Why Flow-Completion Time is the Right metric for Congestion Control and why this means we need new algorithms

Nandita Dukkipati, Nick McKeown
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9030, USA
{nanditad, nickm}@stanford.edu

***Abstract*— Users typically want their flows to complete as quickly as possible: They want a web-page to download quickly, or a file transfer to complete as rapidly as possible. In other words, Flow Completion Time (FCT) is an important - arguably the most important - performance metric for the user. Yet research on congestion control focuses almost entirely on maximizing flow throughput, link utilization and fairness, which matter more to the operator than the user. In this paper we show that existing (TCP Reno) and proposed (XCP) congestion control algorithms make flows last much longer than necessary - often one or two orders of magnitude longer than they need to; and we argue that over time, as the network bandwidth-delay product increases, the discrepancy will get worse. In contrast, we show how a new and practical algorithm - RCP (Rate Control Protocol) - enables flows to complete close to the minimum possible.**

***Index Terms*— Flow completion time, Congestion Control, Processor Sharing, TCP, eXplicit Control Protocol**

I. Introduction

When users download a web page, transfer a file, or send/read email, they want their transaction to complete in the shortest time; and therefore, they want the shortest possible *flow completion time* (FCT).[1] They care less about the throughput of the network, how efficiently the network is utilized, or the latency of individual packets; they just want their flow to complete as fast as possible. Today, most transactions are of this type, and it seems likely that a significant amount of traffic will be of this type in the future [1], [2]. So it is perhaps surprising that almost all work on congestion control focuses on metrics such as throughput, bottleneck utilization and fairness. While these metrics are interesting – particularly for the network operator – they are not necessarily of interest to the user; in fact, high throughput or efficient network utilization is not necessarily in the user's best interest. Certainly, as we will show, these metrics are not sufficient to ensure a quick FCT.

So why are congestion control algorithms not optimized to make flows finish quickly? The reason seems to be that it is easier to design an algorithm that efficiently utilizes the bottleneck link – and to prove that it does – than to prove that FCTs (or expected FCTs) have been minimized. Intuition might suggest that if the bottleneck link is highly utilized, then the rate of each flow is high and therefore flows will finish quickly. As we'll see, this intuition is wrong: For example, TCP will efficiently fill the bottleneck link, but flows frequently take over ten times longer to complete than they need to. In this paper, we provide evidence that existing and proposed congestion control algorithms do not come close to minimizing FCT or expected FCT in real scenarios — in fact, algorithms such as TCP and eXplicit Control Protocol (XCP) [3] have expected FCT one or two orders of magnitude larger than need-be.

In general, it is not possible to provably minimize the FCT for flows in a general network, even if their arrival times and durations are known [4] [5]. In a real network flows start and finish all the time, and different flows take different paths, and so minimizing FCT is intractable. But we believe - and it is the main argument of this paper - that instead of being deterred by the complexity of the problem, we should find algorithms that come close to minimizing FCTs, even if they are heuristic.

A well-known and simple method that comes close to minimizing FCT is for each router to use *processor-sharing* (PS) – a router divides outgoing link bandwidth equally among all the flows for which it currently has queued packets. If all packets are equal-sized, the router can maintain a queue for each flow, and simply round-robin among the non-empty queues, serving one packet at a time. If packets are not equal-sized, the router can use packetized processor sharing or fair queueing [6][7]. On a single link, it is known that Shortest Remaining Processing Time (SRPT) [8] minimizes expected FCT; and PS comes very close to SRPT in this case, even though PS doesn't need to know flow-sizes apriori.

On the face of it, TCP seems to approximate PS — if several infinitely long TCP flows share a bottleneck, the "sawtooth" pattern that TCP uses in congestion-avoidance mode will converge eventually on the max-min fair-share rate for each flow [9]. Similarly, XCP will converge on the fair-share rate for each flow by gradually (explicitly) assigning excess bandwidth to slow flows and reducing the rate of fast flows. But because they react over many round-trip times, neither TCP or XCP come close to processor-sharing when many flows are short-lived. TCP is characterized by the slow-start mechanism, in which a flow starts slowly and then – over several round-trip

[1]FCT = time from when the first packet of a flow is sent (in TCP, this is the SYN packet) until the last packet is received.

