MAKING SHORT FLOWS FINISH FASTER WITH TCP

BY

SAMUEL D MUSSMANN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Assistant Professor P. Brighten Godfrey

Abstract

Users want web pages to load quickly. Because modern web pages make connections to many hosts, this requires that small flows complete quickly at high percentiles.

We explore how to achieve this goal with protocols that do not require modifications to all the routers and agents transmitting the flow. Our simulations indicate that both Random Early Detection and Fair Queuing can significantly reduce flow completion times at both the median and the 99^{th} percentile. Fair Queueing provides more consistent reductions across varying bandwidth-delay products and background traffic.

Table of Contents

Chapter 1

Introduction

Users want web pages to load faster. When Google search results were artificially delayed by 400 milliseconds, users performed 0.74% fewer searches after 4-6 weeks[3]. When Bing delayed search results by 500 milliseconds, they lost 1.2% of revenue per user[18]. A user study from 2000[11] indicated that users perceive the information on a website (with text and graphics) as being of higher quality when it is delivered 3000 milliseconds more quickly.

However, when we look at the transport layer underlying web page loading, it's not just average latency that counts – outliers matter too. According to a Google survey of the top 380 million web sites[17], a page on one of these sites requests resources from a median of 7 hosts. Under the simplifying assumption that each of these requests is about the same size, page load time is controlled by the request which takes the longest. Since the probability that no request takes a time in its 90^{th} percentile is $0.9^7 = 0.48$, the median page loading time is controlled by the 90^{th} percentile flow completion time at the transport layer.

Dukkipati et al. propose Flow Completion Time (FCT) in [5] as a measure that more closely correlates with the results users see. By reducing FCT, we reduce the latency that users see. Dukkipati et al. also propose Rate Control Protocol (RCP)[4] as a complete replacement for TCP that drastically reduces average FCT, especially in high-bandwidth long-delay networks. However, this work pays equal attention to long and short flows while Web browing disproportionately involves small files. According to the same Google survey of the most popular 380 million sites on the Internet[17], the median number of KB requested per host for a page is 12 KB and the 90^{th} percentile is 133 KB.

In this paper, we investigate the gap in FCT between TCP and RCP for short flows, and we attempt to close that gap using local changes – protocols that don't require changes to the whole Internet to be effective. Since outliers matter, we pay attention not only to the median FCT, but also to the 99^{th} percentile of FCT.

We observe from our simulations that TCP Slow Start imposes a fundamental lower bound to FCTs for short flows. With small bandwidth-delay products, this lower bound is close to RCP and so we can make TCP flows finish comparably quickly to RCP using Random Early Detection (RED) and Fair Queueing. With larger bandwidth delay products, the gain is less and only Fair Queuing reduces FCT at the 99^{th}

percentile. With this higher bandwidth-delay product, we recover 50% of the gap between TCP and RCP at the median, and 75% at the 99^{th} percentile.

Chapter 2

Related Work

Related work falls into two categories – new protocols to replace TCP and improvements of TCP – but no work has been done on local network changes and their impact on flow completion time for short flows.

One cannot talk about FCT without making reference to Dukkipati et al.[5][4] for the concept of FCT and for RCP, a TCP replacement designed to decrease FCT by emulating processor sharing. RCP proves excellent at minimizing FCT, but requires a new protocol stack, with both end host and router modifications required.

Quick Start for TCP and IP [9] uses IP options to ask routers along the path whether they have spare capacity, and it increases the initial window if that spare capacity exists. However, like RCP, it requires end host and router modifications.

XCP[12] is an older protocol in the space of TCP replacements. XCP works by separating the notions of congestion control and fairness and explicitly allocating bandwidth to flows. However, XCP requires modifications at the router, per-packet calculations, and end host support.

A family of protocols have followed in XCP's footsteps: VCP[20], MLCP[15], and BMCC[16] all borrow XCP's notion of separation between congestion control and fairness. They differ in that they are evolutions of TCP – they use TCP's ECN bits or a few bits somewhere else in the TCP header to communicate congestion levels in the network. Then, based on those congestion levels, they adjust the size of the sender's window. All of these protocols require modifications to all routers along transmission paths as well as end host support.

