Why Flow-Completion Time is the Right Metric for Congestion Control

Nandita Dukkipati
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9025
nanditad@stanford.edu

Nick McKeown
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9025
nickm@stanford.edu

ABSTRACT

Users typically want their flows to complete as quickly as possible. This makes Flow Completion Time (FCT) an important - arguably the most important - performance metric for the user. Yet research on congestion control focuses almost entirely on maximizing link throughput, utilization and fairness, which matter more to the operator than the user. In this paper we show that with typical Internet flow sizes, existing (TCP Reno) and newly proposed (XCP) congestion control algorithms make flows last much longer than necessary - often by one or two orders of magnitude. In contrast, we show how a new and practical algorithm - RCP (Rate Control Protocol) - enables flows to complete close to the minimum possible.

1. WHY WE SHOULD MAKE FLOWS COMPLETE QUICKLY

When users download a web page, transfer a file, send/read email, or involve the network in almost any interaction, 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 very interesting 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.

Intuition suggests that as network bandwidth increases flows should finish proportionally faster. For the current Internet, with TCP, this intuition is wrong. Figure 1 shows how improvements in link bandwidth have not reduced FCT by much in the Internet over the past 25 years. With a 100-fold increase in bandwidth, FCT has reduced by only 50% for typical downloads. While propagation delay will always place a lower bound, FCT is dominated by TCP's congestion control mechanisms which make flows last multiple RTTs even if a flow is capable of completing within one round-trip time (RTT).

[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.

[2]Real-time streaming of audio and video are the main exceptions, but they represent a tiny fraction of traffic.

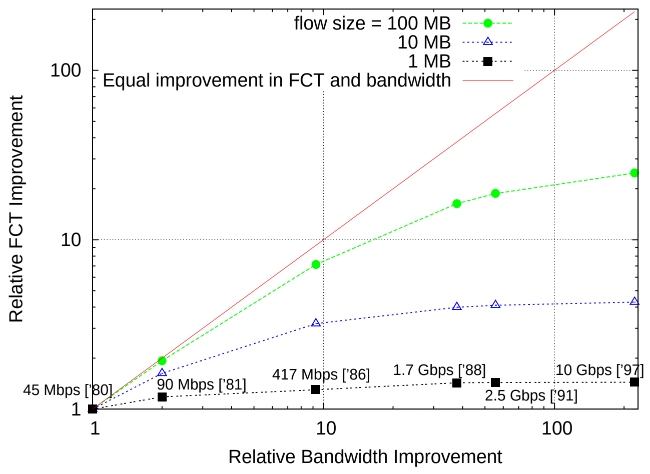

Figure 1: Improvement in flow-completion time as a function of link bandwidth for the Internet, normalized to 45 Mbps introduced in 1980. Flows have an RTT of 40 ms and complete in TCP slow-start. Plot inspired by Patterson's illustration of how latency lags bandwidth in computer systems [2].

So can we design congestion control algorithms that make flows finish quickly? Unfortunately, it is not usually possible to provably minimize the FCT for flows in a general network, even if their arrival times and durations are known [3]. Worse still, in a real network flows come and go unpredictably and different flows take different paths! It is intractable to minimize FCT. So instead congestion control algorithms are focussed on efficiently using bottleneck link (and only for long-lived flows) because this is easier to achieve. 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 ongoing flows.[3] On the face of it, TCP seems to approximate PS — if several infinitely long TCP flows with the same round-trip time share a bottleneck, TCP will eventually converge on a fair-share allocation. Similarly, eXplicit Control Protocol (XCP) [4][4] will converge on the fair-share allocation 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 nor XCP come close to processor-sharing for a mix of flow-sizes that represent the current usage of the Internet: both can have expected FCTs one or two orders of magnitude larger than need-be. Figure 2, obtained with *ns-2*, illustrates how much longer TCP and XCP flows take when compared to ideal PS. 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 [6]. The values for PS are derived analytically.

There are three main reasons why TCP and XCP lead to such long FCTs:

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

TCP: When a TCP flow starts, the source doesn't know the rate to use, and so it uses a small value to start with and ramps the rate over multiple RTTs. This is the well-known slow-start phase. Essentially, the source starts with a conservative rate, and forces the flow to last multiple RTTs. Hence, a typical flow today never leaves slow-start; and a flow of size L has a FCT given by $[\log_2(L+1)+1/2] \times RTT + L/C$ (excluding the queueing delay). For example with a typical flow size today of 15 packets, an RTT of 200 ms and a user connected via 1Mb/s link, TCP will force the flow to last 800 ms – about 4 times longer than the minimum possible. Over time, as bandwidth-delay products increase, the discrepancy will get worse. For example, in Figure 3 the link capacity is 100 packets/RTT. Two flows, each with size 50 packets, arrive at the start of every RTT beginning from $t = 0$. In PS, both flows would complete in one RTT, the 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 as shown in Figure 3. In steady-state there are 12 flows - six times higher than for PS; consequently the flow duration is six times higher than need-be.