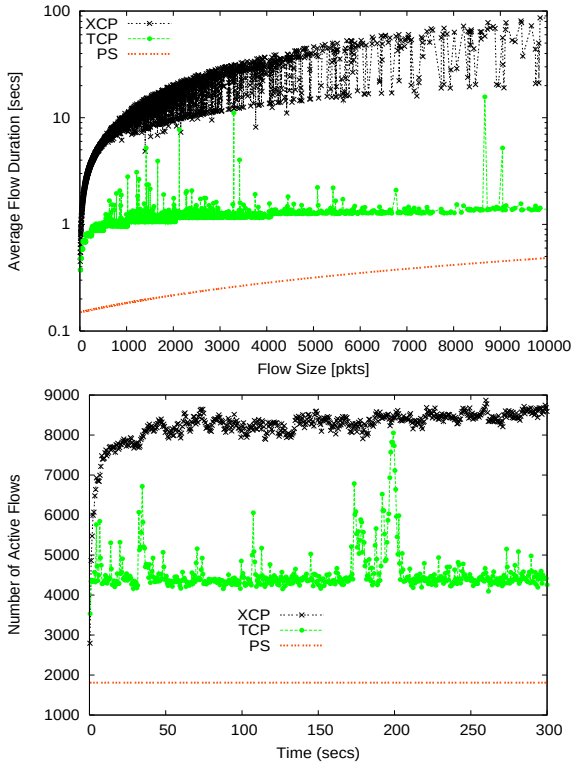

Fig. 1. The top plot shows the average flow duration versus flow size under TCP and XCP from a simulation with Poisson flow arrivals, flow sizes are Pareto distributed with mean = 30 pkts (1000 byte/pkt) and shape = 1.4, link-capacity = 2.4 Gbps, Round Trip Time = 100 ms, offered load = 0.9. The bottom plot shows the number of active flows versus time. In both plots the PS values are computed from analytical expressions [11].

times – increases its rate until it encounters a loss, then enters congestion-avoidance. Short-lived flows never leave slow-start and therefore operate below – often well below – their fair-share rate. Because of this, for a given network load, there are more active flows at any one time than there would be if they finished faster, which means the fair-share rate is lower and the FCT is larger.

A. Example

To illustrate how much longer flows take to complete with TCP and XCP, when compared to ideal PS, we used *ns-2* [10] to obtain the results shown in Figure 1. The simulation conditions (explained in the caption) were chosen to be representative of traffic over a backbone link today, and this graph is representative of hundreds of graphs we obtained for a variety of network conditions and traffic models. The values for PS are derived analytically, and show that flows would complete an order of magnitude faster than for TCP. 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. 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) [3]. XCP is designed to work well in networks with large per-flow bandwidth-delay product. 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.

II. Understanding Flow Completion Times in TCP and XCP

So why do TCP and XCP result in such long flow durations? In this section, we will try to explain why both mechanisms prolong flows unnecessarily. There seem to be four main reasons: (1) Flows start too slowly and are therefore artificially stretched over multiple round-trip times, (2) Bandwidth is allocated unfairly to some flows at the expense of others; either statically (e.g. TCP favors flows with short RTTs), or dynamically (XCP allocates excess bandwidth slowly to new flows), (3) Buffers are filled (TCP) and therefore delay all packets, and (4) Timeouts and retransmissions due to packet losses (TCP). We will examine each reason in turn, and use simple examples to clarify each factor.

A. Stretching flows to last many Round Trip Times (RTT) even if they are capable of finishing within one/few RTTs

Figure 2 shows an example with TCP and XCP; the top plot compares the mean flow durations, the middle plot shows the number of active flows with time and the bottom plot shows the link utilization.

1) TCP: In Figure 2, we can see that most flows never leave slow-start and therefore experience the same FCT as they would if TCP never entered the congestion-avoidance mode. In slow-start, the FCT for a flow of size L is $[\log_2(L+1)+1/2]\times RTT + L/C$ (excluding the queuing delay). Flows which experienced at least one packet drop in their lifetime will enter 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. Slow-start plus slow adaption by AIMD results in long flow durations. This is illustrated further by a simple deterministic example in Figure 3. In the example, the link capacity is 100 pkts/RTT. Two flows, each with size 50 pkts, arrive at the start of every RTT beginning from $t = 0$. In PS, both flows would complete in one RTT, equilibrium number of flows in system is 2 and the link utilization would be 100%. With TCP slow-start, the number of flows in the system evolves like in figure 3. The steady-state number of flows equals 12 which is six times higher than PS, consequently the flow duration is six times higher as well. Similar examples can be constructed with TCP flows in AIMD phase.