All of these protocols make flows complete faster, but they also all require sweeping changes in the network – a difficult proposition.

On the other hand, there is significant research on making TCP flows faster in high bandwidth-delay product networks. Compound TCP[19], High Speed TCP[7], CUBIC TCP[10], and Scalable TCP[13] are notable examples of this research. However, these examples (as well as the vast majority of similar work) focus on TCP's Congestion Avoidance stage, which most short flows do not enter.

There has been some work done on increasing the throughput in TCP's Slow Start stage, however.

Google[6] found that increasing the initial window to at least 10 segments can decrease web object download latency by 10%. FLD_TCP[14] also increases the speed of TCP Slow Start (responding with three packets to every ACK received rather than two), but is not thoroughly tested. All of this work focuses solely on the end host and not the network.

Work has also been done on sizing buffers to complete flows faster (an in-network change), but Appenzeller et al.[1] found that buffer sizing has minimal effect on flows that do not leave Slow Start.

Chapter 3

A Theoretical Note

Our experiments will show how much of the gap between RCP and TCP we can empirically close. It is also interesting to consider how much of that gap it is possible for us to close. Put differently, how does TCP provide a lower limit to FCT for small files?

Small files are limited by TCP in two ways. First, TCP Slow Start fundamentally limits the transmission time of any file to at least $(1 + \log(\frac{size}{initialwindow})) * RTT$. This is particularly harmful for small files – with an initial window of 2 and packet size of 1KB, a 12KB flow will take 3 round trips to finish.

Second, a single drop on a congested link will force a flow out of Slow Start and into Congestion Avoidance. While this tends to happen less frequently for smaller flows (due to the smaller number of packets), when it does, that flow is stuck with a particularly small window that will only grow additively.

Thus, in order to reduce the FCT of small flows, we must work to keep these flows in Slow Start for as long as possible, and we must make Slow Start as fast as possible.

Chapter 4

Simulation Setup

4.1 Simulator

All results in this paper are from NS-2[2] simulations. We use version 2.34 with two patches. The first is the RCP patch distributed by Dukkipati et al. from `http://yuba.stanford.edu/rcp/`, which adds RCP agents and queues. The second is a patch we developed for a more robust fair queueing implementation. The default fair queueing implementation limits the number of flows in the simulation to 32 and has runtime linear in the number of maximum allowed flows.

Our implementation deals with an unbounded number of flows by distributing them across 1000 queues. It has runtime logarithmic in the number of queues that currently contain packets. In addition, we bound the total number of packets queued by the queue size limit – if a packet is enqueued while the queue is full, the last packet in the longest queue is dropped. In contrast, the default implementation allows each flow to have up to the queue size limit of packets enqueued.

4.2 Topology

For our primary test we use a half-dumbell network – each flow has its own link connecting it to the source side of the bottleneck link, but there is only a single node on the destination side of the bottleneck. When we set a simulation to a certain bandwidth, we set all links to use that bandwidth. When we set a simulation to use a certain latency, we set the links connecting the flows to the dumbell to have that latency. The bottleneck link always has 10ms latency unless otherwise mentioned. Thus, the RTT is two times the mentioned latency plus 20ms.

For a run of our simulation with n background flows, we create $n + 1$ flows. n flows start sending flows of some size at about time 0. When one of these flows finishes sending a flow, it simply restarts, thus maintaining a steady flow of traffic. After 250 RTTs and a random offset, the measuring flow is started. We record for each packet in the measuring flow the time when that packet and all preceding packets have

arrived at the destination. This time for the kth packet is considered to be the FCT for a flow of length kKB, since we configure all packets to have size 1KB.

4.3 Protocols under Test

In our tests we use a number of protocols. Each protocol controls the transport protocol as well as the queueing discipline at the routers.

In order to have a reference point for our changes, we test a baseline "standard" TCP. For this purpose, we use TCP Reno with an initial window of 2 packets and a droptail queueing discipline. Similarly, to have a reference point for the amount of possible improvement, we test RCP as a target. We set $\alpha = 1$ and $\beta = 1$.

