot of Figure 9.

XCP vs. TCP: The reversal in performance of XCP and TCP is also clearly illustrated in Figure 10. The top left plot shows a snap shot of the AFCTs for $E[L] = 30$ pkts and the other two plots are for $E[L] = 30000$ pkts. In the bottom plot the AFCT

[6] We consider these two different ranges because, with a pareto distribution, there are many more short flows than long flows. Just taking the average AFCT over all flows is more representative of the short flows than the long flows.

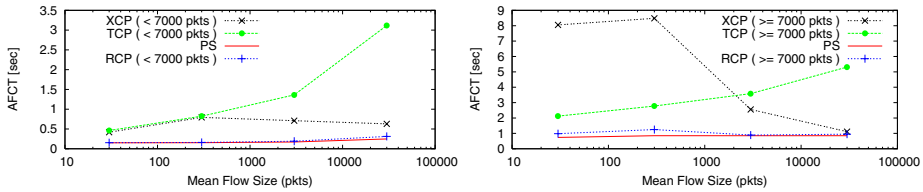

Fig. 9. Comparison of AFCT as the mean flow size increases. Flows are pareto distributed with shape 1.2 and the mean flow size varies as shown on x-axis. $C = 2.4$ Gb/s, RTPD $= 0.1$s and $\rho = 0.8$. The left plot shows the AFCT for flows with < 7000 pkts vs. mean flow size; the right plot shows the AFCT for larger flows (> 7000pkts) vs. mean flow size

of TCP flows is more than an order of magnitude higher than in PS – this is due to the well known problem with TCP in high bandwidth delay product environments i.e., long flows are unable to catch up with spare bandwidth quickly after experiencing a loss. XCP and RCP are both close to PS. On the other hand, for small flows, XCP's performance is worse than TCP's because XCP is conservative in giving bandwidth to flows, especially to newly starting flows. This unnecessarily prolongs flows and so the number of active/ongoing flows begins to grow. This in turn reduces the rate of new flows, and so on. Our many simulations showed that new flows in XCP start slower than with Slow Start in TCP.

We observe from a large number of simulations that the trends in flow completion times observed above for RCP, TCP and XCP hold true for other traffic characteristics

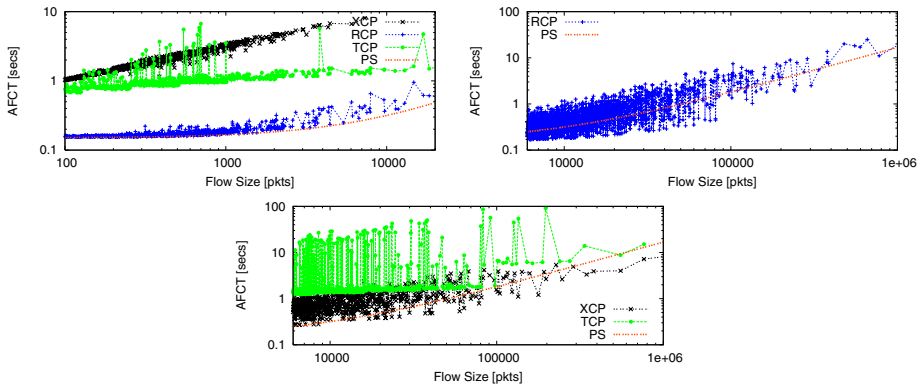

Fig. 10. Comparison of AFCT as the mean flow size increases. The simulation set up is the same as in Figure 9. The top left graph shows the AFCT vs. flow size when $E[L] = 30$; the top right (RCP) and bottom graph (TCP, XCP) show the AFCT vs. flow size when $E[L] = 30000$ pkts. RCP does close to PS irrespective of mean flow size. The performance of XCP and TCP are reversed with the increase in the mean flow size

such as different flow size distributions, as the offered load varies, and under different non-poisson flow arrival time distributions. Simulation results for these are in [3].

3.3 When Network Conditions Vary

In this section we explore how well the congestion control algorithms match PS under different network conditions. We show one particular scenario here when flows with widely different round trip times share a common bottleneck link. Simulations for other network conditions such as varying bottleneck link capacities, RTTs, increasing number of bottleneck links are shown in [3]. In each case, we find that RCP matches PS closely for a single bottleneck. In case of multiple bottlenecks, RCP achieves max-min fairness.