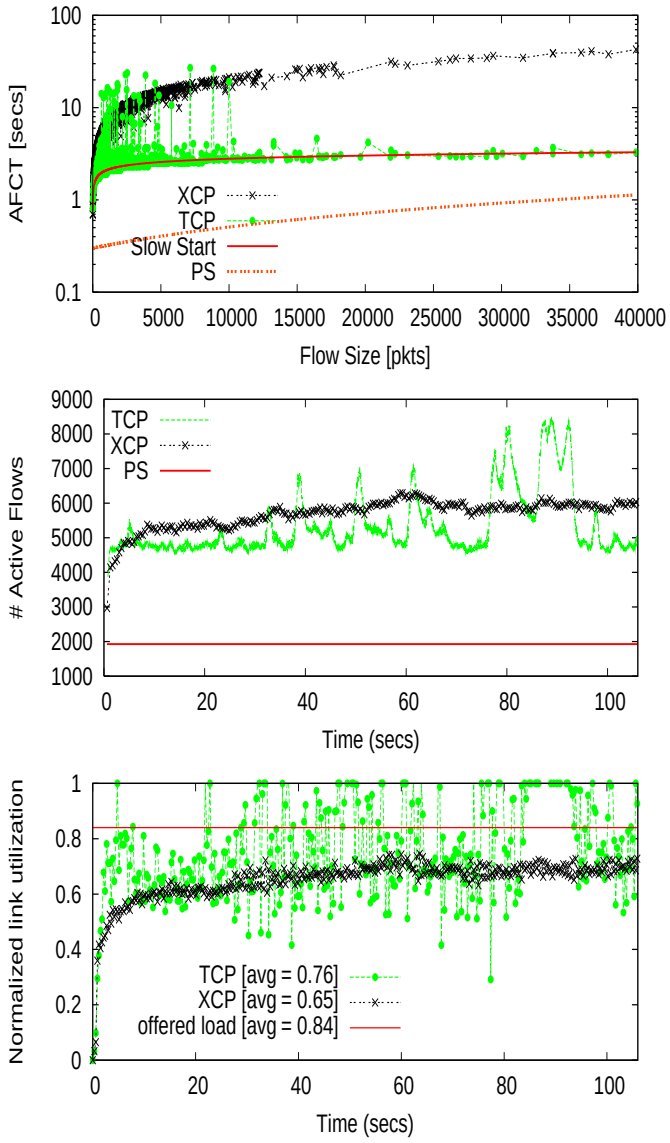

Fig. 2. The top plot shows the average flow completion time (AFCT) versus flow size under TCP and XCP from a simulation with Poisson flow arrivals, flow sizes are Pareto distributed with mean = 50 pkts (1000 byte/pkt) and shape = 1.2, link-capacity = 2.4 Gbps, Round Trip Time = 200 ms, offered load = 0.84. The middle plot shows the the number of active flows versus time. The bottom plot shows the link utilization for the two protocols measured over every 100 ms.

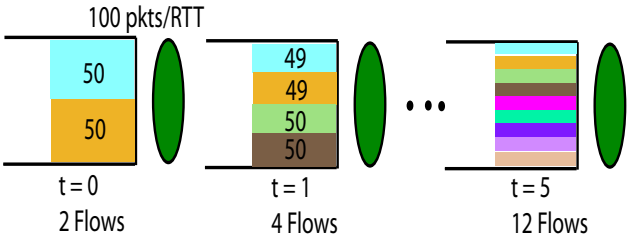

Fig. 3. An example illustrating how flows in TCP slow-start accumulate over time.

2) XCP: XCP is even more conservative in giving bandwidth to flows – particularly to new flows – which is why there are always more active, incomplete 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 RTTs, to avoid over-subscribing the link, and so keep buffer occupancy low. 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.

Flows in XCP can be prolonged further if flows are of different sizes. For example, suppose as in Figure 3, two flows arrive periodically at the start of every RTT, but with flow sizes 99 pkts (flow 1) and 1 pkt (flow 2). The flows only transmit the number of packets the server asks them to. At the start of every RTT, k, the server sends feedback[2] to each flow to send $(C * RTT)/N[k]$ amount of traffic in that RTT, where $C = 100$ pkts/RTT, $N[k]$ are the number of flows in the system at the start of RTT k, and $N[0] = 2$. This ensures that in every RTT there is no bandwidth over-subscription, and the link is being shared equally among flows. Although not identical, this is similar to what XCP would do. The server asks flow 1 to send 50 pkts/RTT and the same for flow 2. Flow 2 completes, but flow 1 lands up waiting more RTTs and competing for bandwidth with the newly arriving flows. At the start of second RTT, there are 3 flows so the server gives out $C/N(t) = 33.33$ pkts to each of 3 flows. The top plot of Figure 4 shows the evolution of the number of flows in the system. The number of flows keeps accumulating until it reaches a steady-state of 100 flows, at which point the fair share rate of every flow is 1 pkt/RTT. This is 50 times worse than the ideal steady state in PS. The system described above and the ideal PS both have the same long term link utilization of 100%, as shown in the bottom plot of Figure 4, and yet very different flow performance.

