Processor Sharing Flows in the Internet

Nandita Dukkipati, Nick McKeown
Computer Systems Laboratory
Stanford University
Stanford, CA 94304-9030, USA
{nanditad, nickm}@stanford.edu

***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 RTT 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.**

***Index Terms*— Processor Sharing, Congestion Control, Flow Completion Time, High Bandwidth-Delay Networks, Short Flows**

I. 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 queueing 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 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 with ideal PS (as well as XCP, which we'll discuss shortly). The top plot compares the mean duration of flows (how long they take to complete) as a function of flow size. The values for PS are derived analytically, and show that flows would complete an order of magnitude faster than for TCP.

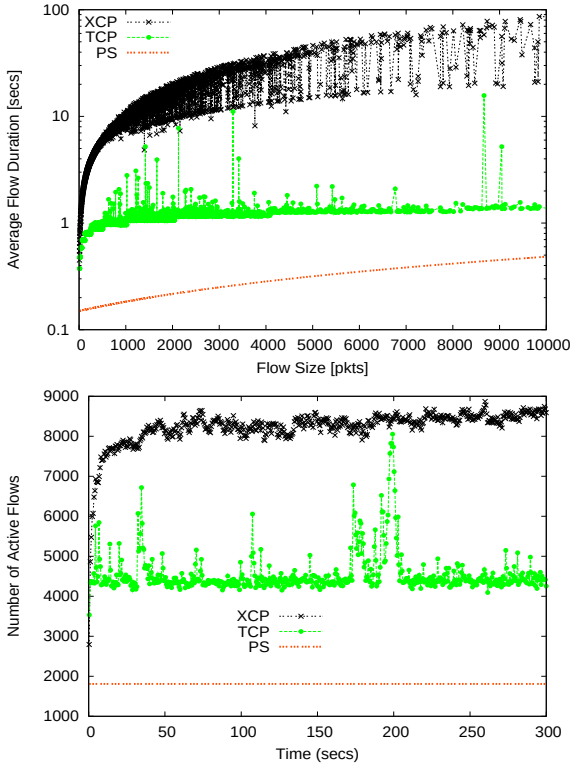

Fig. 1. The top plot shows the average flow duration verses flow size under TCP and XCP from a simulation with Poisson flow arrivals, flow sizes are Pareto distributed with mean = 30 pkts (1000 byte/pkt) and shape = 1.4, link-capacity = 2.4 Gbps, Round Trip Time = 100 ms, offered load = 0.9. The bottom plot shows the the number of active flows verses 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

[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 [1] and K is a constant.

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, 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 flow size distributions 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.

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

A. Picking the Flow Rate

We're 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 [2]and C is the link capacity. 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]The sources are informed at what rate to transmit. We will shortly see how this is done.

B. 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.
2) Every packet header carries a rate field, R_p. When transmitted by the source, $R_p = \infty$. 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 don't. This does not happen in ideal PS where at any instant every ongoing flow will get it's fair share[3]. The following rate update equation is based on this intuition:

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

where d_0 is a moving average of the RTT measured across all flows, $R(t - d_0)$ 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 can't 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, T.[4] The desired aggregate

[3]Assuming a fluid model for simplicity.

[4]This is in case we want to drain a filling queue more quickly than once per RTT. The update interval is actually $\min(T, d_0)$ since we want it to be at least equal to RTT.

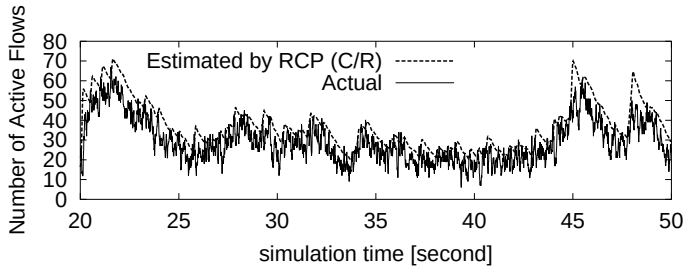

Fig. 2. Comparison of the number of measured active fbws and the estimate (C/R). Bottleneck capacity, C = 10Mb/s, RTT = 50ms, fbw arrival rate = 400 fbws/sec, and fbw sizes are Pareto with mean = 25 pkts (1000 byte/pkt) and shape parameter is 1.2.

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:

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

C. Understanding the RCP Algorithm

1) 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 $N(t)$.

In the simplest scenario with only long-lived flows, C/R converges to the correct number of flows, N. An example is shown in Figure 19 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 converges to $N(t)$. The values of α and β only affect the rate of convergence; we'll examine the stability region for α and β shortly.

When flows are not long-lived, C/R 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 to or greater than bandwidth×RTT), then C/R is a good estimate. It is a smoothing estimate since flows arrive and depart quickly and $N(t)$ changes rapidly. An example of this case is shown in Figure 2.

When $E[L] \ll$ bandwidth × 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.

But, what if some of the flows are bottlenecked elsewhere and cannot send in traffic at rate R. For example if N_1 flows are bottlenecked at a certain node and N_2 flows are bottlenecked elsewhere and are arriving at rate, R_2, less than their fair share i.e. $R_2 < C/(N_1 + N_2)$. In this case, the rate R in RCP will be such that $RN_1 + R_2N_2 = C$ i.e. $R = \frac{C - R_2N_2}{N_1}$. In other words, RCP achieves max-min fairness. An example is shown in Figure 3. From time, $t = 40$ to 100, $N_1 = 8$ flows (Group A) and $N_2 = 4$ flows (Group B) share the bottleneck of 400 Mbps (Link C). Group B flows are bottlenecked at Link B of 80 Mbps. As seen in the bottom figure, the flows achieve their max-min fair rates. In this case, C/R is an estimate of $N_1 + \theta N_2$ where $\theta = R_2/R$. This is exactly what is desired if we want max-min fairness.

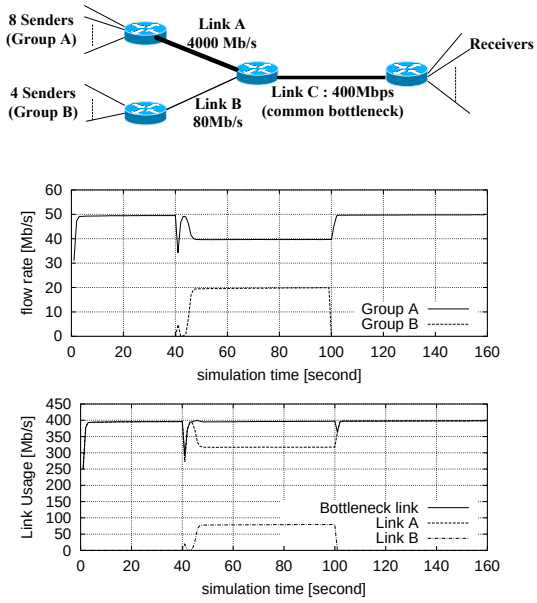

Fig. 3. Long fbws achieve max-min fair under RCP: 8 fbws (Group A) start at t = 0, and 4 fbws (Group B) join at t = 40 and leave at t = 100. Group A is bottlenecked at Link C and Group B is bottlenecked at Link B. The fbws achieve their max-min fair rates.

2) 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.

RCP's control is robust and stable even when packets are lost. If, for example, ACKs are lost, then rate feedback information is delayed. This is equivalent to increasing the feedback delay (or round trip time in the network). We will see in Section III that RCP's control is robust to increasing round trip delay.

3) 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

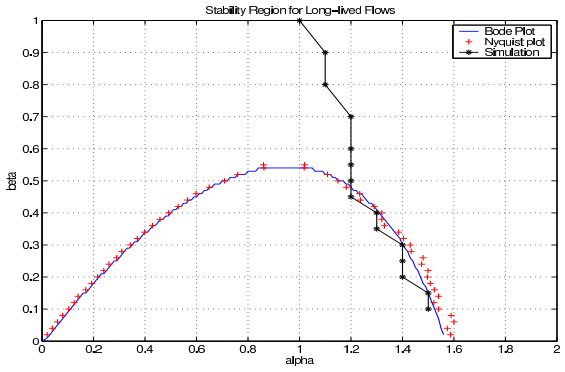

Fig. 4. Region enclosed by the solid curve is the stability region obtained via Bode and Nyquist analysis. The region to the left of the $'-*'$ line is the stability region of the non-linear system obtained from simulations and phase plane method.

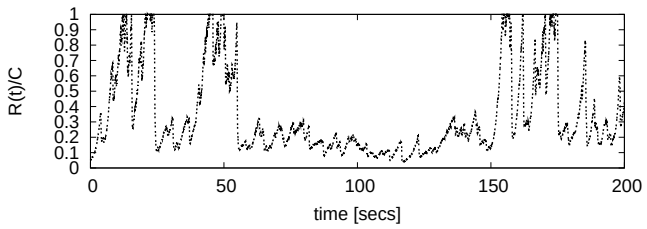

Fig. 5. The figure shows the normalized rate, $R(t)/C$ versus time for Poisson flow arrivals with pareto distributed flow sizes. Bottleneck capacity, C = 150 Mb/s, RTT = 100ms, offered load = 0.7, mean flow size = 30 pkts (1000 byte/pkt) and shape parameter is 1.2. Initial rate, $R(0) = 0.05C$. $R(t)$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

stable region obtained by Bode and Nyquist analysis of the linearized system. Details of the linear stability analysis are given in section IV. 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 [8].

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 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 III.

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 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.

4) 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 III-C.3 where flows with RTT ratios up to two orders of magnitude co-exist on a single link and RCP successfully emulates processor sharing.

5) 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 III that this difference between RCP and XCP contributes to a big difference in their performance.

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.[5]

D. 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 don't work well for short-lived flows, yet most flows in the Internet are of this type [3]. An example of the RCP startup mechanism is illustrated in Figure 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 piggybacked on the data and the ACK messages.

III. RCP Performance

A. Simulation Setup

In this section we study RCP's performance using ns-2 [4] (Version 2.26) augmented with RCP end-host and queue modules.

[5]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.

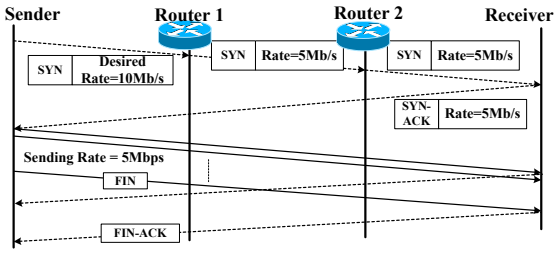

Fig. 6. The sender sets the desired rate in the SYN packet, and a router can overwrite this rate if the current rate R that it can give out is lower then the desired value. The receiver then sends back the rate in the SYN-ACK packet. If the flow lasts longer than one RTT, the subsequent rates are piggy-backed on the data and ACK packets.

We compare the performance of RCP with processor sharing, TCP and XCP. We are primarily interested in the average flow completion time (AFCT).[6] Flow completion 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 queueing 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 [5]:

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

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, 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 [6]. 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, $T = \min(10\text{ms}, RTT)$ 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 [7], 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. It is known that the session arrivals can be accurately modeled as Poisson and that the

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

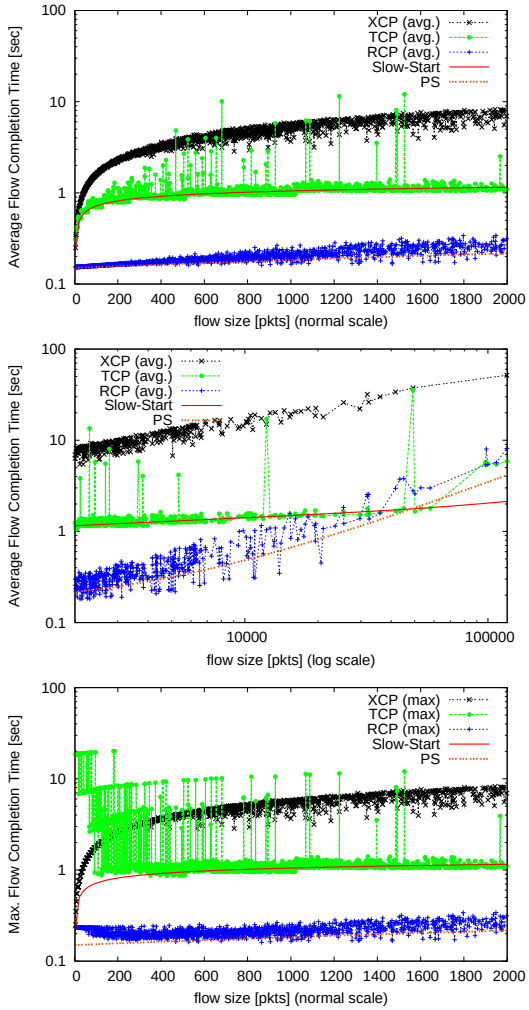

Fig. 7. AFCT for different fbw sizes when $C = 2.4$ Gb/s, RTPD=0.1s, and $\rho = 0.9$. Flows are pareto distributed with $E[L] = 25$ pkts, shape = 1.2. The top plot shows the AFCT for fbw sizes 0 to 2000 pkts; the middle plot shows the AFCT for fbw sizes 2000 to 10 ^{4} pkts; the bottom plot shows the maximum fbw completion time among all fbws of the particular size.