We then test three protocols that improve on TCP in FCT to see how much they improve. The first is a standard TCP with a larger initial window. We use TCP Reno with droptail queueing and an initial congestion window of 10, in line with Google's recommendation[6]. The second is RED, which we choose as a fairly standard, well studied AQM technique. We use TCP Reno with an initial congestion window of 10. We allow RED to use ns-2's automatic initialization, as recommended by Sally Floyd[8]. The third is Fair Queueing, for which we use TCP Reno with an initial congestion window of 10.

Chapter 5

Results

For our primary result, we use a simulation with 10 Mb/s links, 10ms latency, and 4 background flows of infinite size. This situation was chosen in order to measure the performance of a mouse flow (the measuring flow) against elephant flows. In other sections we will observe the behavior of mice flows against a Pareto-distributed background (§5.1) and the behavior of mice flows against mice flows (§5.2).

In addition, we choose a rather small bandwidth-delay product. We do this in order to show that if Slow Start is competitive with RCP, then Fair Queueing and RED can make TCP competitive with RCP, even at high percentiles. More broadly, if flows can start quickly, then TCP can be made competitive with RCP.

The results of this simulation are in Figure 5.1. We show the median and 99th percentile flow completion times, as well as CDFs of flow completion time at 12 and 133 packets. We choose 12 and 133 packets because those are the median and 90th percentile number of KB per host in the top 380 million sites on the web[17].

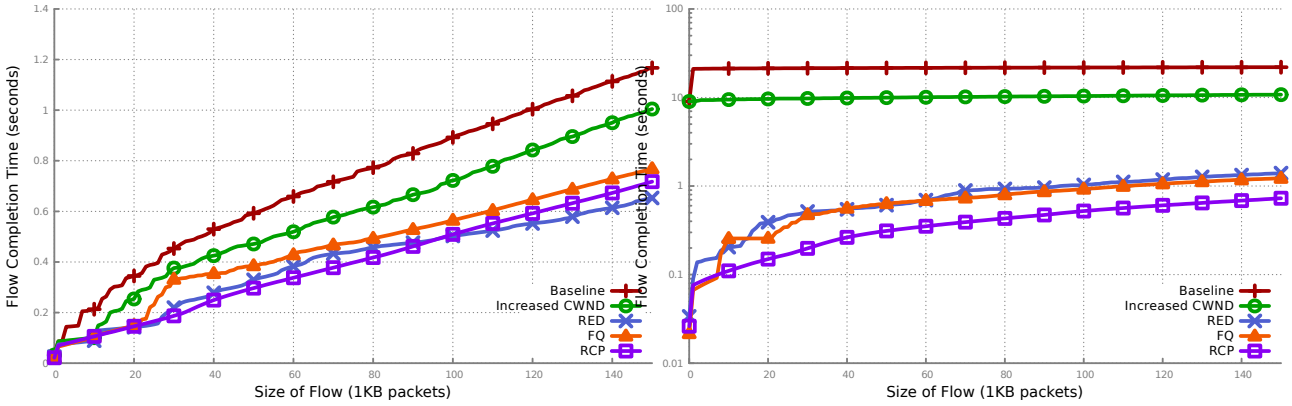
We observe in these figures that both RED and Fair Queueing provide significantly more consistency at small flow sizes. In RED's case, we posit that this comes from reducing queue occupancy and thus increasing the probability that none of the first few packets of smaller flows get dropped. However, at the larger flow size, RED becomes more inconsistent because, even with the reduced queue occupancy, a significant number of packets have still likely been dropped.

Fair Queueing on the other hand is very consistent because it treats each flow in isolation. As long as the measuring flow has fewer packets in the queue than any other flow, none of its packets will be dropped.