B. Unfair bandwidth sharing

Unfair bandwidth sharing is another reason why flow completion times could be high even if the link is well utilized. When different size flows share a link, the largest flow keeps the link busy while the other flows struggle to acquire their fair share, often finishing before getting their share. This is especially prominent in an Internet like scenario where flows have a heavy-tailed distribution. An example is shown in figure 5 where a long flow keeps the link occupied, and 3 new flows each of size 300 pkts start in an RTT. Ideally in PS, the 3 flows would finish in 1 RTT, but are made to last upto 8 times longer in XCP and TCP. In general, this effect is more prominent in XCP where if a long flow is hogging

[2]For sake of simplicity, assume that all the feedback delay is in the forward loop from the sender to the router, and so any feedback from the router to the sender in the reverse path is instantaneous.

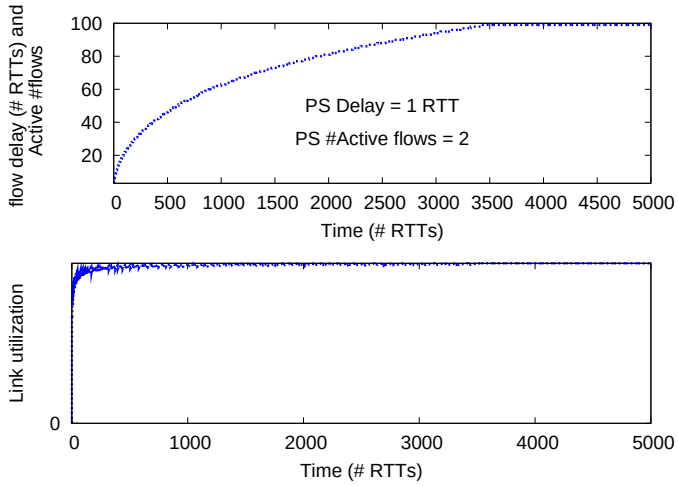

Fig. 4. An example illustrating how XCP flows accumulate over time.

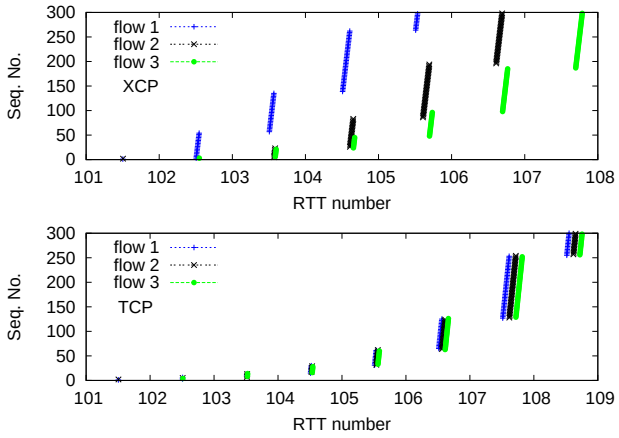

Fig. 5. Example illustrating unfair bandwidth sharing. A long flow is keeping the link occupied, 3 flows of size 300 pkts each, start in RTT number 100. Before TCP and XCP get a chance to take bandwidth away from the long flow and give these new flows their fair share, the flows are done. Link capacity = 100 Mbps, RTT = 0.1s. Under PS, the three flows would have finished in 1 RTT.

bandwidth, only 10% of link capacity is available for all newly arriving flows through *bandwidth shuffling* – a process where bandwidth is slowly reclaimed from the ongoing flow and distributed to new flows.

C. Filling up buffers

If RED is not used, TCP fills up any amount of buffering available at the bottleneck links. While this helps keep the link utilization high, it also increases the queuing delay, thus increasing the flow completion time. An example of TCP queue size is shown in figure 6. There are stretches for which the queue is consistently full. Flows arriving at these epochs experience large and variable delays. On the other hand the link utilization is close to the offered load.

D. Retransmissions and Timeouts

Flows experience losses when the buffer overflows. Eventually they are notified of the losses or time out and retransmit

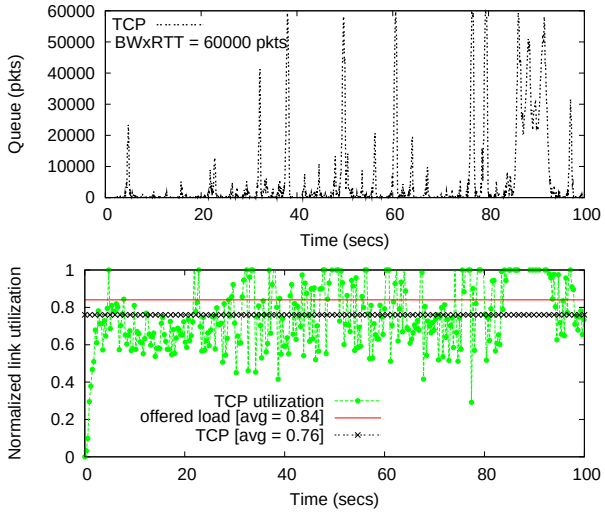

Fig. 6. The simulation set-up is the same as in figure 2. Top plot shows the instantaneous queue and the bottom plot shows the link utilization.