The problem is not limited to slow-start. As Figure 2 shows even flows that have entered AIMD phase have long FCTs. This is because AIMD increases the window sizes even slowly.

XCP: XCP can be even more conservative in giving bandwidth to flows than TCP, particularly to new flows. This is why there are always more active, incomplete flows. XCP gradually reduces the window size of existing flows and increases the window size of new flows, making sure the bottleneck is never over-subscribed. It takes multiple RTTs for

[3]On a single link, it is known that Shortest Remaining Processing Time (SRPT) minimizes expected FCT; yet trying to emulate PS is a good goal to start with because: a) FCTs in PS come close to SRPT, b) unlike SRPT, PS does not need information on flow sizes – which is often unavailable to the routers, c) flow durations in PS are invariant of the flow size distribution and finally, but probably of less importance, d) PS is inherently fair among flows.

[4]XCP is designed to work well in networks with large per-flow bandwidth-delay product. The routers provide feedback - incremental window changes - to the sources over multiple round-trip times, which works well when all flows are long-lived.

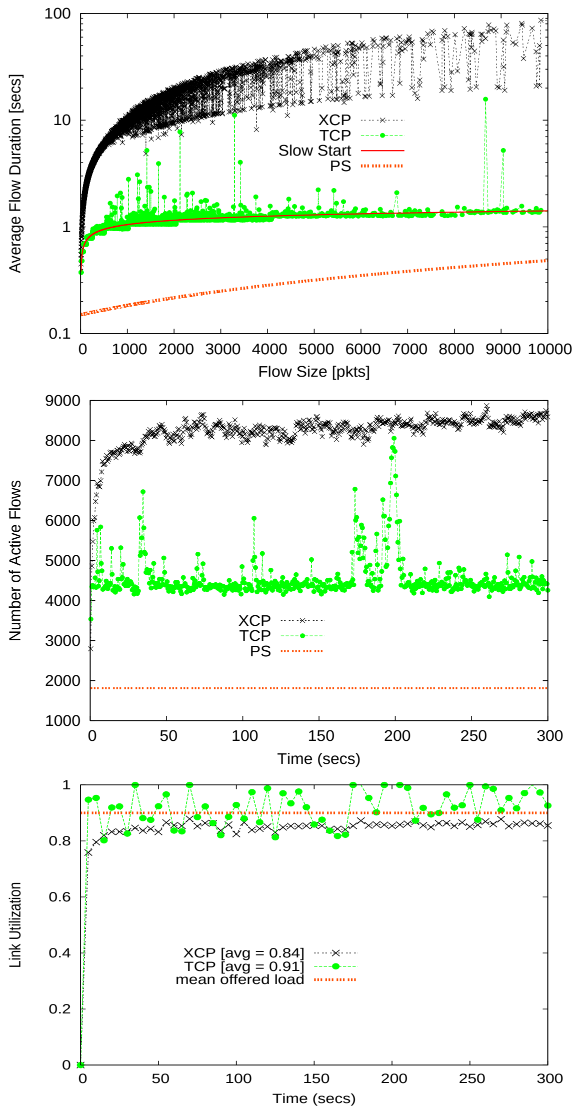

Figure 2: The top plot shows the average flow duration versus flow size under TCP and XCP from a simulation with Poisson flow arrivals, flow sizes are Pareto distributed with mean = 30 pkts (1000 byte/pkt) and shape = 1.4, link-capacity = 2.4 Gbps, Round Trip Time = 100 ms, offered load = 0.9. The middle plot shows the number of active flows versus time. In both plots the PS values are computed from analytical expressions. The bottom plot shows the offered load and the link utilization for the two protocols measured over 100 ms time intervals.

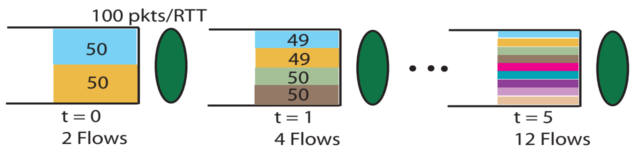

Figure 3: An example illustrating how flows in TCP slow-start accumulate over time.