5.1 RCP Comparison

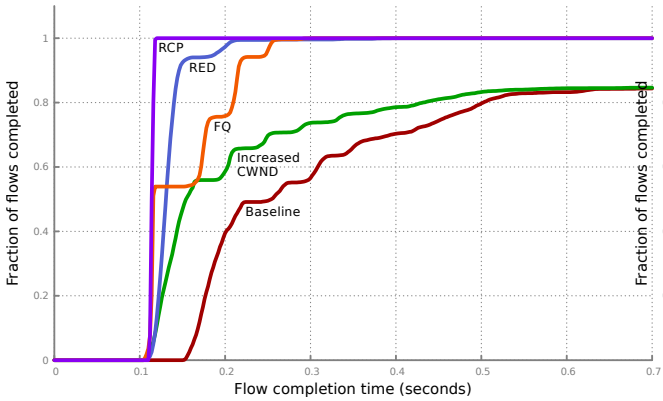
In this section, we test RCP on its home turf, matching the simulation in figure 6 of [5]. However, we test a different mix of protocols and divide between file sizes slightly differently.

We test the five protocols from §4.3. This adds RED and Fair Queuing to the protocols used in [5] and removes XCP. We remove XCP because it is previously tested, and because it is not relevant to our inquiry, as argued earlier.

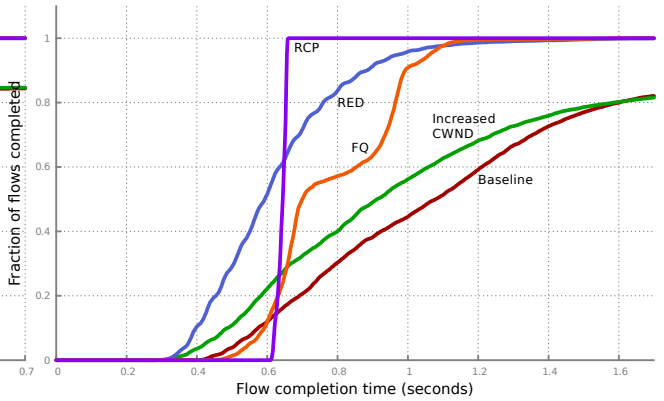


(a) Median flow completion times

(b) 99th percentile flow completion times (log scale)



(c) CDF of flow completion times for 12 packet flows



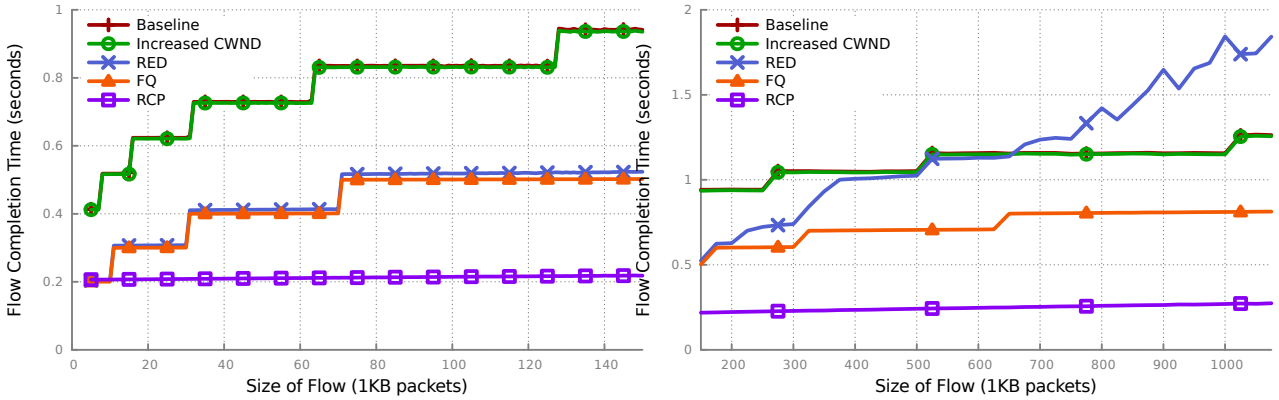
(d) CDF of flow completion times for 133 packet flows

Figure 5.1: Flow completion times for a simulation with 10 Mb/s links and 10ms latency (40ms RTT).

The simulation setup is a single 2.4 Gb/s link with 100ms RTT. Flows are offered at Poisson-distributed arrival times and Pareto-distributed sizes in such a way as to use 90% of the link's capacity. The mean of the size distribution is 25 packets and the shape is 1.2.