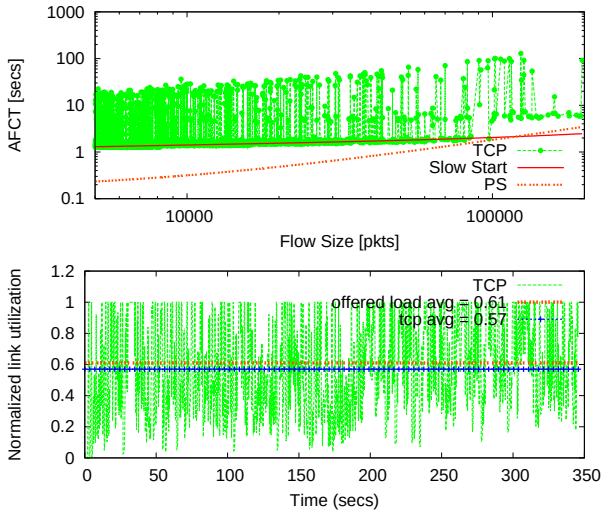

Fig. 7. Experiment to illustrate that link utilization (bottom plot) could be high due to retransmissions, but that does not mean much for Flow Completion Times (top plot). Set up: $C = 2.4$ Gbps, RTT = 0.1s, Poisson flow arrivals, offered load $= 0.8$, Pareto distributed flow sizes, Mean flow length $= 30000$ pkts. Link utilization is measured over 100 ms intervals.

the lost packets. Part of the link utilization is contributed by the retransmitted packets. On the other hand flow completion times could be long due to timeouts and the many additional RTTs the flows are made to last. Figure 7 shows an example with TCP flows in which retransmitted packets contribute to 3.6% of the link utilization. While the average link utilization (bottom plot) is close to that of the offered load, the flow durations (top plot) are two orders of magnitude higher than PS. Odlyzko gives an example of a real scenario [1] in which he notes that high utilization carries a penalty. During the hours of peak usage the average packet drop rate was around 5%, so service quality was poor, but the throughput figure was deceptively high since it included substantial retransmissions.

III. Modifying TCP's AIMD does not necessarily improve Flow Completion Times

There have been many proposals recently, such as High-Speed TCP [14] and Scalable TCP [15], to improve TCP's AIMD behavior. The main goal is to make TCP work well under low-multiplexed high bandwidth-delay networks. In particular, a single TCP flow should be able to achieve a large sustained equilibrium window size under realistic drop probabilities. For large window sizes, these schemes make the window increase more aggressive (as compared to additive increase of one packet per RTT) and the window decrease less drastic than halving on a packet drop. As we would expect, these schemes work well in the environment of a few high bandwidth flows. However, they do not necessarily improve flow completion times in the presence of a statistical mix of flows. For example, Figure 8 shows the flow completion times for a heavy-tailed mix of flow sizes for High Speed TCP (HSTCP) along with the regular TCP-Sack. HSTCP uses TCP's standard increase and decrease parameters for packet drop rates up to 0.0015 or roughly for window sizes up to 38 packets. Beyond that it uses a more aggressive increase and smaller decrease to achieve a different response function, for example a congestion window size of 83000 packets for a packet drop rate 10^{-7}. Flows in HSTCP take about the same time to finish as regular TCP flows. Figure 9 shows the queue occupancy with HSTCP and regular TCP. Both achieve similar link utilizations.

If RED or ECN is used, flow completion times are much longer as shown in Figure 10. There seem to be two main reasons for this: a) RED and ECN indicate the onset of congestion much sooner than Drop Tail, making flows exit slow-start early and therefore longer to finish. RED manages to keep queues very small, as shown in Figure 11, but at the cost of longer FCTs. Notice the flows which managed to complete in slow-start have relatively shorter FCTs as opposed to those which were pushed to congestion avoidance. b) In HSTCP, flows with large window sizes such as the ones with GB file sizes, increase their windows more aggressively and decrease less aggressively than flows with smaller windows, possibly resulting in a greater degree of unfairness than regular TCP.

It is hard to achieve low flow completion times by merely adapting TCP's AIMD. To finish flows close to processor sharing we will need a more radical solution like the one proposed in the next section.