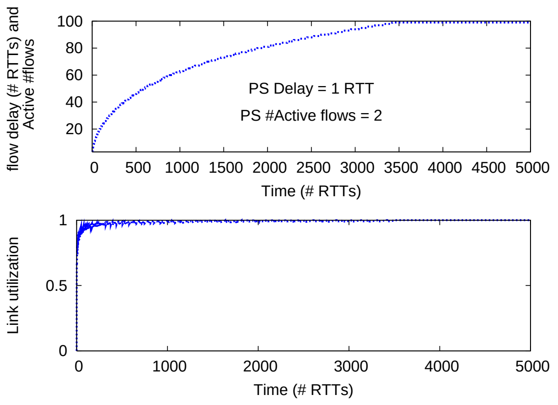

Figure 4: XCP flows can accumulate over time. Two flows arrive at the start of every RTT, with flow sizes 99 pkts (flow 1) and 1 pkt (flow 2); link of capacity = 100 packets/RTT. In the first RTT, the server asks both flows to send $\frac{C \times RTT}{2} =$ 50 packets. Flow 2 completes; flow 1 waits for more RTTs competing for bandwidth with newly arriving flows. At the start of second RTT, there are 3 flows so the server gives out $\frac{C \times RTT}{3} = 33.33$ packets to each of 3 flows. This continues until the system reaches a steady-state of 100 flows.

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 flows over multiple RTTs to avoid over-subscribing the bottleneck link. This prolongs flows and so the number of active/ongoing flows grows, which 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.

Every control interval, XCP carefully hands out spare capacity to ongoing flows. While this works well if all flows are long-lived, it is inefficient for a typical mix of flow-sizes. Short flows – that are about to finish – cannot use the bandwidth they are given while other flows which deserve more bandwidth land up waiting more RTTs for their share. The following example shows how XCP leads to long FCTs even when the link utilization is high. Consider two new flows starting every RTT, flow 1 = 99 packets, flow 2 = 1 packet and the link-capacity = 100 packets/RTT. In the k^{th} RTT each flow is told to send $(C \times RTT)/N(k)$ packets where $N(k)$ is the number of ongoing flows in RTT k [5]. Figure 4 shows that even though the link utilization is eventually 100%, the number of flows keep growing until there are 100 competing flows each sending one packet per RTT. The FCT in this system is 50 times worse than ideal PS, even while the link utilization in both is 100%.

2) Bandwidth hogging

Both TCP and XCP allow flows to hog the bottleneck link, which increases the FCT for other flows. TCP does this by favoring flows with short RTTs; in fact, flows with long RTTs can be made to have arbitrarily small rates (i.e. a large FCT) even when the bottleneck link is fully utilized. In addition, both TCP and XCP allocate excess bandwidth slowly to new flows. So when there is a mix of flow-sizes (for example, the heavy-tailed mix of flow-sizes in today's

[5]Although not identical, this is similar to what XCP would do.

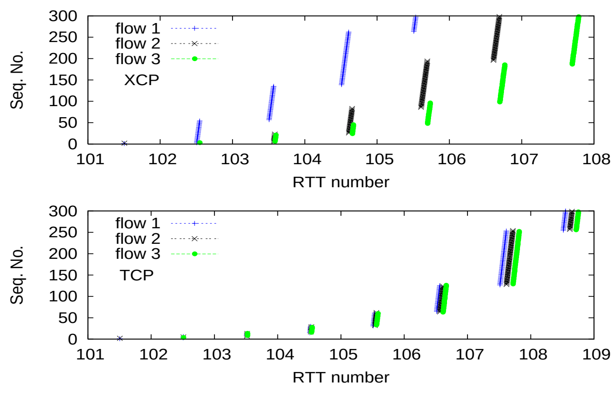

Figure 5: Example illustrating unfair bandwidth sharing. A long flow is keeping the link occupied, 3 flows of size 300 pkts each, start in RTT number 100. Before TCP and XCP get a chance to take bandwidth away from the long flow and give these new flows their fair share, the flows are done. Link capacity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

Internet), the long flows will converge on their fair share, while the short flows don't have time to reach their fair share before they finish. This is illustrated in Figure 5 in which a long flow keeps the link occupied, and three new flows start in the same RTT, each of size 300 packets. With PS, the three flows would finish in one RTT; but XCP and TCP make them last eight times longer.

The problem is in general more pronounced in XCP. When link is close to full utilization only 10% of link capacity is available for all newly arriving flows through bandwidth shuffling – bandwidth is slowly reclaimed from ongoing flows and distributed to new flows. Favoring early flows over new flows nudges XCP toward an M/G/1-FCFS discipline (for heavy-tailed jobs FCFS has the worst mean job-completion time among many known scheduling disciplines), instead of the more desirable M/G/1-PS.

3) Filling up buffers

TCP will try to fill the buffers at the bottleneck link - until a packet is dropped. While this leads to high utilization of the bottleneck link, it increases queueing delay, RTT and the FCT. Flows that arrive when queue occupancy is high will experience large and highly variable delays. In this regard, XCP is better than TCP because it always drives the buffer towards empty.