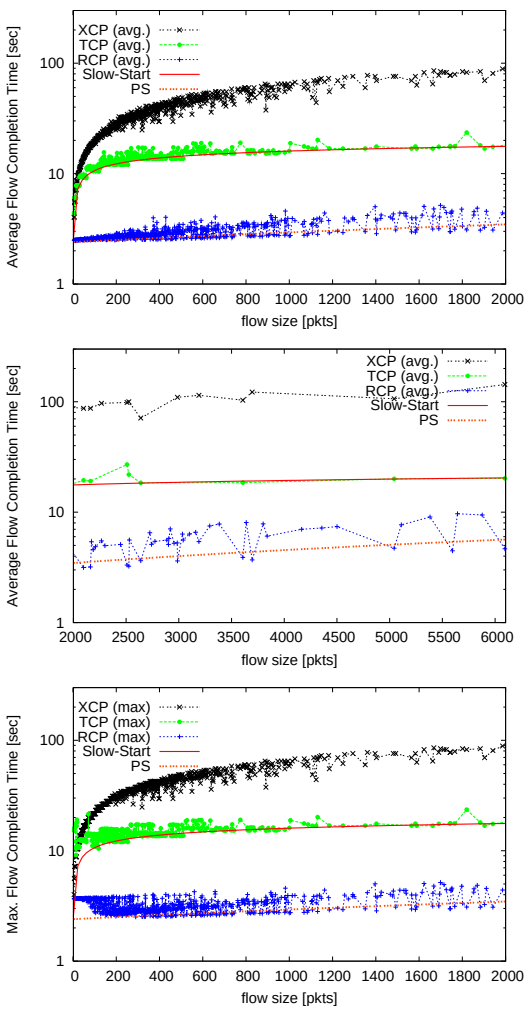

Fig. 8. AFCT for different fbw sizes when C=0.15 Gb/s, RTPD=1.6s, and $\rho = 0.9$. Flows are pareto distributed with $E[L] = 25$ pkts, shape = 1.2. The top plot shows the AFCT for fbws sizes 0 to 2000 pkts; the middle plot shows the AFCT for fbw sizes 2000 to 6000 pkts; the bottom plot shows the maximum fbw completion time among all fbws of the particular size.

flows within a session could be bursty in nature [9]. However it is reasonable to suppose that flows arrive as a Poisson process, which would be the case when they correspond to a large number of independent sessions [1]. Unless otherwise mentioned we will assume that flows arrive as a Poisson process with rate λ and flow sizes are pareto distributed [3], [9]. The offered load on a link is $\rho = \lambda E[L]/C$. In our simulations we vary each of the following parameters – $\rho, E[S], C, RTPD$, flow size distribution, arrival process distribution – while keeping the rest constant, and observe how RCP, TCP and XCP compare with PS when a network or traffic parameter is varied from one extreme to the other.

B. 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.

1) 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 two particular simulation setups. These setups are chosen to represent high bandwidth-delay product ($C \times \mathrm{RTPD}$) networks, since this is the scenario that often differentiates the performance of protocols. In both setups flow sizes are pareto distributed.

• Setup 1: C = 2.4 Gbps, RTPD = 100 ms, ρ = 0.9

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.

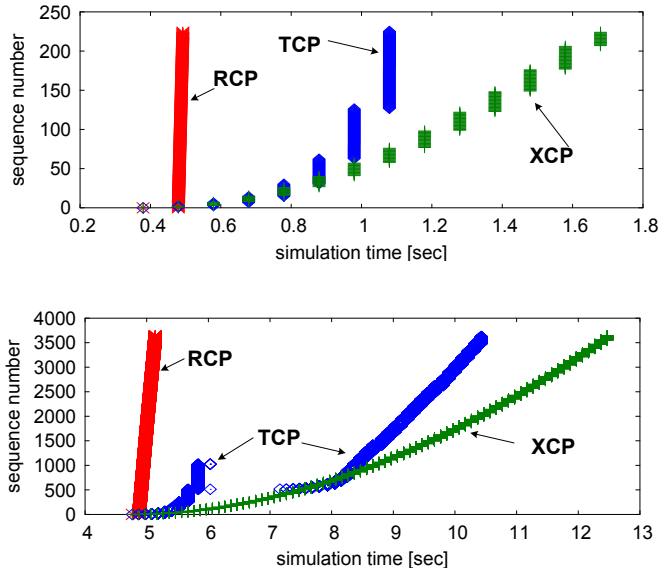

Fig. 9. Time evolution of the sequence numbers of two flows under TCP, XCP and RCP, chosen from the simulation set up of Figure 7. The flow size in the top plot is 230 pkts, and in the bottom plot is 3600 pkts.

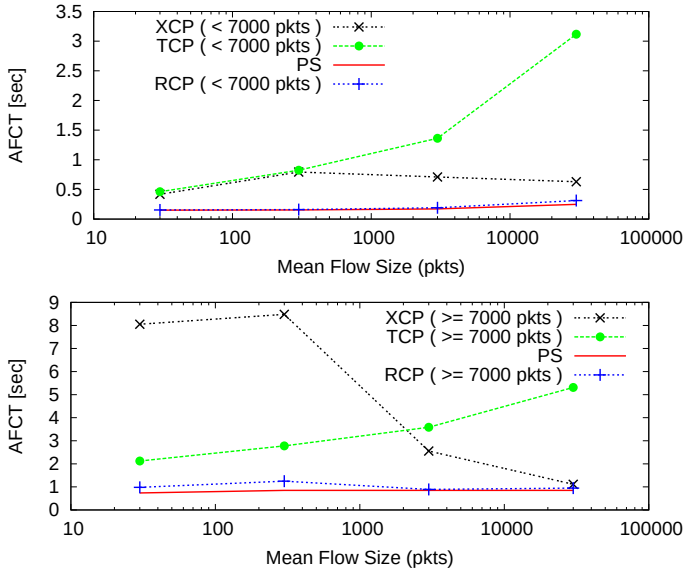

Fig. 10. Comparison of AFCT as the mean flow size increases. Flows are pareto distributed with shape 1.2 and the mean flow size varies as shown on x-axis. $C = 2.4$ Gb/s, RTPD $= 0.1$s and $\rho = 0.8$. The top plot shows the AFCT for flows with < 7000 pkts vs. mean flow size; the bottom plot shows the AFCT for larger flows (> 7000pkts) vs. mean flow size.

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.

• Setup 2: C = 150 Mbps, RTPD = 1.6 s, ρ = 0.9

AFCT is plotted against flow size in the top two graphs of Figure 8. Once again, the AFCT for RCP is always lower than for TCP and XCP and is close to the PS delay for all flow sizes. TCP's delay is about five times higher than RCP, while XCP is 20 times higher.

The third figure shows the maximum delay for all three algorithms, and again TCP has the highest delay variance. RCP's mean and maximum are close.

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 both figures, 7 and 8, 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 queueing 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 9 (top 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 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 9 (bottom plot).

RCP vs. XCP: The time evolution of XCP for two sample flows is shown in Figure 9. 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.

2) When mean flow size increases: Figure 10 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 top plot shows the AFCT averaged over flows with $< 7,000$ pkts and the bottom one is for flows $\geq 7,000$ pkts.[7] 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: For small flows, XCP performs far worse than TCP – see bottom plot of Figure 10. As flows get larger, XCP's performance gets closer to PS while TCP deviates further from it.

XCP vs. TCP: The reversal in performance of XCP and TCP is also clearly illustrated in Figure 11. The top plot shows a snap shot of the AFCTs for $E[L] = 30$ pkts and the bottom two plots are for $E[L] = 30000$ pkts. In the bottom plot the

[7]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.

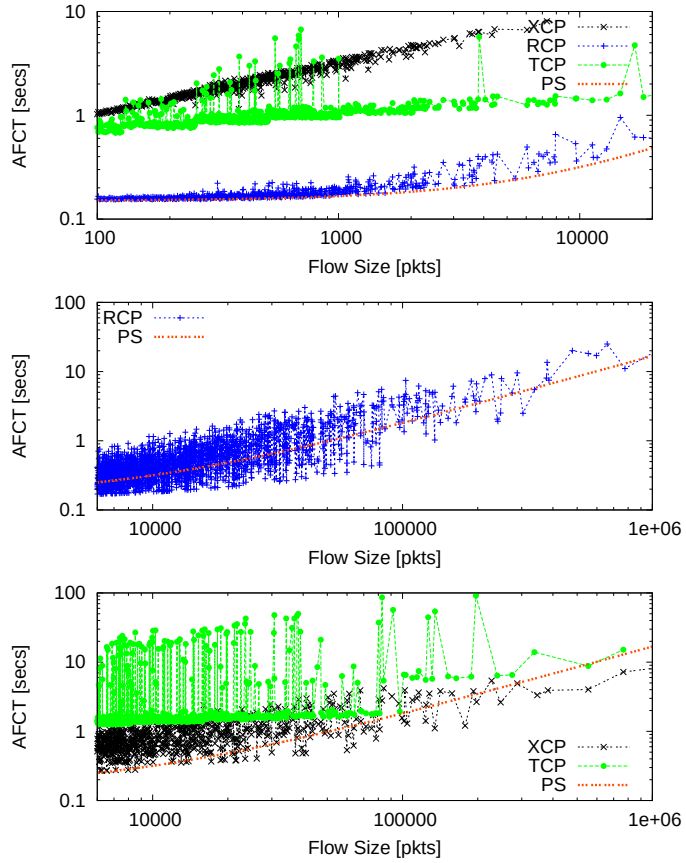

Fig. 11. Comparison of AFCT as the mean fbw size increases. The simulation set up is the same as in Figure 10. The top graph shows the AFCT vs. fbw size when $E[L] = 30$; the middle (RCP) and bottom graph (TCP, XCP) show the AFCT vs. fbw size when $E[L] = 30000$ pkts. RCP does close to PS irrespective of mean fbw size. The performance of XCP and TCP are reversed with the increase in the mean fbw size.

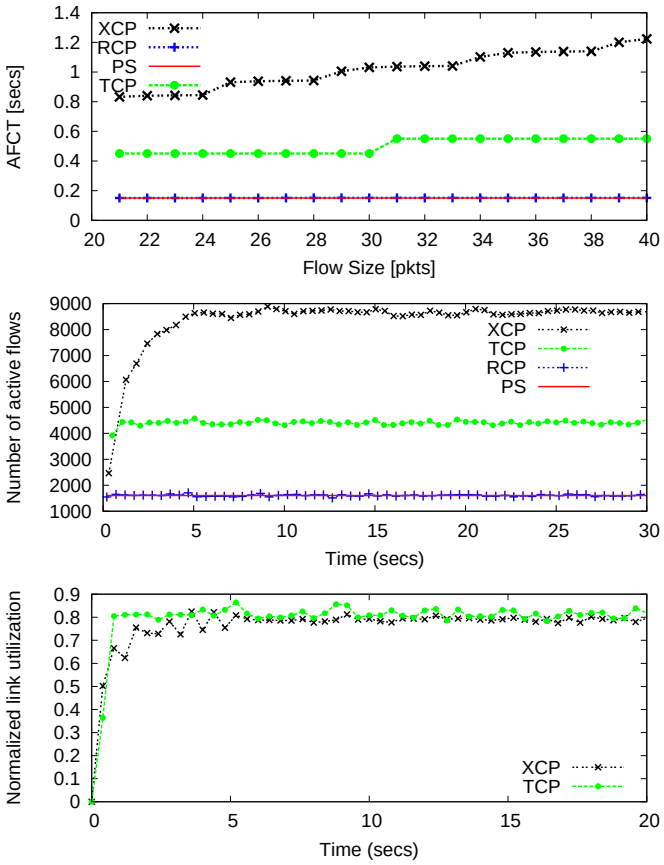

Fig. 12. Comparison under uniform distributed fbw sizes $[20, 40]$. $C = 2.4$ Gbps, RTPD $= 0.1$ sec, $\rho = 0.8$. The top graph shows the AFCT vs. fbw size, the middle grapgh shows the number of active fbws verses time and the bottom graph shows the normalized link utilization (RCP utilization is omitted for sake of clarity). Note how XCP builds up work in the system due to it's conservative nature of giving out bandwidth to fbws.

AFCT 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 [10] 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 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 will see next that this phenomenon is not specific to Pareto distributed flows, but happens with other distributions as well.

3) Different flow size distributions: We simulated RCP, TCP and XCP under several different flow size distributions *and* for different parameters of each distribution. Due to space constraints we are only able to present a small sample here. We chose three distributions – uniform, exponential and Pareto – to represent super exponential-tailed, exponential-tailed and heavy-tailed distributions. Figures 12, 13, 14 and show the AFCT and number of active flows for uniform, exponential and

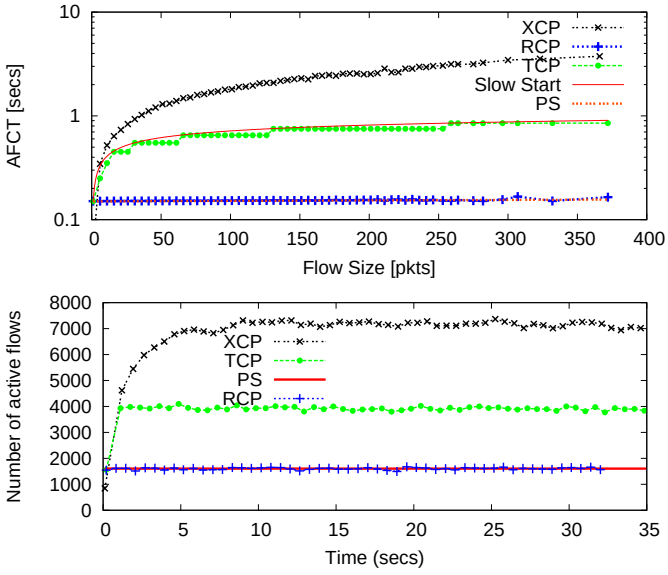

Fig. 13. Flows sizes have exponential distribution with $E[L] = 30$ pkts. The rest of the set up is same as in Figure 12.

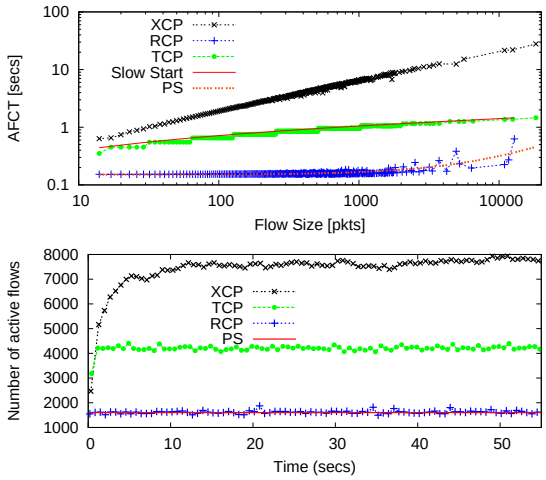

Fig. 14. Flows have pareto distributed fbw sizes with $E[L] = 30$ pkts and shape parameter = 1.8. The rest of the set up is same as in Figure 12.

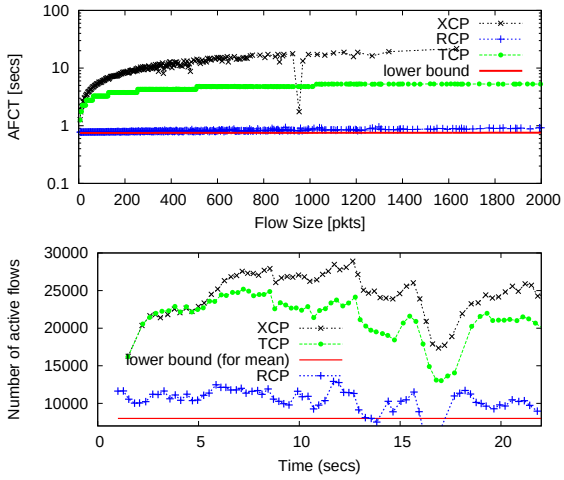

Fig. 15. Comparison when fbws arrival times are pareto distributed, shape = 1.2. $E[L] = 30$ pkts, $C = 2.4$ Gbps, RTPD $= 0.5$ sec, $\rho = 0.9$. RCP, TCP and XCP are compared with the minimum possible AFCT $= 1.5RTPD + \frac{L}{C}$. The lower bound for the mean number of active fbws is given by Little's Law.