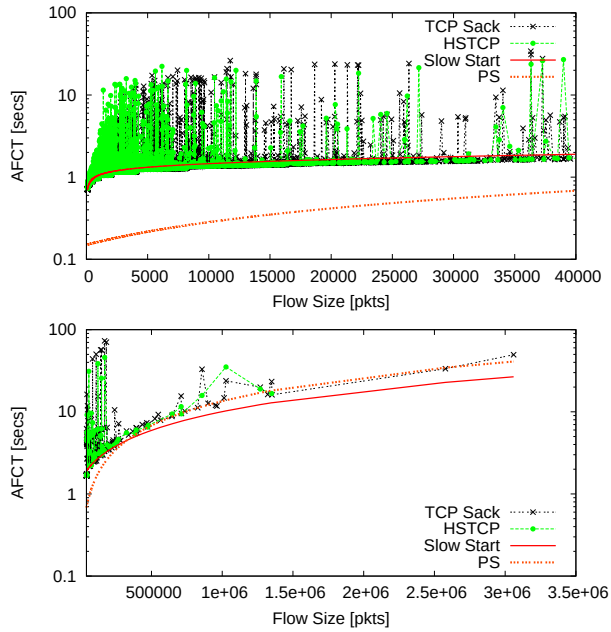

Fig. 8. Experiment comparing flow completion times in HSTCP and TCP Sack, both with Drop Tail queues. HSTCP (default) parameters: Low window = 38 packets, High Window = 83000 packets, High $P = 10^{-7}$. Set up: $C = 1$ Gbps, RTT = 0.1s, Poisson flow arrivals, offered load $= 0.4$, Pareto distributed flow sizes, Mean flow length $= 500$ pkts. Buffer size = bandwidth times RTT.

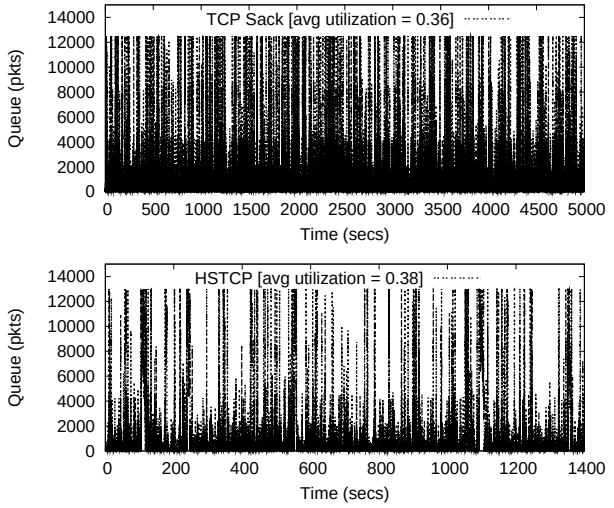

Fig. 9. Queue sizes for TCP Sack and HSTCP. Both achieve about the same link utilization.

IV. RCP: A simple congestion control algorithm with FCTs close to Processor Sharing

We recently described a simple congestion control algorithm called the Rate Control Protocol (RCP) that greatly reduces FCTs for a broad range for network and traffic characteristics [12]. RCP achieves this by explicitly emulating PS at each router.

A. Rate Control Protocol (RCP)

In RCP, a router assigns a single rate, $R(t)$, to all flows that pass through it; i.e. unlike XCP, it does not maintain and give a different rate to each flow. RCP is 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. It has three main characteristics that make it simple and practical:

1) The flow rate, $R(t)$, 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.

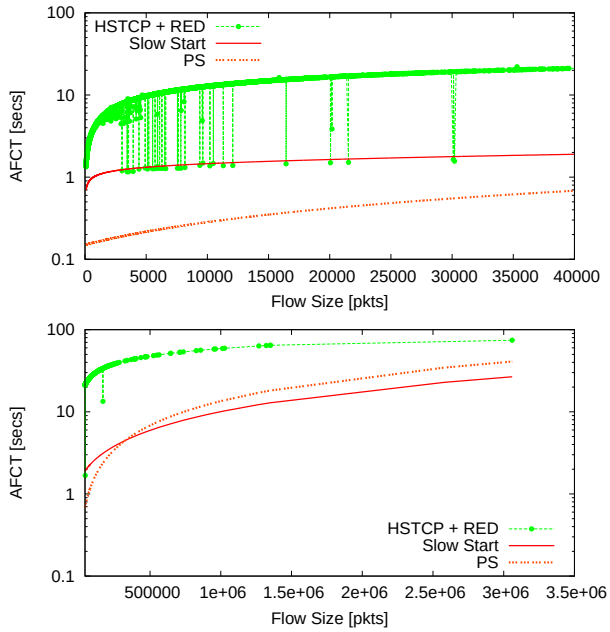

Fig. 10. Flow completion times in HSTCP with RED and ECN. HSTCP (default) parameters: Low window = 38 packets, High Window = 83000 packets, High $P = 10^{-7}$. RED is configured to (recommended settings [16]) gentle and adaptive with target delay as 0.005s. Set up: $C = 1$ Gbps, RTT = 0.1s, Poisson flow arrivals, offered load = 0.4, Pareto distributed flow sizes, Mean flow length = 500 pkts. Buffer size = bandwidth times RTT.