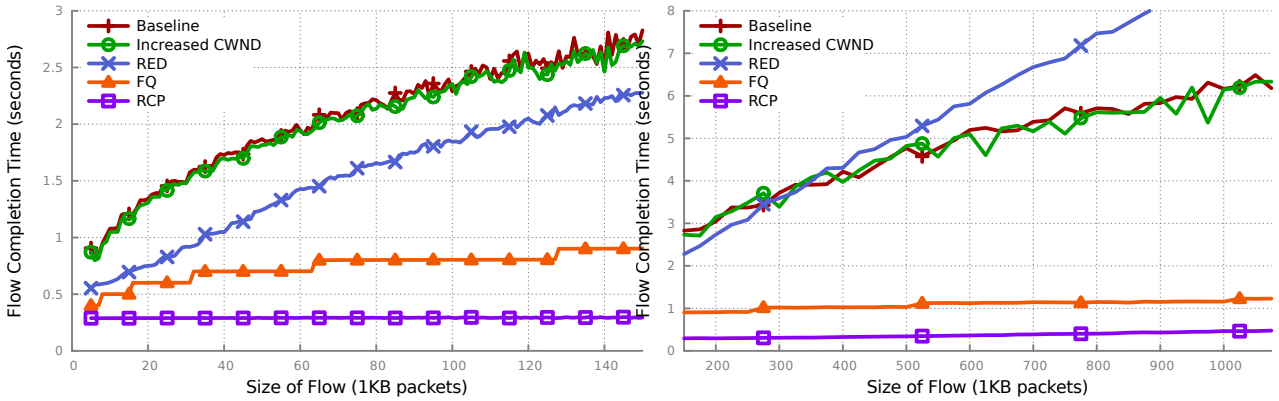
In addition, in order to have more statistical certainty at the 99th percentile, we would like to have at least 5000 flows at larger flow sizes. With 10^8 flows in the simulation, we expect to get 5000 flows for all lengths up to 213 packets long. While this is where most web traffic exists, we would also like to have an accurate idea of performance at higher flow lengths, so we modify 1 in 500 flows to have a length uniformly distributed over a set of marker lengths (we use every 25^{th} packet from 100 to 1075). The mean of this measuring distribution is 587.5, so it only affects the mean of the overall distribution by 1.175 (587.5/500).

Figure 5.2 shows the results of the simulation at both the median and the 99^{th} percentile. Note that each data point on the graph is generated from at least 5000 flows of that length in our simulation. We can clearly observe the fundamental limit that slow start imposes – any TCP that doesn't change slow start cannot be faster than that.



(a) Median flow completion times for small flows

(b) Median flow completion times for larger flows



(c) 99th percentile flow completion times for smaller flows

(d) 99th percentile flow completion times for larger flows

Figure 5.2: Flow completion times for a simulation with 2.4 Gb/s links and 100ms RTT. Flow arrivals are Poisson distributed so as to offer link utilization of 0.9. Flow sizes are Pareto distributed (except for 0.2% measuring flows).

These figures also indicate that we can help TCP flows stay much closer to that theoretical limit. RED helps small flows stay near that limit at the median, although with larger flows the median drifts higher. In addition, at the 99$^{\text{th}}$ percentile, RED behaves roughly equivalently to standard TCP.

Fair Queueing is the clear winner here. At the median, Fair Queuing makes up half the gap between RCP and TCP. However, at the 99$^{\text{th}}$ percentile, Fair Queueing grows more slowly than TCP. At 13 and at 133 packets, Fair Queueing makes up 75% of the difference between TCP and RCP. At 500 packets, Fair Queueing makes up 85% of the difference, and at 1000 packets, it makes up 88% of the difference.