Pareto respectively. The take away point in all these plots is that RCP follows PS closely independent of the flow-size distribution.

As noted before, because the mean flow size is small compared to the bandwidth delay product, XCP has a large number of active flows and an AFCT higher than TCP.[8]

4) Non-Poisson flow arrivals: Up until now we have only considered the case when new flows arrive according to a Poisson process. In practice, "flash crowds" can happen due to a sudden interest in a website, time of day, a particular set of data, or just bad luck. While we can expect a sudden burst of new flows to upset RCP — in particular, to temporarily upset RCP's estimate of $N(t)$ — RCP should recover and converge to the new situation within a few RTTs.

[8]In some of the simulations, not shown here, the number of active fbws in XCP was still growing when the simulation ended and did not converge to any equilibrium point. We are investigating whether this is just because XCP does not have any equilibrium point i.e. the system is stochastically unstable in the sense of $E[N] \to \infty$ or that it has an equilibrium but which is a larger number than the simulation reached.

Our simulations suggest that this is the case. We simulated RCP, TCP and XCP with bursty flow arrivals using a Pareto distribution. Figure 15 shows one such example. Qualitatively, our results are the same as we observed with Poisson flow arrivals. Since we do not have an expression for PS under a $G/G/1$ model, we compare the AFCT with its lower bound i.e. $1.5RTPD + L/C$. RCP compares well with the lower bound in all cases we considered.

5) As load increases: The results so far were for the case when average offered load is at or below 90%. Figure 16 shows the AFCT as the offered load is varied. The main observations are:

1) The AFCT of RCP follows that of PS closely.
2) Under low loads and for large flows, TCP's AFCT is as much as fourteen times higher than PS (and RCP) but improves with increased load. XCP on the other hand has sufficient spare capacity under low loads so as not to make all flows last many RTTs. On the other hand XCP's performance gets worse with an increase in load – because with increased load, the system maintains more active flows and hence increases flow duration.

We have seen such a switch in performance between XCP and TCP once before in section III-B.2 as the mean flow size increases. We see that here again as the offered load increases. A detailed plot is shown in Figure 17 under a different setting, for a low offered load of 0.2 and in Figure 18 for a high offered load of 0.94. Notice that XCP performs better than TCP on average under a low load regime and vice versa under a high load regime. RCP's performance is fairly independant of the offered load.

6) Long-lived flows: RCP works well for long-lived flows, as well as for a mix of flow lengths.

As an example, consider a single link bottleneck with $C =$ 150 Mb/s. The flows have RTPDs uniformly distributed from 20-200ms. Twenty flows start at $t = 0$, then 20 more flows arrive at $t = 40$s, and 20 flows finish at $t = 100$s. Figure 19 shows the time evolution of the rate factor $\gamma(t) = R(t)/C$ and the measured utilization at the bottleneck link. In equilibrium, the value of C/R exactly equals N; i.e. as expected, RCP shares the bottleneck link fairly across all flows, and the link utilization at the bottleneck is close to 100%.

C. When Network Conditions Vary

In this section we explore how well the congestion control algorithms match PS under different network conditions. We will vary the link capacity, round-trip times, and increase the number of bottlenecks. In each case, we find that RCP matches PS closely.

In the simulations that follow, flows arrive as a Poisson process with pareto distributed flow sizes, $E[L] = 25$ pkts, shape = 1.2 [3].

1) When link capacity increases: As link capacity increases, the bandwidth-delay product increases, and so more flows can potentially complete in fewer RTTs. Figure 20 shows the AFCT for flows < 500 pkts and ≥ 500 respectively, [9] for link capacities varying from 100 Mbps to 9.6 Gbps [10]. AFCT for RCP is

[9]99.5% of fbws have size < 500 pkts

[10]RTPD and load are fi xed at 100 ms and 0.9 respectively for all simulations.

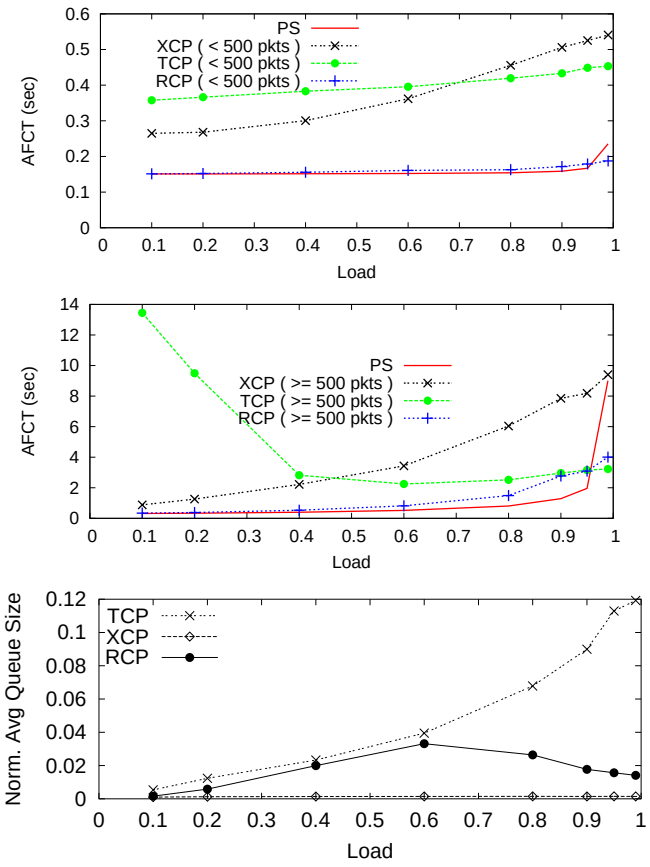

Fig. 16. Comparison of RCP, TCP and XCP under different loads with $C = 150$ Mb/s and RTPD $= 0.1$s. The top two plots shows the AFCT vs. load and the bottom plot shows the normalized average queue length vs. load.

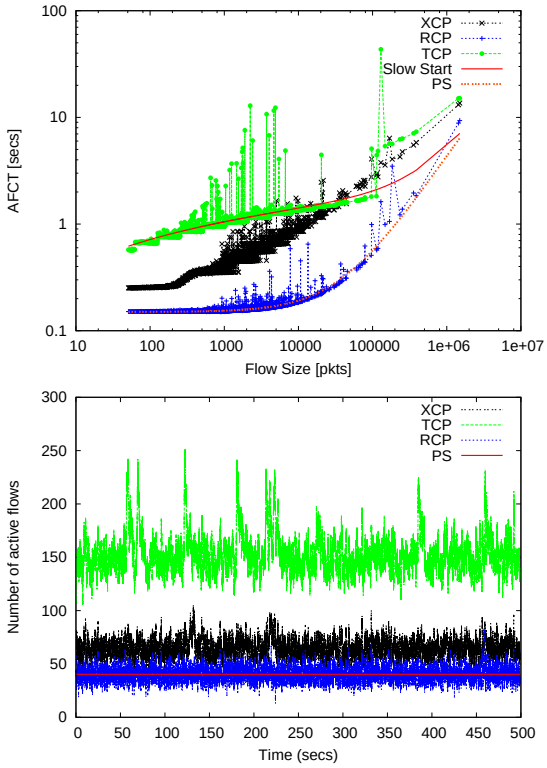

Fig. 17. Comparison of RCP, TCP and XCP under low offered load of $\rho = 0.2$, $C = 2.4$ Gb/s, RTPD $= 0.1$s, pareto flows $E[L] = 300$ pkts. The top plot shows the AFCT vs. flow size and the bottom plot shows the number of active flows vs. time. XCP finishes flows faster than TCP under this low load regime.

always lower than for TCP and XCP, and is close to PS for any link capacity. TCP and XCP require multiple RTTs to find the rate for a flow, and so short flows are forced to last longer than necessary. As link rates increase, the problem gets worse for short flows.

For flows with fewer than 500 packets, flows last three times longer with XCP than with RCP; and twice as long for TCP than for RCP. For flows with more than 500 packets, flows last 30 times longer with XCP, and six times longer with TCP than with RCP.

Figure 20 also shows the normalized queue sizes i.e. average queue size divided by the product $C \times \text{RTPD}$. The normalized queue size of RCP is 2% of the bandwidth-delay product for all the link capacities and is smaller than that of TCP. XCP always maintains a very small queue size.

2) When Round Trip Propagation Delay increases: Increasing the path length also increases the bandwidth-delay product, making it possible for flows to complete in fewer RTTs.

Figure 21 shows the performance for different RTPDs ranging from 10 ms to 1.6 s, with a fixed $C = 150$ Mb/s and $\rho = 0.9$.

Again, RCP flow duration is lower than for TCP and XCP, and follows the PS line closely for all RTPDs. Flows with fewer than 500 pkts last about 2.5 times longer with TCP and XCP than for RCP. And flows with more than 500 pkts last four times longer with TCP and thirteen times longer with XCP than with RCP.

As with PS, RCP is robust to variations in propagation delay: As RTPD increases, the AFCT of flows increases as 1.5 RTPD – the first term in Equation (3); the queueing + transmission delay is independent of RTPD in RCP, just as with PS.

3) 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 22 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. For a more explicit illustration of this fact, Figure 23 shows the AFCT for two different groups of flows. The top figure shows the AFCT for flows with RTPD of 0.02s (smallest value) while the bottom figure is for flows with RTPD 0.2s (largest value). RCP seems robust to variations in RTT and achieves close to the minimum AFCT for each of the group of flows.

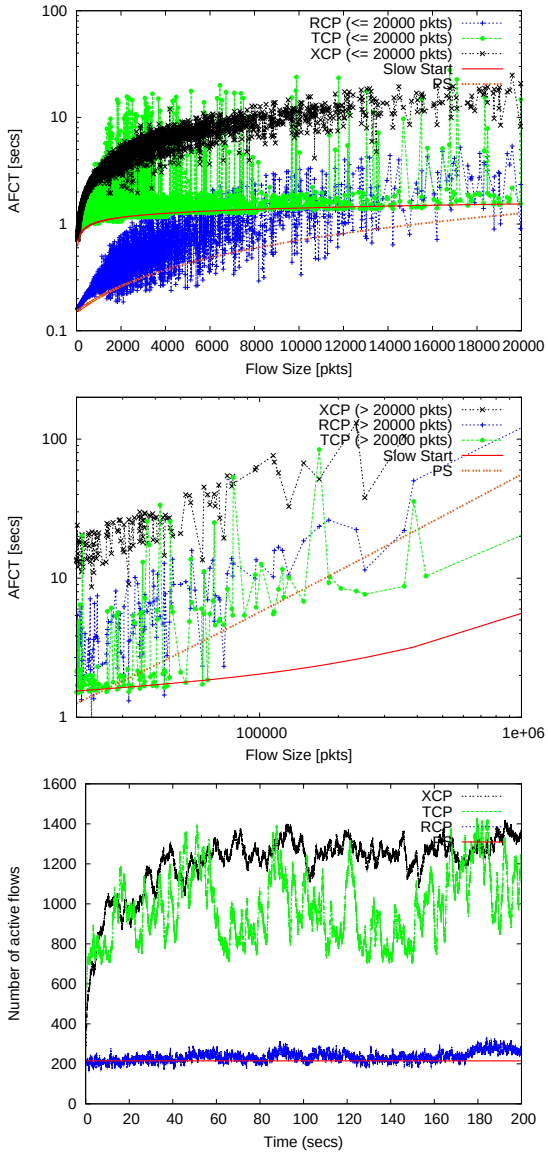

Fig. 18. Comparison of RCP, TCP and XCP under a high offered load of $\rho = 0.94$, $C = 2.4$ Gb/s, RTPD $= 0.1$s, pareto fbws $E[L] = 283.35$ pkts. The top and middle plot show the AFCT verses fbw size and the bottom plot shows the number of active fbws verses times. TCP fi nishes fbws quicker than XCP under the high load regime.

4) When the reverse link is congested: So far the flows encountered a single bottleneck link in the forward direction. In this section the reverse link is also bottlenecked, allowing us to observe any effect on flow completion times when the ACKs encounter congestion. Figure 24 shows the performance of the three protocols when the forward and reverse links are both loaded at 0.75. For comparison purposes Figure 25 shows the corresponding performance when there is no reverse traffic. The only way we can expect a reverse congested link to affect the forward traffic is through the ACKs experiencing queueing delay or being dropped. Since the queue sizes in RCP and XCP are under control and are deliberately driven to zero, we can expect the reverse congestion to have little or no effect on the forward traffic. The results in Figures 24, 25 confirm this. TCP on the other hand maintains large buffer occupancies resulting

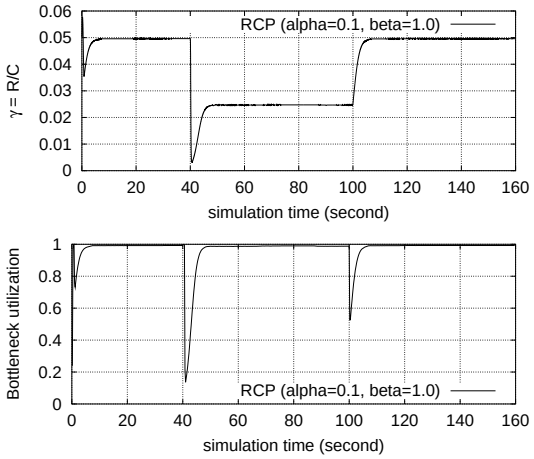

Fig. 19. The time evolution of RCP rate factor $\gamma(t) = R(t)/C$ and measured bottleneck utilization under long-lived fbws. At $t = 0$, 20 fbws start; at $t =$ 40s, 20 more fbws start; at $t = 100$s, 20 fbws fi nish. In each case, $C/R(t)$ converges to $N(t)$.