[7] gives evidence on why merely modifying TCP's AIMD will not ensure quick FCTs and using AQM schemes such as RED can lead to even worse FCTs than Drop Tail.

2. ACHIEVING SHORT FCTS

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 [5]. RCP achieves this by explicitly emulating PS at each router. In RCP, a router assigns a single rate, $R(t)$, to all flows that pass through it [6]; i.e. unlike XCP,

[6]Every packet header carries a rate field, R_p which is set to ∞ by the source. 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

it does not maintain 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: a) 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). b) Each router assigns a *single* rate for all flows passing through it. c) The router requires no per-flow state or per-packet calculations.

In the basic RCP algorithm, every router maintains a single fair-share rate, $R(t)$, that it offers to all flows. It updates $R(t)$ approximately once per RTT. 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. The following rate update equation is based on this intuition:

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

where d_0 is a moving average of the RTT measured across all packets, T is the update rate interval (i.e., how often $R(t)$ is updated) and is less than or equals d_0, $R(t-T)$ is the last updated rate, C is the link capacity, $y(t)$ is the measured input traffic rate during the last update interval, $q(t)$ is the instantaneous queue size, 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 there is a queue building up ($q(t) > 0$), then the link is oversubscribed and the flow rate is decreased evenly. 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 the number of ongoing flows, gives the change in traffic rate needed per flow. Built into Equation (1) is the routers estimate of the number of flows, $N(t)$, as $\hat{N}(t) = \frac{C}{R(t-T)}$. 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 [5].

Example: Figure 6 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 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. More simulations under different network topologies, conditions and traffic characteristics are discussed in [6].

We have seen in several examples throughout this paper that increasing network bandwidth doesn't help a flow finish faster if it is artificially forced to last several RTTs. This was not a concern when link speeds were small and the FCT was dominated by transmission delay. With increasing link speeds, the discrepancy will grow, and flows will not complete any faster.

destination copies R_p into the acknowledgment packets, so as to notify the source. The source transmits at rate R_p, which corresponds to the smallest offered rate along the path.

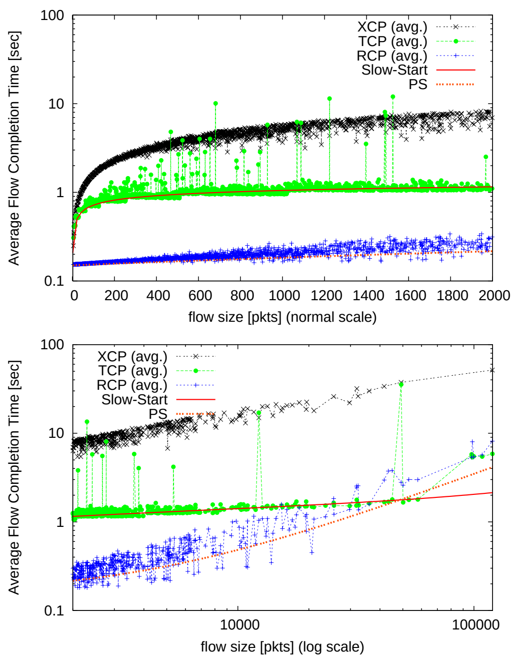

Figure 6: Average Flow Completion Time (AFCT) for different flow sizes when $C = 2.4$ Gb/s, RTPD=0.1s, and $\rho = 0.9$. Flows are pareto distributed with $E[L] = 25$ pkts, shape = 1.2. The top plot shows the AFCT for flow sizes 0 to 2000 pkts; the bottom plot shows the AFCT for flow sizes 2000 to 10^4 pkts.

3. 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] D. A. Patterson, "Latency Lags Bandwidth," In *Communications of the ACM*, Volume 47, Number 10 (2004), Pages 71-75.

[3] 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.

[4] D. Katabi, M. Handley, C. Rohrs, "Internet Congestion Control for High Bandwidth-Delay Product Networks," In *Proceedings of ACM Sigcomm 2002*, Pittsburgh, August, 2002.

[5] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, "Processor Sharing Flows in the Internet," In *Thirteenth International Workshop on Quality of Service (IWQoS)*, Passau, Germany, June 2005.

[6] N. Dukkipati, N. McKeown, "Processor Sharing Flows in the Internet," In *http://yuba.stanford.edu/tr.html*, Stanford HPNG Technical Report TR04-HPNG-061604, June 2004.

[7] N. Dukkipati, N. McKeown, "Why Flow-Completion Time is the Right metric for Congestion Control and why this means we need new algorithms," In *http://yuba.stanford.edu/tr.html*, Stanford HPNG Technical Report TR05-HPNG-112102, November 2005.