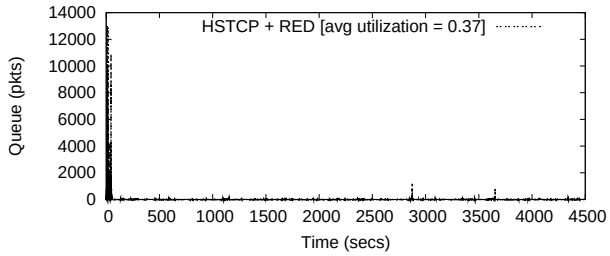

Fig. 11. Queue occupancy for HSTCP with RED and ECN. RED and ECN manage to keep a low queue occupancy. This however has an undesired effect on flow completion times.

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.

Note that the equation bears some similarity to the XCP control equation because both RCP and XCP are trying to emulate PS, but the manner in which they converge to PS are very different. The precise differences are elaborated in [12], [13]. The basic idea of equation (1) 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 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:

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

B. Example

Figure 12 shows the Average Flow Completion Time (AFCT) versus flow size for RCP, TCP, XCP and PS. 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

[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.

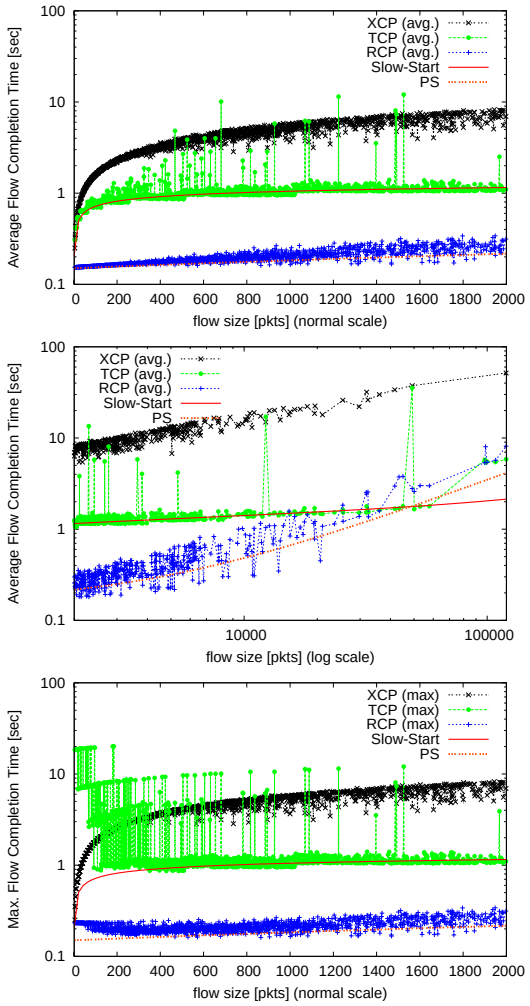

Fig. 12. Average Flow Completion Time (AFCT) for different flow sizes when $C = 2.4$ Gb/s, RTPD=0.1s, and $\rho = 0.9$. Flows are pareto distributed with $E[L] = 25$ pkts, shape = 1.2. The top plot shows the AFCT for flow sizes 0 to 2000 pkts; the middle plot shows the AFCT for flow sizes 2000 to 10^4 pkts; the bottom plot shows the maximum flow completion time among all flows of the particular size.

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

More simulations under different network topologies, conditions and traffic characteristics are discussed in [13].

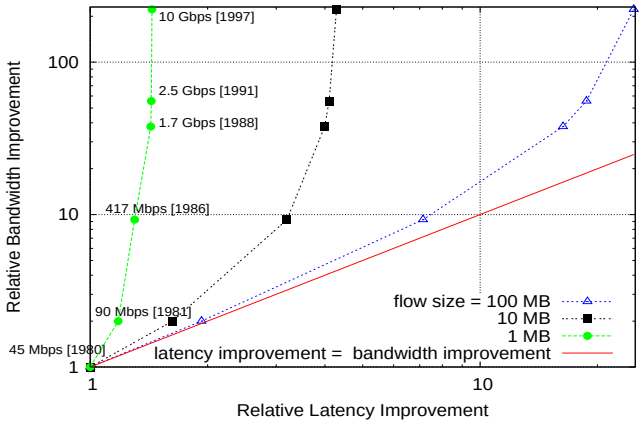

Fig. 13. Log-log plot of relative bandwidth and flow completion time improvements for $1, 10, 100MB$ flows – relative to the first milestone of 45 Mbps introduced in 1980. The time-line for the backbone link capacities used for this plot are from [17]. The optimistic flow completion time is computed assuming the flow has the entire link to itself and completes in slow-start phase. The round trip time used is 80 ms (comparable to RTT in U.S. backbone). Bandwidth in the network backbone improved by more than $100X$, while the flow completion time improved by less than $2X$ for a $1MB$ flow, less than $5X$ for a $10MB$ flow and less than $20X$ for a $100MB$ flow. The lagging improvement in flow completion times is primarily due to TCP's congestion control mechanism. This plot is inspired by a similar plot by Patterson [18], illustrating how latency lags bandwidth in computer systems.