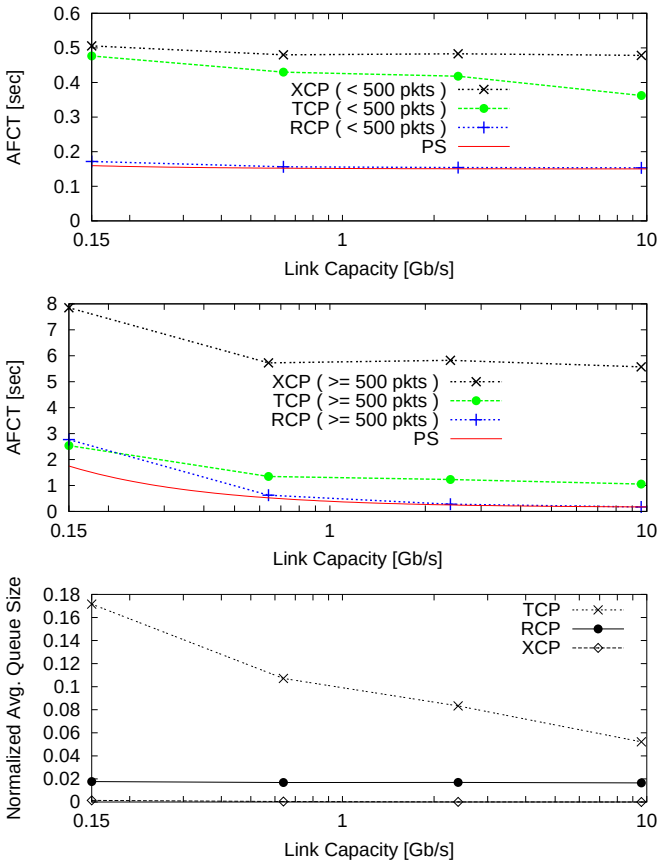

Fig. 20. The comparison of RCP, TCP and XCP under different capacities; RTPD $= 0.1$s and $\rho = 0.9$. The top plot shows AFCT of fbws with size < 500 pkts, the middle plot shows AFCT of fbws with size ≥ 500 pkts. The bottom plot shows the normalized average queue size.

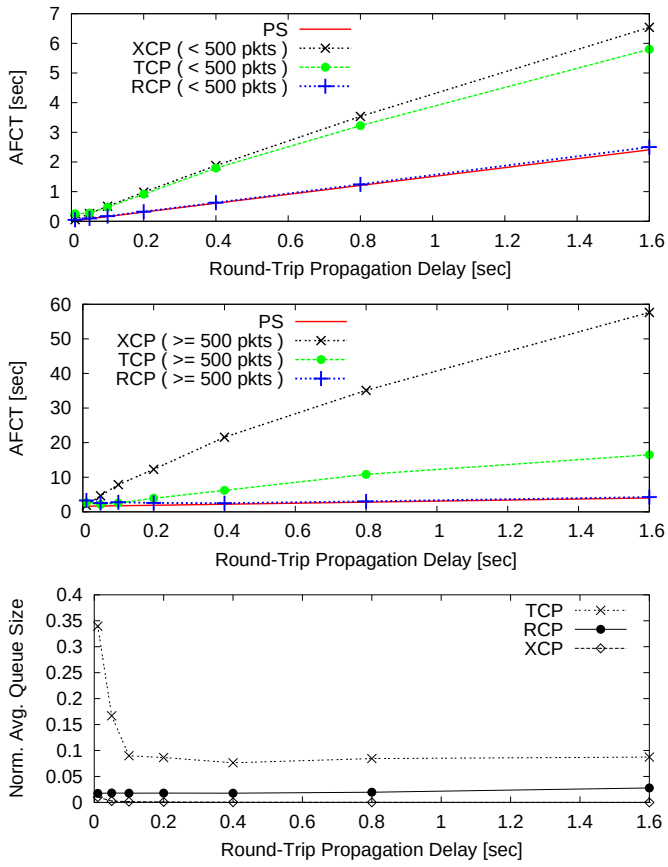

Fig. 21. The comparison of RCP, TCP and XCP under different RTPDs, C =150Mb/s, ρ =0.9. The top plot shows AFCT of fbws with sizes $<$ 500 pkts and the middle plot shows AFCT of fbws with sizes $\geq$ 500 pkts. The bottom plot shows the normalized average queue size.

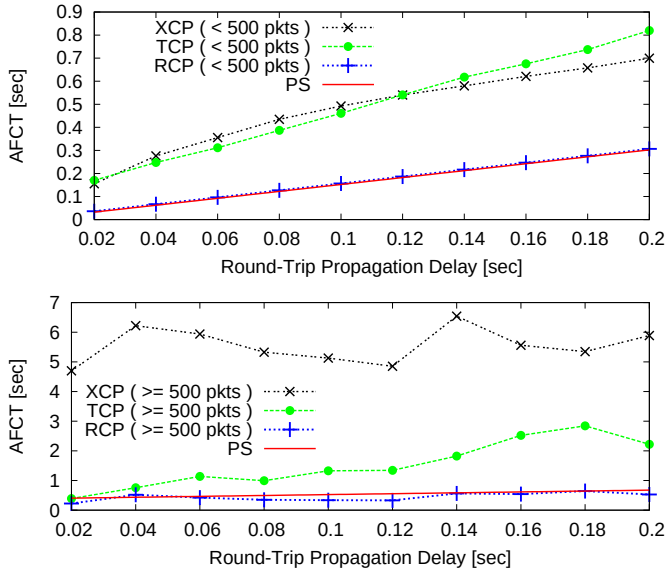

Fig. 22. Comparison of RCP, TCP and XCP when fbws with different RTPDs coexist on a single bottleneck of $C = 0.64$ Gb/s. RTPD of fbws vary from 0.02s to 0.2s. The top fi gure is the AFCT of fbws with fbw size $\leq$ 500 pkts and the bottom fi gure shows the AFCT for fbws with size $>$ 500 pkts.

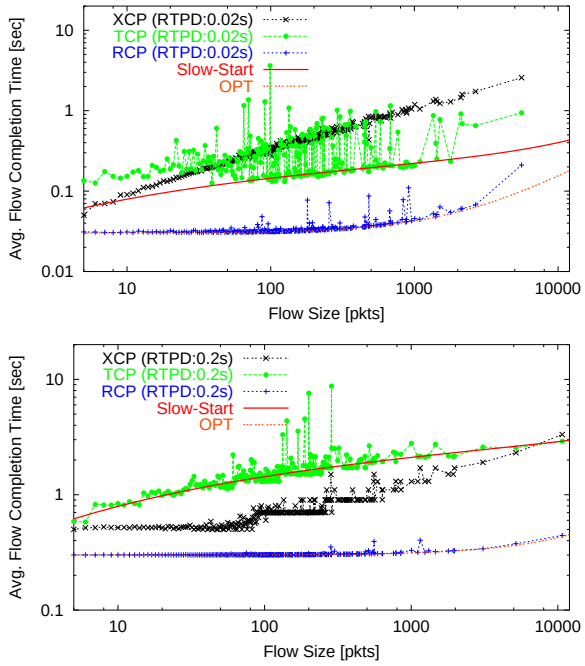

Fig. 23. Comparison of RCP, TCP and XCP when fbws with different RTPDs coexist on a single bottleneck of $C = 0.64$ Gb/s. RTPD of fbws vary from 0.02s to 0.2s. The upper fi gure is the AFCT of fbws with RTPD = 0.02s and the lower fi gure shows the AFCT for fbws with RTPD = 0.2s.

in delayed or dropped ACKs. This results in increased timeouts at the sender and eventually increased flow completion times.

TCP's performance deterioration under reverse congestion is further illustrated in Figure 26 where now there is a high offered load of 0.95 on both forward and reverse links. We saw in section III-B.5 that under high loads TCP out performs XCP. But in the presence of reverse congestion, TCP performs just as poorly as XCP. Infact in this particular simulation with a high load on both forward and reverse link, TCP and XCP appear to be stochastically unstable in the sense that $E[N(t)]$ does not converge. Unlike with TCP, XCP's instability is probably just due to the heavy offered load and has nothing to do with the reverse traffic. In case of RCP, the performance is slightly worse than PS, nevertheless comparable to PS and stable.

A final example is shown in Figures 27, 28 with a heavy load of 0.95 on the forward link and a light load of 0.2 on the reverse link. The results reiterate the observations made before — for the heavily loaded link, the active number of flows in XCP continually increases, TCP flows see a large variance in the flow completion times, while RCP's prformance is comparable to PS. In case of the lightly loaded link, as was seen in section III-B.5, XCP performs better than TCP.

5) When there are multiple bottlenecks: While flows typically encounter only one bottleneck, it is possible for a flow to encounter two or more. Figure 29 shows a topology with n cascaded bottleneck links. Flows in group 0 traverse all the bottleneck links, and flows in group $i (i = 1, .., n-1)$ use only the bottleneck link i. All the bottleneck links has the same capacity C and the same delay, and all flow groups have the same arrival rate λ.

Here, we investigate the effect of increasing the number of

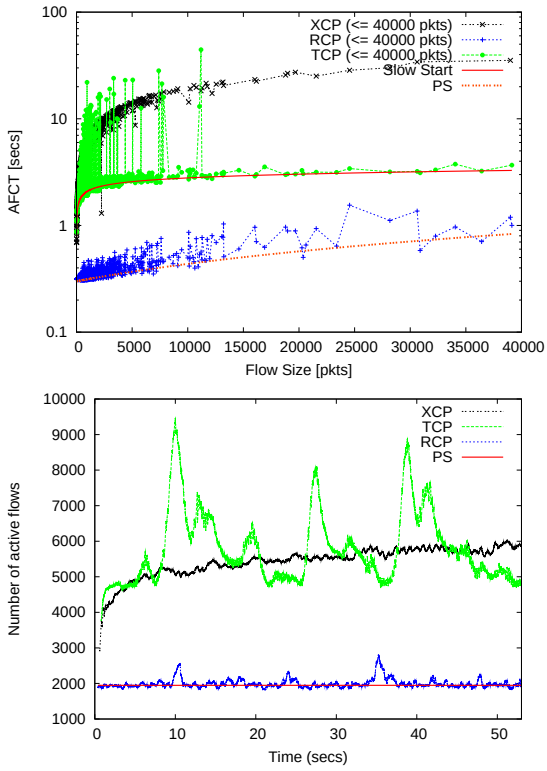

Fig. 24. Comparison of RCP, TCP and XCP when both forward and reverse link are bottlenecked. Offered load $= 0.75$, $E[L] = 46$ pkts, $C = 2.4$ Gbps, RTT $= 0.2$s. The top plot shows the AFCT versus fbw size and the bottom plot shows the number of active fbws over time. Compare the metrics here with Figure 25 and notice the increased variance in TCP's delay and the number of active fbws. The reverse traffi c metrics (not shown here) are qualitatively similar.

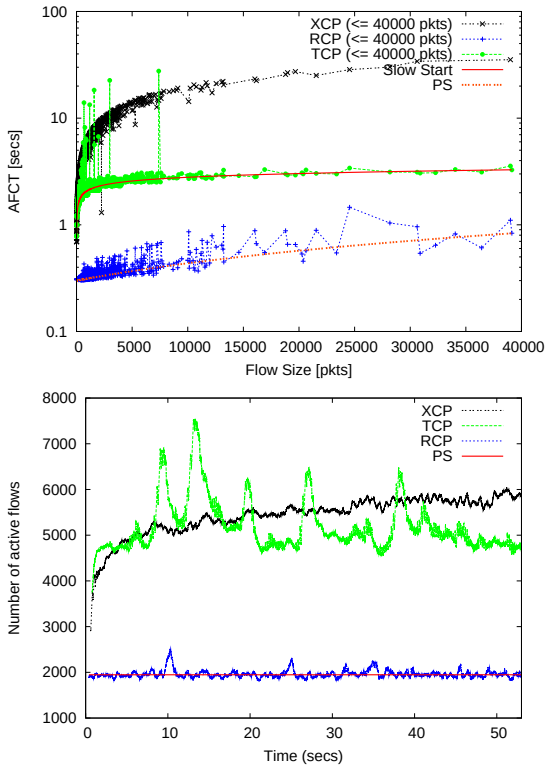

Fig. 25. Comparison of RCP, TCP and XCP when only forward link is congested. Offered load $= 0.75$, $E[L] = 46$ pkts, $C = 2.4$ Gbps, RTT $= 0.2$s.

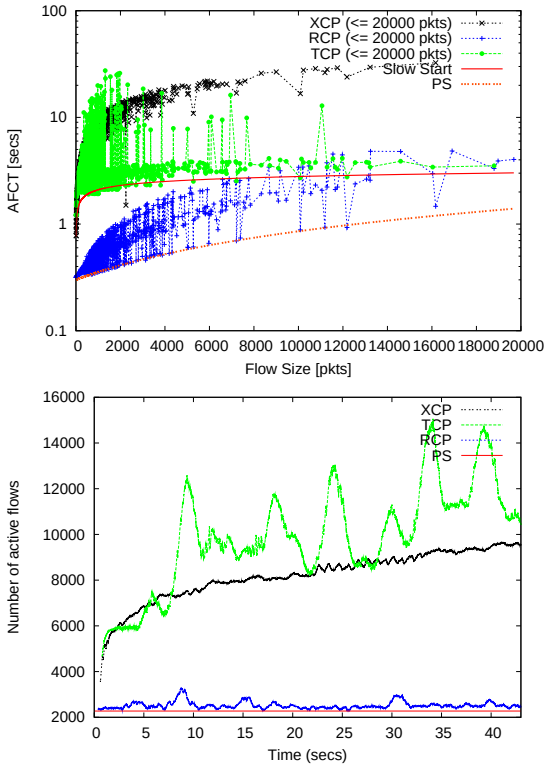

Fig. 26. Comparison of RCP, TCP and XCP when both forward and reverse link are bottlenecked. Offered load $= 0.95$, $E[L] = 46$ pkts, $C = 2.4$Gbps, RTT $= 0.2$s. The top plot shows the AFCT versus fbw size and the bottom plot shows the number of active fbws over time. XCP and TCP appear to be stochastically unstable. RCP's performance deviates slightly from PS but is stable.

bottlenecked hops on flows in group 0. While fixing the RTPD of flow group 0 at 0.2s, we increase the number of hops to one, two, four, eight and observe how AFCT changes. The AFCT for flow group 0 is plotted in Figure 30. RCP always achieves a delay close to the lower bound irrespective of the number of hops. The performance of TCP degrades as the number of hops increase. We noticed that TCP flows have a greater chance of experiencing losses when traversing through multiple bottlenecks, and in general have a lower throughput.

The detailed plots of AFCT vs. flow-size as the number of bottlenecked links increase are shown in Figure 31. Notice how the performance of TCP degrades as the number of bottlenecks increase; the AFCT of TCP gets farther away from the Slow-Start curve as the flow passes through more bottlenecks. This is because the probablility of a loss increases as a flow traverses through multiple bottlenecks, and hence having a greater chance of entering into AIMD phase.