5.2 Mouse versus Mouse

In the primary simulation we used single background flows of infinite size. Here, we instead have some number of background flow generators that restart flows every 32 packets. This gives us a competition

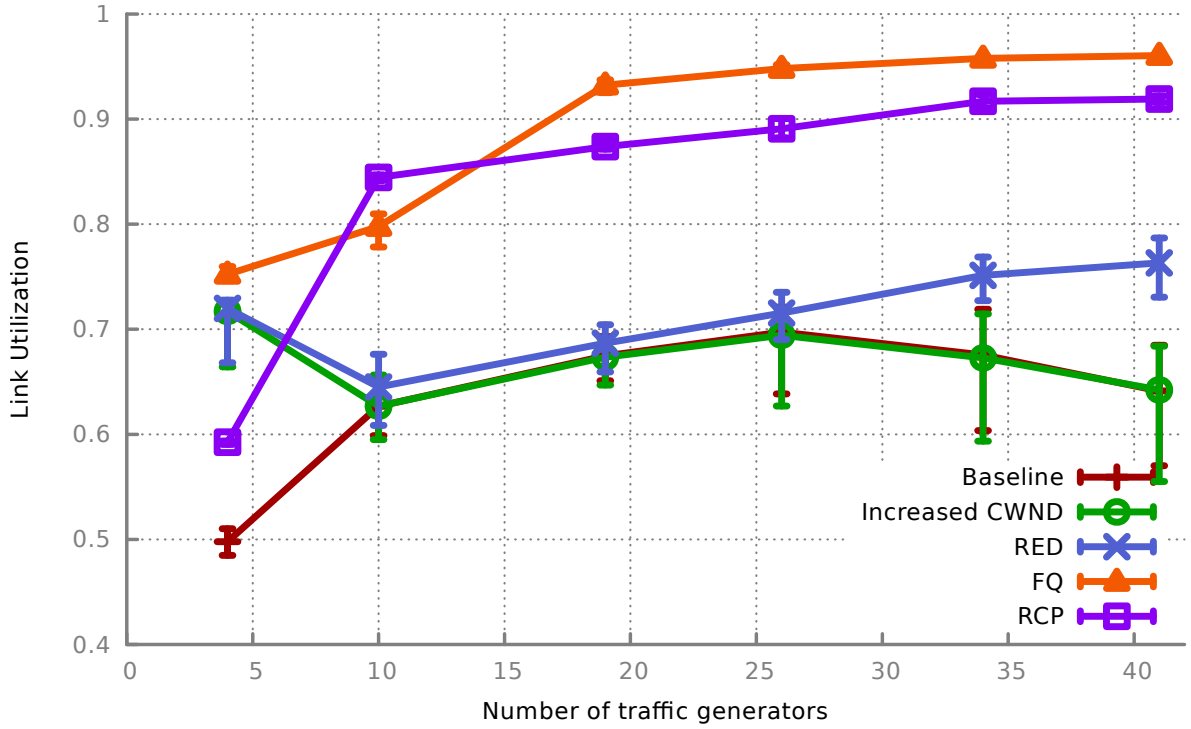


Figure 5.3: Link utilization vs concurrent mouse flows.
Error bars show spread between 10^{th} and 90^{th} percentiles.

between mouse flows.

If we ignore the measuring flow for the moment and start with a small number of background flow generators, we expect the background flows to have a difficult time keeping the bottleneck link busy. As we increase the number of background flow generators, we expect the utilization of the link to increase.

Note that in this case, link utilization is a useful proxy for flow completion time. This is because we never have more flows in-process than the number of background flow generators, and thus every 32 packets sent finish a flow on average.

Figure 5.3 shows median link utilization for each of our five protocols with increasing numbers of background flow generators. The error bars show the spread between the 10^{th} and 90^{th} percentiles.

These numbers were generated from a simulation with 10ms latency and 10Mb/s links, with the measuring flow turned off. After giving the simulation 20 seconds to stabilize, we count the number of packets delivered in each second for the next 20 seconds. Each data point is supported by 500 runs of the simulation, which gives 10,000 samples supporting each data point. The utilization is simply the number of packets delivered during a second divided by 1250, the maximum possible number of packets that can be delivered in a second.