V. Discussion: Flow completion times lag behind the increase in link bandwidths

As we've seen in several examples, increasing network bandwidth doesn't help a flow finish faster if the flow is limited by the number of RTTs it is made to last. Today, TCP makes the flows last multiple RTTs even if the flow is capable of completing within one RTT. While this was not a concern when link speeds were small and the transmission delay dominated flow duration, it is no longer the case, and this problem will only worsen with time: the relative improvements in bandwidth and FCT in wide area networks is shown in Figure 13 for different sized flows. The plot shows the most optimistic improvement in FCT, assuming that the flow has the entire link capacity to itself. And yet we see that bandwidth has improved by more than $100X$ while the flow completion time has lagged by one to two orders of magnitude. A part of it clearly is due to the fundamental limitation of propagation delay, but a significant portion of it is due to TCP's congestion control mechanisms. Also, notice in Figure 13 that as the link speeds increase, the percentage of flows for which there is a large disparity in link speed and flow completion times increases. Even if the flows are capable of completing within a RTT, TCP congestion control makes them last many RTTs, making it inefficient for the vast majority of the flows. While we cannot control the fact that the data must take at least one RTT, it is the premise of this paper that it is better to design congestion control algorithms to minimize the number of RTTs.

References

[1] A. M. Odlyzko, "The Internet and other networks: Utilization rates and their implications," In *Information Economics and Policy*, 12 (2000),

Pages 341-365.
[2] A. M. Odlyzko, “Internet TV: Implications for the long distance network,” In *Internet Television*, E. Noam, J. Groebel, and D. Gerbarg, eds., Lawrence Erlbaum Associates, 2003, pp. 9-18.
[3] D. Katabi, M. Handley, and C. Rohrs, “Internet Congestion Control for High Bandwidth-Delay Product Networks,” In *Proceedings of ACM Sigcomm 2002*, Pittsburgh, August, 2002.
[4] J. Du, J. Y. Leung, G. H. Young, ”Minimizing mean flow time with release time constraint”, In *Theoretical Computer Science archive*, Volume 75, Issue 3, October 1990.
[5] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker, ”Complexity of machine scheduling problems”, In *Annals of Discrete Mathematics*, Volume 1, Pages 343-362, 1977.
[6] A. K. Parekh, R. G. Gallager, ”A generalized processor sharing approach to flow control in integrated services networks: the single-node case,” In *IEEE/ACM Transactions on Networking*, Volume 1, Issue 3, June 1993
[7] A. Demers, S. Keshav, and S. Shenker, ”Analysis and simulation of a fair queueing algorithm,” In *In Proceedings of the ACM SIGCOMM*, Austin, TX, September 1989.
[8] L. Schrage, ”A proof of the optimality of the shortest remaining processing time discipline”, In *Operations Research*, Volume 16, Pages 687-690, 1968.
[9] 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.
[10] The Network Simulator, http://www.isi.edu/nsnam/ns/
[11] W. Wolff, “Stochastic Modeling and the Theory of Queues,” Prentice-Hall, 1989
[12] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown, “Processor Sharing Flows in the Internet,” In *Thirteenth International Workshop on Quality of Service (IWQoS)*, Passau, Germany, June 2005.
[13] Nandita Dukkipati, Nick McKeown, “Processor Sharing Flows in the Internet,” In *http://yuba.stanford.edu/rcp/*, Stanford HPNG Technical Report TR04-HPNG-061604.
[14] Sally Floyd, ”HighSpeed TCP for Large Congestion Windows,” In *http://www.icir.org/floyd/hstcp.html*, RFC 3649, December 2003
[15] Tom Kelly, ”Scalable TCP: improving performance in highspeed wide area networks,” In *A*CM SIGCOMM Computer Communication Review, Volume 33 , Issue 2, April 2003.
[16] Sally Floyd, Setting Parameters for RED, *http://www.icir.org/floyd/red.html*.
[17] D. Miller, Professor Electrical Engineering, Stanford University.
[18] D. A. Patterson, “Latency Lags Bandwidth,” In *Communications of the ACM*, Volume 47, Number 10 (2004), Pages 71-75.