RCP achieves max-min bandwidth sharing in the multiple bottleneck case, and with multiple bottlenecks, flows complete an order of magnitude faster with RCP than with TCP or XCP.

IV. Stability Analysis

A desirable characteristic of any congestion control scheme is that it is stable, in the sense that — under equilibrium load conditions — it converges to some desirable operating behavior. Significant research has lead to an understanding of the stability region for TCP and other algorithms [31], [10].

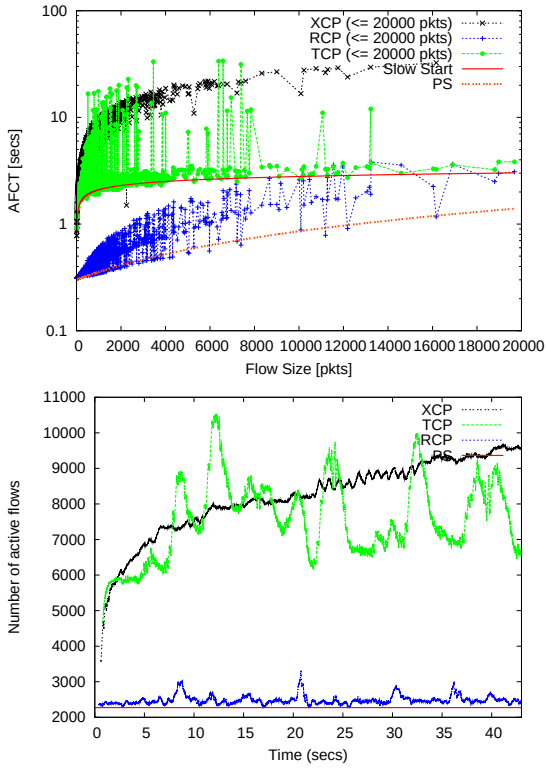

Fig. 27. Comparison of RCP, TCP and XCP when both forward and reverse link are bottlenecked. Forward link: offered load $= 0.95$, $E[L] = 46$ pkts, $C = 2.4$Gbps, RTT $= 0.2$s. The top plot shows the AFCT versus fbw size and the bottom plot shows the number of active fbws over time. XCP appears to be stochastically unstable. TCP fbws have a large variance in the fbw completion times. RCP's performance deviates slightly from PS but is stable.

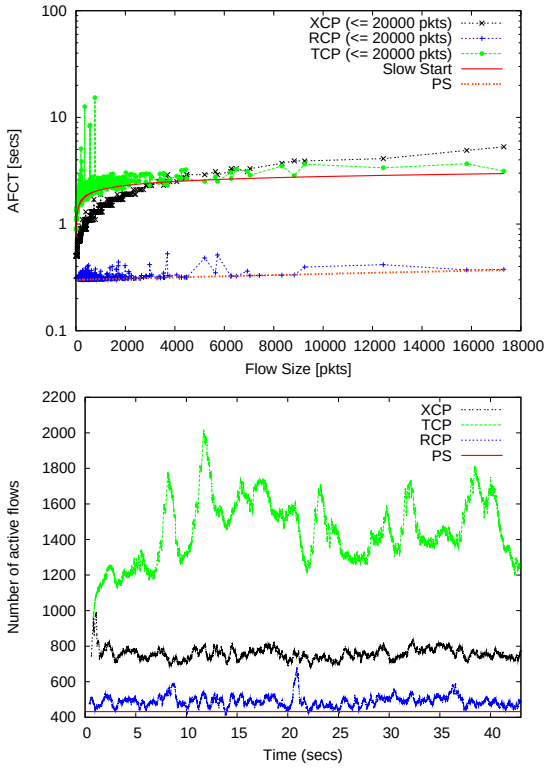

Fig. 28. Comparison of RCP, TCP and XCP when both forward and reverse link are bottlenecked. Reverse link: offered load $= 0.2$, $E[L] = 46$ pkts, $C = 2.4$Gbps, RTT $= 0.2$s. The top plot shows the AFCT versus fbw size and the bottom plot shows the number of active fbws over time. As expected under low load, XCP performs better than TCP, while RCP is close to PS.

In this section we derive the stability region of RCP for long-lived flows. We find that we can choose RCP's parameters, so that the network is stable independent of the link capacity, number of flows and round trip delay.

A. RCP System Equations

The RCP system is described by the following coupled non-linear differential equations:

$$\dot{R}(t) = R(t-T)\left(\frac{\alpha(C - y(t)) - \beta\frac{q(t)}{d(t)}}{Cd(t)}\right) \quad (4)$$

$$y(t) = NR(t-d_0) \quad (5)$$

$$d(t) = d_0 + \frac{q(t)}{C} \quad (6)$$

$$\dot{q}(t) = [y(t) - C]; \quad q(t) > 0 \quad (7)$$
$$= \max[y(t) - C, 0]; \quad q(t) = 0$$

where $d(t)$ is the RTT of the flows at t. The rest of the symbols are as we saw in Section II-B.

B. Equilibrium and Linearization

Taking (R, q) as the state, the equilibrium point (R_e, q_e) is defined by $\dot{R} = 0$ and $\dot{q} = 0$. So we get $(R_e, q_e) = (\frac{C}{N}, 0)$. Because the rate update equation (4) is non-linear, we'll first

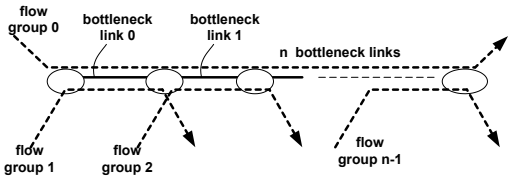

Fig. 29. Multiple bottleneck topology. Flow group 0 traverses all the bottleneck links and the other group of fbws (group 1,..,n-1) use only one bottleneck link.

linearize the RCP rate update equation around the equilibrium point. Note that Equation (7) for the queue evolution is nonlinear as well. This equation has a discontinuity at the equilibrium point, making it hard to deal with the nonlinearity. For now we will ignore this non-linearity and approximate the queue evolution as:

$$\dot{q}(t) = y(t) - C$$

We will come back to this nonlinearity later. The details of the linearization steps are in Appendix II, and lead to the following linearized equations:

$$\delta\dot{q}(t) = N\,\delta R(t-d_0) \quad (8)$$

$$\delta\dot{R}(t) = -\frac{\alpha}{d_0}\delta R(t-d_0) - \frac{\beta}{Nd_0^2}\delta q(t)$$

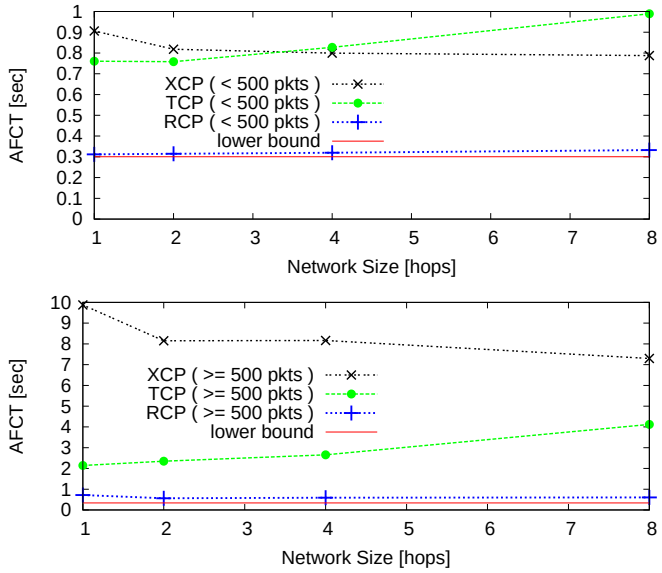

Fig. 30. The comparison of RCP, TCP and XCP under 1,2,4 8 bottlenecks, C=0.64 Gb/s, RTPD of fbw group 0 is 0.1 s, $\rho = 0.9$. The top plot shows the AFCT of fbw group 0 for fbw sizes < 500; the lower plot shows the AFCT of fbw group 0 for fbw sizes ≥ 500.

where $\delta R \doteq R - R_e$ and $\delta q \doteq q - q_e$ are perturbations around the equilibrium point. Taking Laplace transform of the linearized equations gives:

$$\begin{aligned} s\delta R(s) &= -\frac{\alpha}{d_0} e^{-sd_0}\delta R(s) - \frac{\beta}{Nd^2}\delta q(s) \\ s\delta q(s) &= Ne^{-sd_0}\delta R(s) \end{aligned}$$

The block diagram of this feedback control system is shown in Figure 32.

C. Bode Plot Analysis

We will now obtain the stability region of our system via the Bode plot analysis. From Figure 32, the open loop transfer function of the RCP system is:

$$G(s) = e^{-sd_0}\frac{\alpha s d_0 + \beta}{s^2 d_0^2} \tag{9}$$

We can already see that the variables N and C do not appear above in the transfer function, which means that the system stability is independent of the number of flows and link capacity. Let us then see if there is a region for α and β such that the system is stable for any d_0.
From Equation (9) the magnitude and phase of $G(s)$ are given by:

$$\begin{aligned} |G(j\omega)| &= \frac{\sqrt{\beta^2 + (\omega d_0 \alpha)^2}}{(\omega d_0)^2} \\ \angle G(j\omega) &= -\omega d_0 + \arctan(\frac{\omega \alpha d_0}{\beta}) - \pi \end{aligned} \tag{10}$$

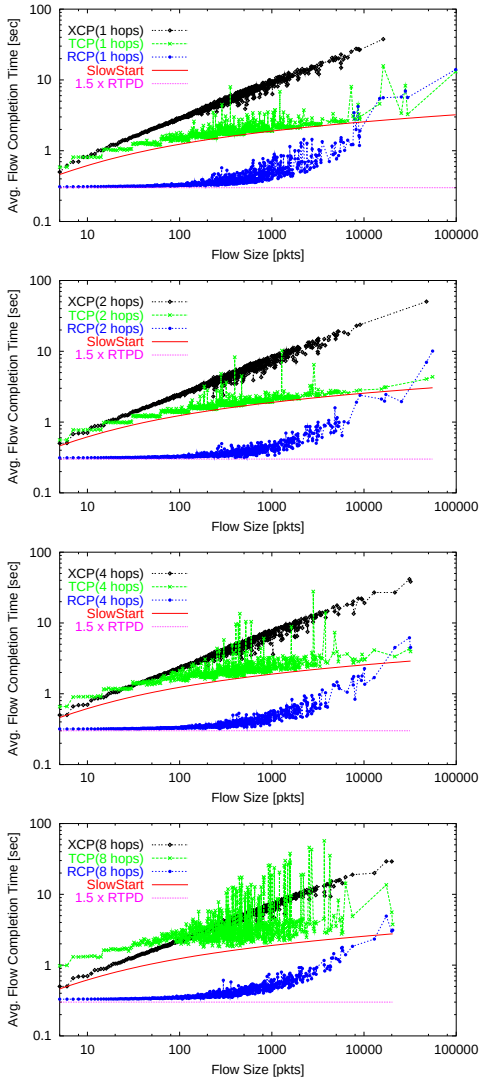

Fig. 31. The comparison of RCP, TCP and XCP under 1,2,4 8 bottlenecks, C=0.64 Gb/s, RTPD of fbw group 0 is 0.1 s, $\rho = 0.9$. Top through bottom are plots showing AFCT of fbw group 0 for number of bottleneck links = 1, 2, 4 and 8 respectively.

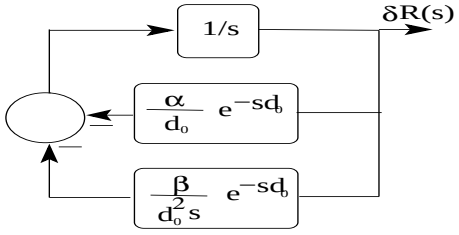

Fig. 32. Block Diagram of the linearized RCP system

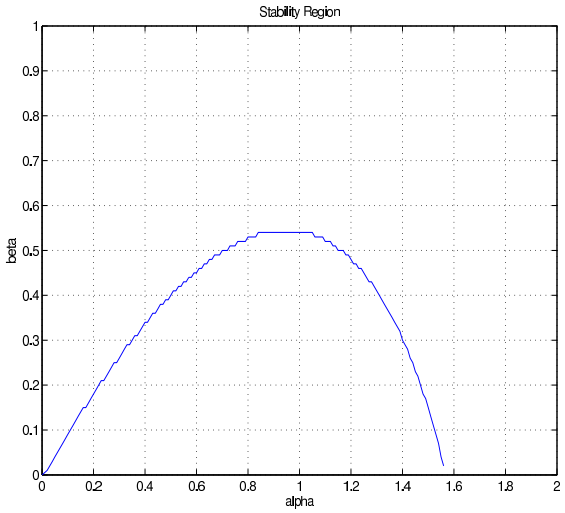

Fig. 33. Region enclosed by the curve is the stable region

The stability criterion for the closed loop system is:

$$|G(j\omega)| < 1 \text{ at } \angle G(j\omega) = -\pi \tag{11}$$

This stability criterion is true for systems where $|G(j\omega)|$ crosses the magnitude $= 1$ line only once. This is true for our system. If ω_z is the frequency at which $|G(j\omega)| = 1$ and ω_c is the frequency at which $\angle G(j\omega) = -\pi$, then the stabilty criterion in Equation (11) is equivalent to $\omega_z < \omega_c$ i.e.

$$\omega_z = \frac{1}{d_0}\sqrt{\frac{\alpha^2 + \sqrt{\alpha^4 + 4\beta^2}}{2}} < \omega_c \tag{12}$$

where ω_c is the solution of the equation:

$$\frac{\omega\, d_0\, \alpha}{\beta} = \tan(\omega d_0) \tag{13}$$

There is no closed form solution to the above equation. We need the following condition for the existence of a non-zero solution to Equation (13). See Appendix III for details on this.

$$\frac{\alpha}{\beta} > 1 \tag{14}$$

All that remains now is to solve for α and β satisfying Equations (12), (13) and (14). We did this in Matlab by first choosing α_i and β_i satisfying (14), then solving (13) numerically for ω_c and finally determining if inequality (12) is satisfied for the chosen α_i and β_i. If it is, then for these parameter values (α_i, β_i) the closed loop system is stable. Observe from (12) and (13) that the term d_0 on both sides of the inequality gets cancelled and hence the the system is stable for this (α_i, β_i), for every d_0. Continueing this way, we obtained the region (α, β) for which the RCP system is stable for all N, C and d_0. This is shown in Figure 33.