Flows with different round-trip times: All three congestion control schemes depend on feedback to adjust the window size and/or sending rate. If different flows have shorter round trip times, we do not want them to benefit at the expense of others.

To explore this effect we simulated flows that share a common bottleneck link, but with different RTPDs. The round-trip delay of the common bottleneck link is 0.01 s and its capacity is 640Mb/s. Arriving flows are classified into ten groups. Flows in the same group have the same end-to-end RTPD, and each group has an RTPD of 0.02, 0.04, ,.., 0.18, or 0.2s. All groups have the same flow arrival rate and total $\rho = 0.9$.

Figure 11 shows the AFCT for these different groups of flows. The x-axis is each group's RTPD. For each RTPD, RCP is close to PS, suggesting that RCP is not biased in favor of flows with shorter RTPD.

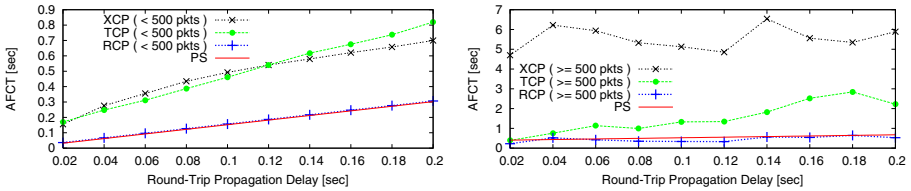

Fig. 11. Comparison of RCP, TCP and XCP when flows with different RTPDs coexist on a single bottleneck of $C = 0.64$ Gb/s. Flows arrive as a Poisson process with pareto distributed flow sizes, $E[L] = 25$ pkts, shape = 1.2. RTPD of flows vary from 0.02s to 0.2s. The left figure is the AFCT of flows with flow size $\leq$ 500 pkts and the right figure shows the AFCT for flows with size $>$ 500 pkts

4 Conclusion

TCP's congestion control mechanisms work well in a static network with only long-lived flows. With long lasting flows, small mistakes in control do not lead to a big drop in performance. This is no longer true in a dynamic environment with random flow arrivals and arbitrary amounts of data to send. We saw in this paper the unnecessary number of round trip times taken by the TCP slow-start and AIMD algorithm to find

the fair-share rate. Often, the flow has finished before the fair-share rate has been found. Unfortunately, making the network faster does not help, because the flow duration is dominated by the propagation delay. The same is true for XCP.

It is the premise of this paper that it is better to design congestion control algorithms to closely emulate processor sharing. This way the algorithm scales naturally with link capacities, RTTs and other network conditions. The performance is invariant of the flow size distribution – so it will not matter what mix of flows applications generate. Flows will complete sooner for a broad range of network and traffic conditions.

RCP is designed to be a practical way to emulate processor sharing, and appears to come very close to doing so under a broad range of conditions, and allows flow to complete much faster than with TCP or XCP.

References

1. S.B. Fredj, T. Bonald, A. Proutiere, G. Regnie, J.W. Roberts, "Statistical Bandwidth Sharing: A Study of Congestion at Flow Level," In *Proceedings of ACM Sigcomm 2001*, San Diego, August 2001.
2. D. Katabi, M. Handley, and C. Rohrs, "Internet Congestion Control for High Bandwidth-Delay Product Networks," In *Proceedings of ACM Sigcomm 2002*, Pittsburgh, August, 2002.
3. N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, "Processor Sharing Flows in the Internet," *http://yuba.stanford.edu/rcp/*, Stanford HPNG Technical Report TR04-HPNG-061604.
4. H. Balakrishnan, N. Dukkipati, N. McKeown and C. Tomlin, "Stability Analysis of Switched Hybrid Time-Delay Systems – Analysis of the Rate Control Protocol," *http://yuba.stanford.edu/rcp/*, Stanford University Department of Aero/Astro Technical Report.
5. M. E. Crovella and A. Bestavros, "Self Similarity in World Wide Web Traffic: Evidence and Possible Causes," In *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, December 1997.
6. The Network Simulator, http://www.isi.edu/nsnam/ns/
7. W. Wolff, "Stochastic Modeling and the Theory of Queues," PrenticeHall, 1989
8. *ns-2* code for Explicit Control Protocol, *http://ana.lcs.mit.edu/dina/XCP/*
9. V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," In *IEEE/ACM Transactions on Networking*, 3(3):226-44, June 1995.