D. The Nyquist Stability Analysis

In the last Section we obtained the stability region from the Bode plot analysis. There are some conditions to be satisfied for the Bode analysis to hold. Specifically, the stability criterion, $|G(j\omega)| < 1$ at $\angle G(j\omega) = -\pi$, holds for systems where $|G(j\omega)|$ crosses the magnitude $= 1$ line once, the most common situation. However, there are systems when the $|G(j\omega)|$ crosses magnitude $= 1$ more than once. A rigorous way to resolve these ambiguities is to use the Nyquist stability criterion. So, in this Section we will use the Nyquist criterion and confirm the stability region obtained before.

Recall that the open loop transfer function of our system is given by $G(s) = e^{-sd_0}(\alpha s d_0 + \beta)/(sd_0)^2$. The closed loop transfer function is $G(s)/(1 + G(s))$. Therefore, the closed loop roots are the solutions of $1 + G(s) = 0$ i.e.:

$$d_0^2 s^2 + \alpha d_0 e^{-sd_0} s + \beta e^{-sd_0} = 0 \tag{15}$$

We will write $d_0^2 s^2 + \alpha d_0 e^{-sd_0} s + \beta e^{-sd_0} = 0$ in the form $1 + \alpha \frac{b(s)}{a(s)} = 0$. This is given by:

$$1 + \alpha \frac{d_0 e^{-sd_0} s}{d_0^2 s^2 + \beta e^{-sd_0}} = 0$$

i.e. $b(s) = d_0 e^{-sd_0} s$ and $a(s) = d_0^2 s^2 + \beta e^{-sd_0}$. Let $G_1(s)$ denote $\frac{d_0 e^{-sd_0} s}{d_0^2 s^2 + \beta e^{-sd_0}}$. Then, the procedure for obtaining the stability region can be summarized as follows:
1. Fix a value of β. Plot the nyquist plot of $G_1(s)$
2. Determine the number of unstable (i.e. Right Hand Plane) poles of $G_1(s)$ and call that number P.
3. Determine the region on the real axis where $-\frac{1}{\alpha}$ should lie such that if N denotes the number of encirclements of $-\frac{1}{\alpha}$, then $N + P$ should be equal to zero. $Z = N + P$ are the number of unstable closed loop roots, and therefore we want Z to be equal to 0. Note that, N is negative if the encirclement of $-\frac{1}{\alpha}$ is in anticlockwise direction and positive if in clockwise direction.
To draw the nyquist plot of $G_1(s)$, we will use the Pade approximation for e^{-sd_0} [11]. Substituting this in $G_1(s)$ we get:

$$G_1(s) = \frac{\frac{d_0^3}{12}s^3 - \frac{d_0^2}{2}s^2 + d_0 s}{\frac{d_0^4}{12}s^4 + \frac{d_0^3}{2}s^3 + (d_0^2 + \beta\frac{d_0^2}{12} - \frac{d_0\beta}{2})s + \beta}$$

Let's go through the three steps above with an example. Let us take $\beta = 0.3$. Now, we want to find the range of α for which the system is stable. The nyquist plot of $G_1(s)$ is shown in Figure 34 for $\beta = 0.3$. The value of d_0 taken does not matter, the plot does not change with d_0. $G_1(s)$ has two unstable poles and so $P = 2$. Therefore, in order for Z to be equal to 0 we need $N = -2$, which means $-\frac{1}{\alpha}$ must be encircled twice in anticlockwise direction in the plot. The only region in the plot where there are two anticlockwise enciclements is between points A and B shown in the figure. So the stable region is $-2.91 < -\frac{1}{\alpha} < -0.696$ i.e. $0.3436 < \alpha < 1.436$ when $\beta = 0.3$. And thus continueing, we can obtain the stable range for α for every value of β. When β is larger than about 0.55 there does not exist any value of α for which the system is stable. An example of such a nyquist plot is shown in Figure 35. As can be seen, there is no region on the plot where there are two anticlockwise encirclements, and hence we cannot get Z to be equal to 0, and therefore the closed loop system will have atleast one unstable pole for this value of β.

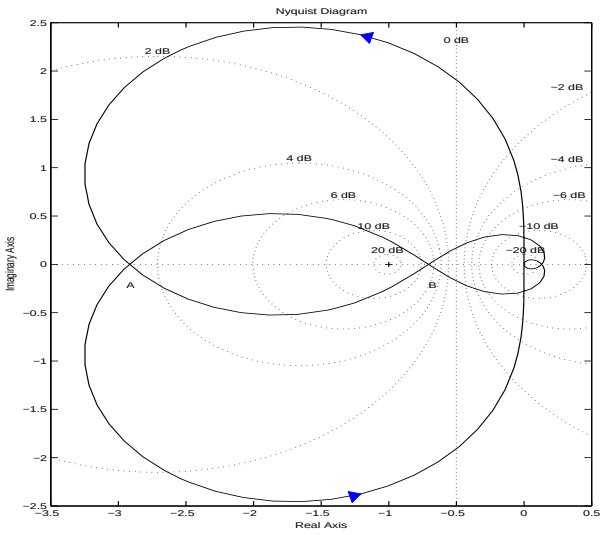

Fig. 34. Nyquist plot of $G_1(s)$ when $\beta = 0.3$

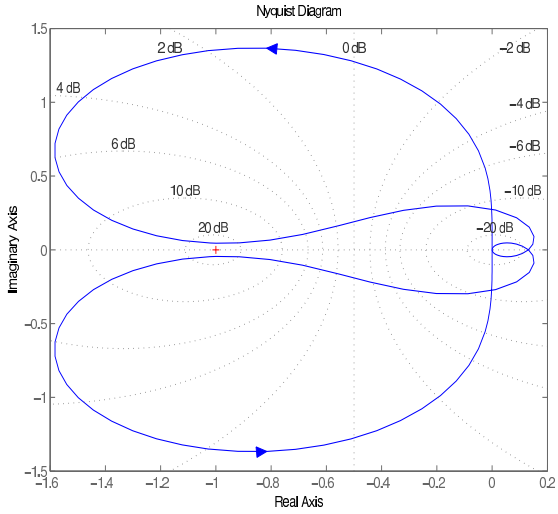

Fig. 35. Nyquist plot of $G_1(s)$ when $\beta = 0.6$

The stable region obtained from the procedure above is shown in Figure 36. The region obtained from the nyquist analysis is shown in $+$ signs, while that obtained from the Bode plot analysis is shown in solid line. As can be seen, they match well. Thus the nyquist criterion analysis confirms the region that we obtained.

E. Stable Region

Let us now step back and understand why the stability region looks like the way it does in Figure 36. Recall the RCP rate update equation:

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

If there is spare capacity available then the role of α in the above equation is to decide how much of this available capacity should

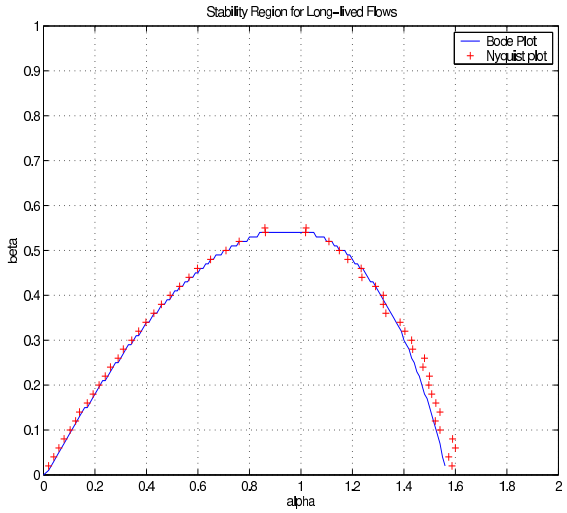

Fig. 36. Stable region obtained from Bode analysis and Nyquist analysis

be allotted to the flows in one round trip time. It is intuitively clear that if α is large, say 10 for example, and if there is spare capacity available then we are increasing the aggregate flow rate by 10 times more than what is available. Obviously, this will result in an overshoot from the equilibrium rate, everytime there is even a small δ amount of spare capacity. So, it is not surprising that α cannot be indefinitely large. As seen in Figure 36, there does not exist a stable region for $\alpha > 1.6$.

The parameter β on the other hand determines the amount that the aggregate incoming traffic should be reduced so that any queue at the router can be drained. Given any stable value of $\alpha, 0 < \alpha < 1.6$, a large value of β only means that we are being very aggressive in draining the queue. It is not intuitive as to why this would lead to an unstable system and why β is limited to values ≤ 0.55. The answer to this is that we did not take into account the non-linearity in the queue size in Equation 7. Without this non-linearity, it would mean that the queue length can become negative as well. This explains why β is capped.

We hypothesize that the actual stability region (i.e. taking the queue non-negativity into account) is larger and includes the one obtained through the linearized model. The initial evidence of this is from simulations. The set up is as follows: 5 long flows, all with the same round trip time, start at time 0. Their equilibrium normalized rate is $\frac{R}{C} = 0.2$. 5 more flows then start at time 40. Now, the equilibrium rate of the 10 flows is 0.1. The original five flows then depart at time 100. This experiment is done for every value of $0.1 \leq \beta \leq 1, 0.1 \leq \alpha \leq 2$ in steps of 0.1. For each value of β we note the α that is the onset of instability. For example in Figure 37 when $\beta = 0.1$, the system does not stabilize for $\alpha \geq 1.6$. In Figure 38, when $\beta = 0.4$, the system is unstable from $\alpha = 1.3$ onwards. And in Figure 39, when $\beta = 0.6$, the system is unstable from $\alpha = 1.2$ onwards. Note that in our linearized stability region, there is no stable point for $\beta = 0.6$. The plots shown here are just a sample of the simulations we did. The stable region obtained as above is shown in Figure 40. It supports our hypothesis that the stable region of the non-linear system is larger than the linearized system.

[11] $e^{-sd_0} \approx (1 - \frac{sd_0}{2} + \frac{(sd_0)^2}{12})/(1 + \frac{sd_0}{2} + \frac{(sd_0)^2}{12})$

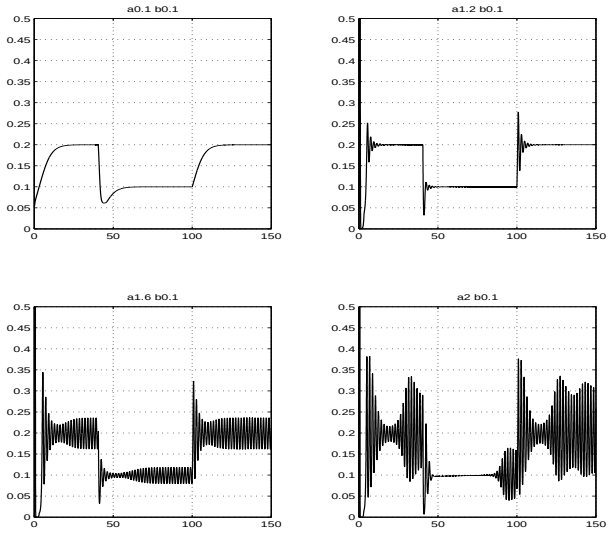

Fig. 37. Long fbw simulations for $\beta = 0.1$. The sytem is unstable for $\alpha \geq 1.6$

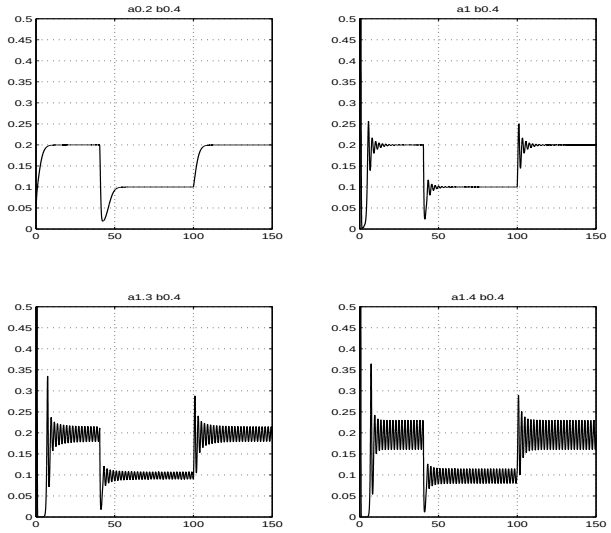

Fig. 38. Long fbw simulations for $\beta = 0.4$. The sytem is unstable for $\alpha \geq 1.3$

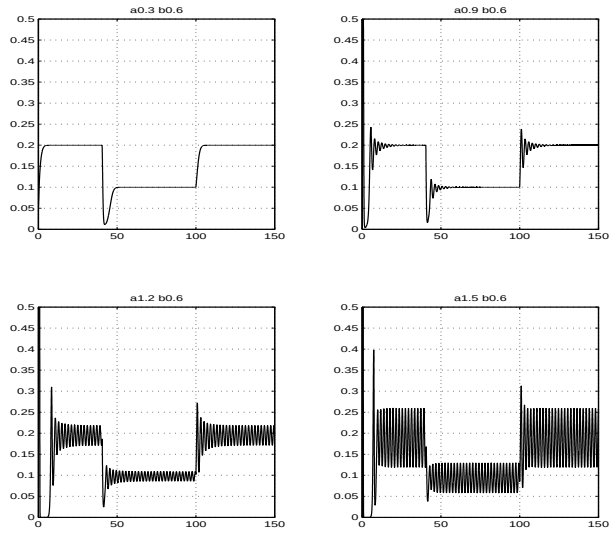

Fig. 39. Long fbw simulations for $\beta = 0.6$. The sytem is unstable for $\alpha \geq 1.2$

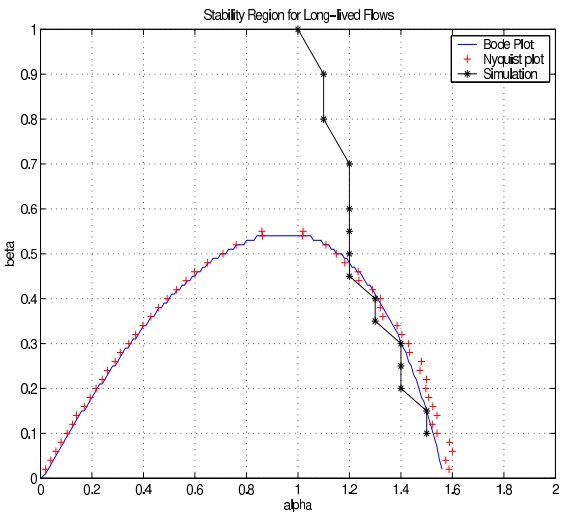

Fig. 40. Stability Region from analysis and simulations

F. Phase Plane Analysis

In this Section we will obtain the stability region of the RCP parameters, taking into account the non-linearity in the queue length equation. We will use a method called *phase-plane analysis*, which is used to solve and understand nonlinear control problems. We will first give a brief introduction to this method. The RCP system described by Equations (4) and (7), can be written as:

$$\begin{aligned} \dot{R}(t) &= f(R(t-T), R(t-d_0), q(t), N, C) \\ \dot{q}(t) &= g(R(t-d_0), N, C) \end{aligned} \quad (16)$$

where $f(.)$ and $g(.)$ are non-linear functions. If we take R and q as coordinates of a plane, then to each state of the system there corresponds a point in this plane. As t varies, this point describes a curve in the $R-q$ plane, indicating the history of the system dynamics. Such a curve is called a trajectory. The geometrical representation of the system behaviour in terms of trajectories is called a phase-plane representation of the system dynamics. The initial condition determines the initial location of a representative point on the trajectory. As time increases, the representative point moves along the trajectory. A family of such trajectories is called a phase-plane portrait.

We will now plot the phase-plane portraits of the RCP system, and determine the stability region from these plots. The phase portraits are generated by a C program which given an initial state (R, q) does a step- by-step evolution of the RCP system state from Equations (4), (6), (5) and (7). Following is the set up for obtaining the phase portraits:

$C = 0.15$ Gbps, $d_0 = 0.2$s, $T = 0.01$s and $N = 10$ flows

The initial conditions are chosen to be: $\gamma_{init} = \{0.5, 0.1, 0.01\}$ where γ is the normalized flow rate i.e. $\gamma = \frac{R}{C}$, and $q_{init} = \{2 \cdot C \cdot d_0, 0\}$. The trajectories are drawn for all combinations

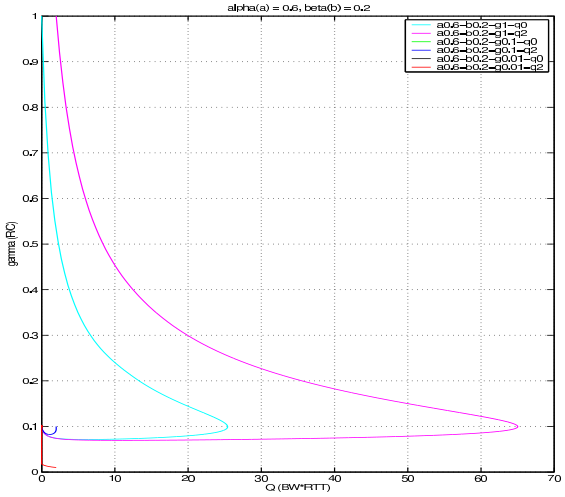

Fig. 41. Phase Portrait for $\alpha = 0.6$ and $\beta = 0.2$

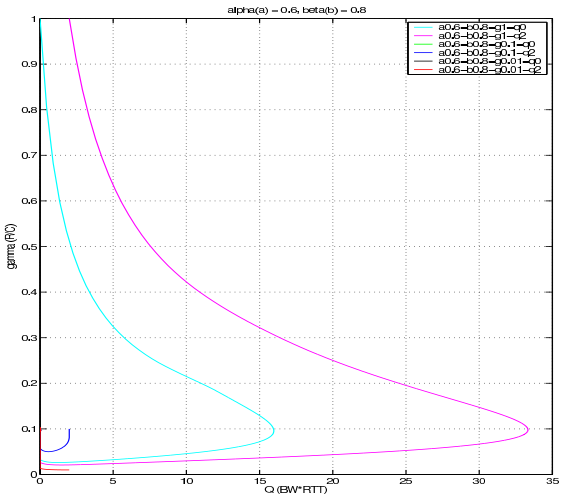

Fig. 42. Phase Portrait for $\alpha = 0.6$ and $\beta = 0.8$

of $(\gamma_{init}, q_{init})$. α and β vary from $(0.1, 2)$ in steps of 0.1
For each value of (α, β) we get a family of trajectories corresponding to each of the initial conditions chosen. Note that the equilibrium state of the system is $(\gamma, q) = (0.1, 0)$. So, for a given (α, β), if for any of the initial conditions the trajectory does not finally end at the equilibrium state we can conclude that this point does not lie in the stable region. Figures 41, 42 and 43 show some sample phase portraits. Figure 41 shows the phase portrait for $(\alpha, \beta) = (0.6, 0.2)$. This point is well within the linearized stability region and as seen in this Figure, all the trajectories converge to the equilibrium point. Figure 42 shows the phase portrait when $(\alpha, \beta) = (0.6, 0.8)$, which is a point outside the linearized stability region but within our hypothesized stable region. Here too all the trajectories converge to the equilibrium point. The top plot in Figure 43 shows phase portrait for $(\alpha, \beta) = (1.4, 0.8)$, a point well outside the stable region. The bottom plot in Figure 43 shows that for the initial condition $(\gamma_{init}, q_{init}) = (0.01, 2 \cdot C \cdot d_0)$, the trajectory never settles down at the equilibrium. It continuously keeps oscillating around the equilibrium point. The magnitude of these oscillations only grow larger as we get farther away from the stable region. Finally, Figure 44 shows the stabililty region obtained from the phase-plane anlysis. The stable regions obtained from the linearized analysis and simulations are also shown for the purposes of comparison. As can be seen, the region obtained from the phase-plane analysis matched well with that obtained via the simulations.

The stable region in Figure 44 holds true for varying C, d_0 and N. The range that we tried are:
$C = \{5.6Kbps, 0.15Gbps, 2.5Gbps, 10Gbps, 1000Gbps\}$
$d_0 = \{0.01, 0.5, 1, 2\}$
$N = \{10, 100, 1000, 5000\}$
Under all these varying conditions the stable region that we obtained remained the same, as shown in Figure 44.

A part of the stability region of the non-linear system has also been analytically derived using tools from non-linear control theory and is the subject of a separate paper [8].

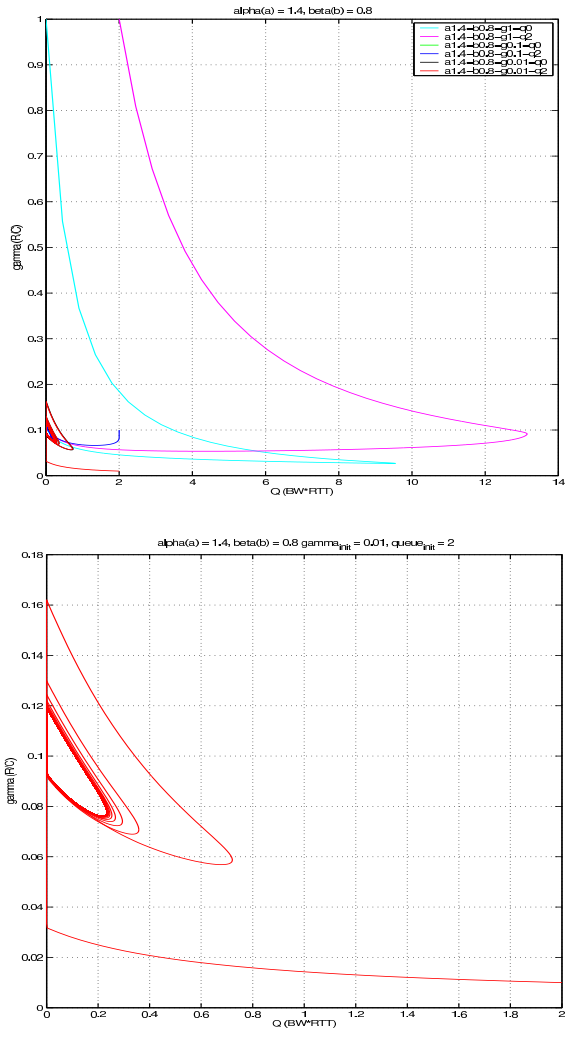

Fig. 43. (a) Top plot: Phase Portrait for $\alpha = 1.4$ and $\beta = 0.8$ (b) Bottom plot: Phase Portrait for $\alpha = 1.4, \beta = 0.8, \gamma_{init} = 0.01, queue_{init} = 2 \cdot C \cdot d_0$

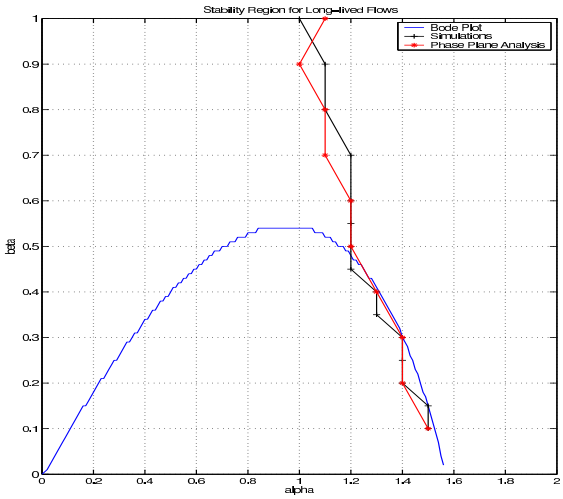

Fig. 44. Stable Region obtained from linearization model, Simulations and Phase Plane analysis

V. Related Work

The closest related work to RCP is the eXplicit Control Protocol (XCP) [2]; we have already seen the algorithmic difference between the two in section II-C.5 and the performance differences in Section III. Other related work includes:

• **Statistical Bandwidth Sharing:** There is a large effort toward understanding how bandwidth in the Internet should be shared in the presence of statistical traffic e.g. [1], [12], [13], [14] so as to achieve different performance goals. Our work is complementary in the sense that we are looking for simple, distributed algorithms that can possibly achieve these bandwidth sharing objectives under a broad range of network scenarios. This work has shown that we achieve insensitive bandwidth sharing for a single bottleneck case with a simple algorithm. We think that finding congestion control algorithms that can achieve similar goals for any general network topology is interesting future work.

• **ATM ABR and Algorithms to achieve Max-min fairness:** Explicit rate based feedback is definitely not novel to this paper, and has been proposed in the ATM ABR literature for example [15]. Although these algorithms usually try to achieve max-min fairness, they are very different from RCP. They either have a low storage and implementation complexity but do not achieve max-min fairness e.g. [16], [17], [18] or they achieve max-min fairness using per-flow state information e.g. [19], [21], [20]. In particular, we are not aware of any ATM ABR algorithm which has been demonstrated to emulate PS under a broad range of dynamic traffic and network conditions, and without per-flow state.

MaxNet [22] is a recently proposed mechanism for achieving max-min fairness. The links update the congestion signal sent to the sources as $d_l(t+1) = d_l(t) + \alpha_l(C_l - y_l(t))$. The sources adjust their rates based on the maximum of all feedbacks of links on their paths. Although the rates eventually converge to max-min fair rates, MaxNet relies on the fact that the flows last multiple round trip times. It is far from ideal in a dynamic environment where most flows could finish within a few RTTs. MaxNet also differs from RCP in that it does not explicitly try to drain the queue.

• **Rate feedback to TCP:** Jain et.al proposed QuickStart, which is an enhancement to TCP to allow an initial rate to be set, or an allowed initial congestion window, in the TCP SYN packet [23]. They propose that a TCP host indicate its desired sending rate in packets per second in the TCP SYN or SYN/ACK packet and each router in turn could either approve the specified rate or decrease it. Their proposal outlines the framework (which is essentially the same as the startup phase of RCP), but does not specify how the routers should pick this initial rate. Explicit rate feedback from the routers to TCP is also proposed in [24]. The rate feedback is mainly used to reduce TCP's window size and the usual modes in which TCP increases its rate – Slow Start and AIMD – still remain. As with TCP, the resulting performance won't be close to processor sharing.

• **Variations of TCP:** There are several proposals that change TCP at the end-hosts without any changes at the routers [27], [28], [29]. All of these address the problems with the Congestion Avoidance mode in TCP in high bandwidth-delay networks, when all the flows are long lived. While these schemes improve the performance of TCP-Reno in a static scenario with long-lived flows, there is no reason to think they will emulate PS under a wide range of dynamic network environments.

• **TCP with scheduling algorithms:** There have been proposals to differentiate short and long TCP flows at the router to improve the response times of short flows, by using schedulers at the routers that give preferential treatment to the short flows. While, we think this is a good approach, they do not however change slow-start or AIMD, which are the primary causes of long flow durations in TCP.

VI. 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 doesn't 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 won't 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. Ben 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] Dina Katabi, Mark Handley, and Charles Rohrs, "Internet Congestion Control for High Bandwidth-Delay Product Networks," In *Proceedings of ACM Sigcomm 2002*, Pittsburgh, August, 2002.

[3] Mark E. Crovella and Azer Bestavros, "Self Similarity in World Wide Web Traffic: Evidence and Possible Causes," In *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, December 1997.

[4] The Network Simulator, http://www.isi.edu/nsnam/ns/

[5] W. Wolff, "Stochastic Modeling and the Theory of Queues," PrenticeHall, 1989

[6] Curtis Villamizar and Cheng Song, "High Performance TCP in ANSNET," *ACM Computer Communication Review*, Vol.24, No.5, October 1994.

[7] *ns-2* code for Explicit Control Protocol, *http://ana.lcs.mit.edu/dina/XCP/*

[8] Hamsa Balakrishnan, Nandita Dukkipati, Nick McKeown and Claire 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.

[9] Vern Paxson and Sally Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," In *IEEE/ACM Transactions on Networking*, 3(3):226-44, June 1995.

[10] Steven H. Low, Fernando Paganini, Jiantao Wang, Sachin Adlakha and John C. Doyle, "Dynamics of TCP/RED and a Scalable Control," In *Proceedings of IEEE Infocom 2002*, New York, June 2002.

[11] C. V. Hollot, Vishal Misra, Donald Towsley and Weibo Gong, "Analysis and Design of Controllers for AQM Routers Supporting TCP Flows", In *IEEE Transactions on Automatic Control*, Vol. 47, No. 6, June 2002.

[12] L. Massoulie and J.W. Roberts, "Bandwidth Sharing and Admission Control for Elastic Traffic," In *Telecommunication Systems*, 15 (2000) 185-201.

[13] T. Bonald and A. Proutiere, "Insensitive Bandwidth Sharing in Data Networks," In *Queueing Systems*

[14] T. Bonald and L. Massoulie, "Impact of fairness on Internet Performance," In *Proceedings of ACM Sigmetrics*, 2001.

[15] Ambalavanar Arulambalam, Xiaoqiang Chen and Nirvan Ansari, "Allocating Fair Rates for Available Bit Rate Service in ATM Networks," In *IEEE Communications Magazine*, November 1996.

[16] L. Roberts, "Enhanced PRCA (Proportional Rate Control Algorithm)," In *ATM Forum cont. 94-0735R1*, August 1994.

[17] S. Muddu et.al., "Max-Min Rate Control Algorithm for Available Bit Rate Service in ATM Networks," In *Proc. ICC*, June 1996.

[18] F. Chiussi, Y. Xia, and V.P. Kumar, "Dynamic Max Rate Control Algorithm for Available Bit Rate Service in ATM Networks," In *Globecom* 1996.

[19] Anna Charny, "An Algorithm for Rate Allocation in a Packet Switching Network With Feedback", MS Thesis, MIT, April 1994.

[20] L. Kalampoukas, A. varma, and K. K. Ramakrishnan, "An Efficient Rate Allocation Algorithm for ATM Networks Providing Max-Min Fairness," In *Proc. 6th IAP Int'l Conf. High Performance Networking*, Sept 1995.

[21] Shivkumar Kalyanaraman, Raj Jain, Sonia Fahmy, Rohit Goyal, and Bobby Vandalore, "The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks," In *IEEE/ACM Transactions on Networking*, Vol. 8, No. 1, February 2000, pp. 87-98.

[22] Bartek Wydrowski and Moshe Zukerman, "MaxNet: A congestion control architecture for maxmin fairness", In *IEEE Communications Letters*, vol. 6, no. 11 , Nov. 2002, pp.512-514.

[23] Amit Jain and Sally Floyd, "Quick-Start for TCP and IP," In *Internet-draft draft-amit-quick-start-02.txt*, http://www.icir.org/floyd/quickstart.html, 2002.

[24] Aditya Karnik and Anurag Kumar, "Performance of TCP Congestion Control with Rate Feedback: Rate Adaptive TCP (RATCP)," In *IEEE Globecom 2000*, San Francisco, November 2000

[25] Eitan Altman, Tamer Basar and R. Srikant, "Robust Rate Control for ABR Sources" In *Proceedings of IEEE Infocom 1998*, San-Fransisco, March 1998.

[26] Orhan C. Imer, Sonia. Compans, Tamer. Basar and R. Srikant, "ABR Congestion Control in ATM Networks," In *IEEE Control Systems Magazine*, 21(1):38-56, Feb. 2001.

[27] Sally Floyd, "HighSpeed TCP for Large Congestion Windows," *RFC 3649*, December 2003

[28] Tom Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," In *Computer Communications Review*, vol. 32, no. 2, April 2003.

[29] Cheng Jin, David X. Wei and Steven H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," In *Proceedings of IEEE Infocom 2004*, Hong Kong, March 2004.

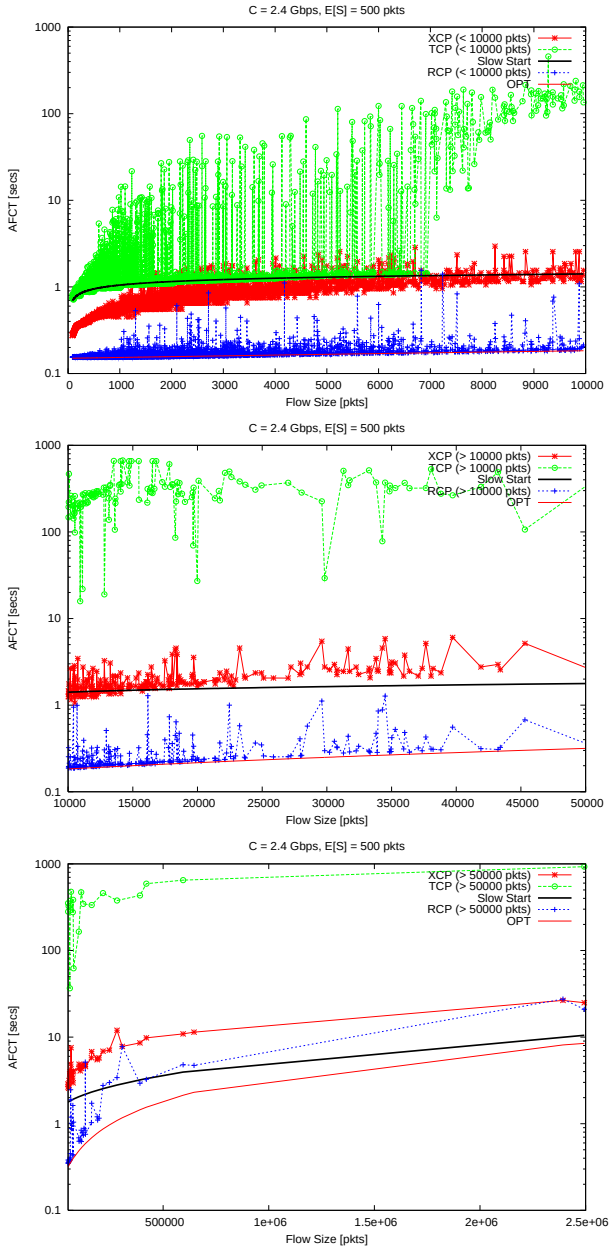

Fig. 45. AFCT of RCP, TCP and XCP under a single bottleneck C=2.4 Gb/s, RTPD = 0.1 s, $\rho = 0.4$, pareto distributed flows with shape = 1.2 and mean flow size = 500 pkts.

[30] Van Jacobson, "Modified TCP Congestion Avoidance Algorithm", Technical Report, April 1990.

[31] Rayadurgam Srikant, "The Mathematics of Internet Congestion Control", University of Illinois, Urbana, IL

APPENDIX I
SIMULATION EXAMPLES: AVERAGE FLOW COMPLETION TIME VS. FLOW SIZE

Two more examples comparing performance of TCP, XCP and RCP are shown in Figures 45,46.

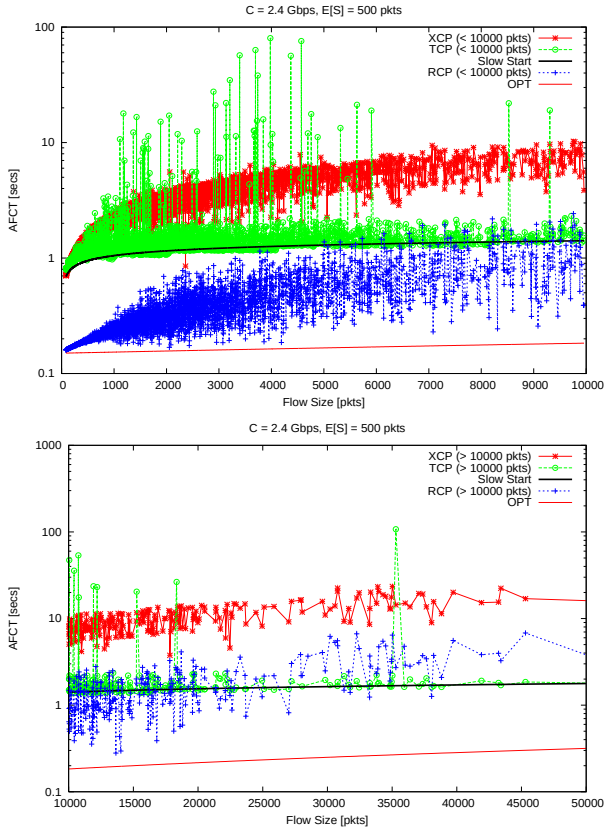

Fig. 46. AFCT of RCP, TCP and XCP under a single bottleneck C=2.4 Gb/s, RTPD = 0.1 s, $\rho = 0.8$, pareto distributed flows with shape = 1.2 and mean flow size = 500 pkts.

APPENDIX II
LINEARIZATION OF RCP RATE UPDATE EQUATION

In this Appendix we will linearize the RCP rate update Equation. The equations describing the system are:

$$\begin{aligned} \dot{q}(t) &= NR(t-d_0) - C \\ d(t) &= \frac{q(t)}{C} + d_0 \\ \dot{R}(t) &= R(t-T)\left(\frac{\alpha(C - NR(t-d_0)) - \beta\frac{q(t)}{d(t)}}{Cd(t)}\right) \end{aligned} \tag{17}$$

We define:

$$\begin{aligned} f(R_T, R_d, q) &\doteq R_T\left(1 + \frac{\alpha(C - NR_d) - \beta\frac{q}{d}}{Cd}\right) \\ g(R_d) &\doteq NR_d - C \end{aligned} \tag{18}$$

where $R_T \doteq R(t-T)$, $R_d \doteq R(t-d_0)$, $d \doteq d(t)$ and $q \doteq q(t)$. Recall that the equilibrium point is given by:

$$\dot{q}(t) = 0 \Rightarrow NR_e = C \Rightarrow R_e = \frac{C}{N}$$

$$\dot{R}(t) = 0 \Rightarrow R_e\left(\frac{\alpha(C - NR_e) - \beta\frac{q_e}{d_e}}{Cd_e}\right) = 0 \Rightarrow q_e = 0$$

From above, the equilibrium value of $d(t)$ is $d_e = d_0$. Evaluating partials of f and g at the equilibrium point $(R_e, q_e, d_e) = (\frac{C}{N}, 0, d_0)$ gives:

$$\begin{aligned} \frac{\partial f}{\partial R_d} &= -\frac{NR_T\alpha}{Cd}\Big|_{R_T=\frac{C}{N},\ d=d_0} \\ &= -\frac{\alpha}{d_0} \\ \frac{\partial f}{\partial R_T} &= -\frac{\alpha}{d} - \frac{N\alpha R_d}{Cd} - \frac{\beta q}{d^2 C}\Big|_{R_d=\frac{C}{N},\ d=d_0,\ q=0} \\ &= 0 \\ \frac{\partial f}{\partial q} &= -\frac{\beta R_T(\frac{q}{C} - d)}{C(\frac{q}{C} + d)^3}\Big|_{R_T=\frac{C}{N},\ q=0,\ d=d_0} \\ &= -\frac{\beta}{Nd_0^2} \\ \frac{\partial g}{\partial R_d} &= N \end{aligned}$$

The linearized equations are:

$$\begin{aligned} \delta\dot{R}(t) &= \frac{\partial f}{\partial R_d}\delta R(t-d_0) + \frac{\partial f}{\partial R_T}\delta R(t-T) + \frac{\partial f}{\partial q}\delta q(t) \\ &= -\frac{\alpha}{d_0}\delta R(t-d_0) - \frac{\beta}{Nd_0^2}\delta q(t) \\ \delta\dot{q}(t) &= \frac{\partial g}{\partial R_d}\delta R(t-d_0) \\ &= N\delta R(t-d_0) \end{aligned} \tag{19}$$

where

$$\begin{aligned} \delta R &\doteq R - R_e \\ \delta q &\doteq q - q_e \end{aligned} \tag{20}$$

APPENDIX III
BODE PLOT ANALYSIS

In this Appendix we will see why we need the condition $\frac{\alpha}{\beta} > 1$, in order for Equation (13) to have a non-zero solution. Recall that if Equation (13) has a solution, ω_c, then this is the frequency at which the phase plot of $G(s)$ crosses the $-\pi$ line. In other words, at ω_c we have:

$$\angle G(j\omega_c) = -\omega_c d_0 + \arctan\left(\frac{\omega_c \alpha d_0}{\beta}\right) - \pi = -\pi$$

Notice that $\angle G(j\omega) = -\pi$ at $\omega = 0$. And for large ω, $\angle G(j\omega)$ is much smaller than $-\pi$. So, unless $\angle G(j\omega)$ first increases and then decreases, as ω increases from 0, it will not cross the $-\pi$ line. Thus, the condition is that there should exist a maxima for $\angle G(j\omega)$. Differentiating $\angle G(j\omega)$ and setting it to 0 gives:

$$\frac{d}{d\omega}\angle G(j\omega_m) = -d_0 + \frac{\alpha d_0}{\beta}\frac{1}{1 + (\frac{\omega_m \alpha d_0}{\beta})^2} = 0$$

$$\Rightarrow \omega_m = \frac{\beta}{\alpha d_0}\sqrt{\frac{\alpha}{\beta} - 1}$$

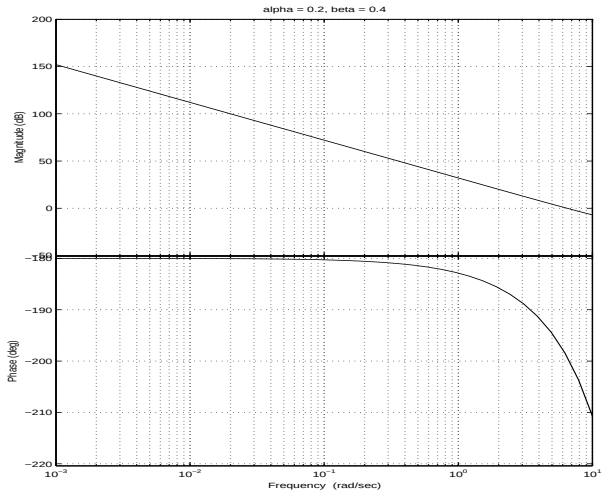

Fig. 47. $(\alpha, \beta) = (0.2, 0.4)$: ω_c does not exist

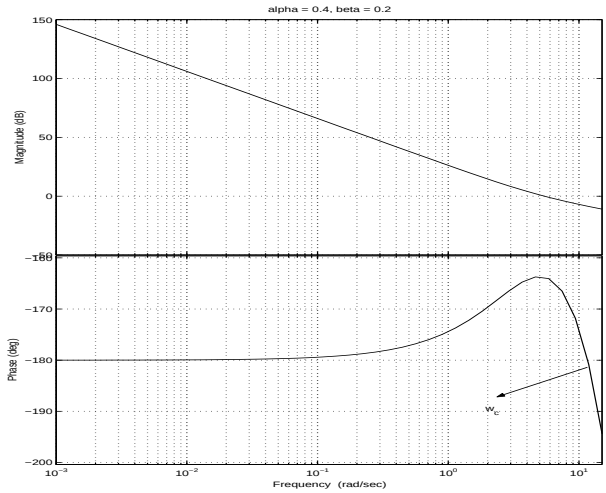

Fig. 48. $(\alpha, \beta) = (0.4, 0.2)$: ω_c exists

Obviously, the above maxima exists only if $\frac{\alpha}{\beta} > 1$
Thus, if the condition $\frac{\alpha}{\beta} > 1$ is satisfied then the the phase plot crosses the $-\pi$ line. Examples: the plot in Figure 47 shows a Bode Plot for $(\alpha, \beta) = (0.2, 0.4)$. Notice that the phase always decreases starting from $-\pi$, and never crosses the $-\pi$ line for any non-zero ω. Hence ω_c does not exist. Figure 48 shows a Bode Plot for the case $(\alpha, \beta) = (0.4, 0.2)$ and in this case, since $\frac{\alpha}{\beta} > 1$, ω_c exists.