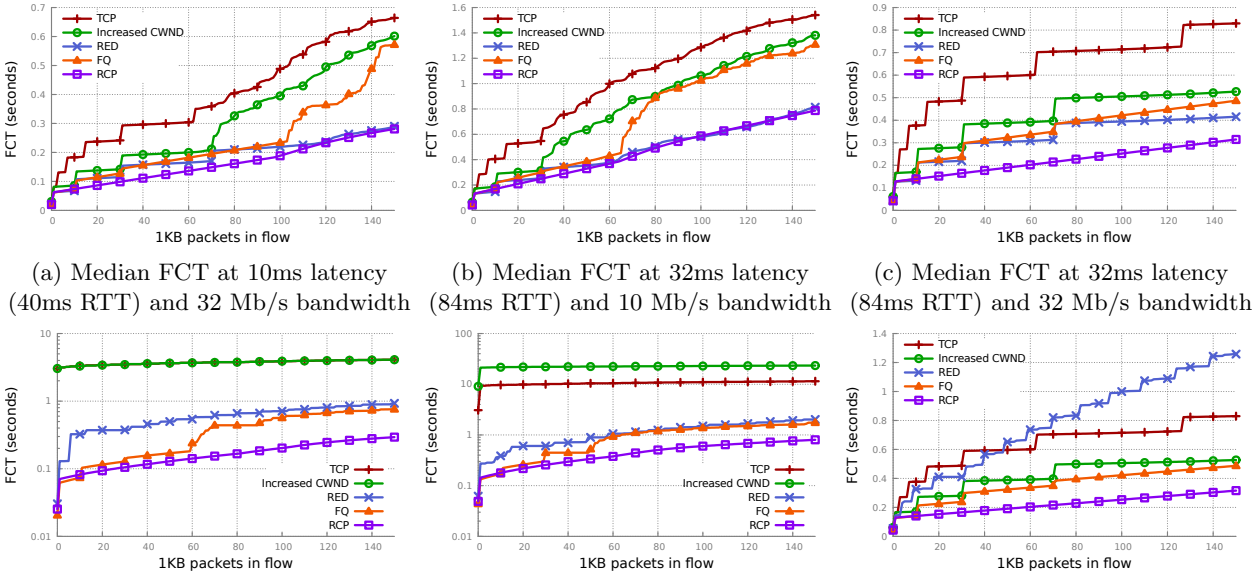
Surprisingly, RCP is not the best. RCP still has a utilization near 0.9, though, and because RCP works

to force the queue towards empty, it is unsurprising that RCP will not fully utilize the link.

Fair Queueing has the highest utilization in this simulation. The intuition here is that Fair Queueing protects the first few packets that a flow sends, keeping each of the flows in slow start longer and thus allowing them to finish faster.

RED and the standard TCPs start together, but as the number of generators increases, RED slowly begins to utilize the link more fully. It is notable that an increased congestion window by itself does not cause an appreciable difference in link utilization.

5.3 Increasing Bandwidth Delay Product



(a) Median FCT at 10ms latency (40ms RTT) and 32 Mb/s bandwidth

(b) Median FCT at 32ms latency (84ms RTT) and 10 Mb/s bandwidth

(c) Median FCT at 32ms latency (84ms RTT) and 32 Mb/s bandwidth

(d) 99th percentile FCT at 10ms latency (40ms RTT) and 32 Mb/s bandwidth (log scale)

(e) 99th percentile FCT at 32ms latency (84ms RTT) and 10 Mb/s bandwidth (log scale)

(f) 99th percentile FCT at 32ms latency (84ms RTT) and 32 Mb/s bandwidth

Figure 5.4: Median and 99th percentile FCTs on links with increasing bandwidth-delay products.

The larger the bandwidth-delay product, the larger the number of packets that could be sent before the sender hears back from the receiver about how many packets made it to the destination. Networks with large bandwidth-delay products are stereotypically more difficult for TCP, but they're particularly difficult for TCP Slow Start.

After one RTT, RCP knows what its fair share rate is and can send at that rate. Slow Start will send an exponentially increasing number of packets each RTT, but the gap between the number of packets sent and the number of packets that could be sent is very large, especially for the first several RTTs. This is one of the fundamental limitations of TCP.

In order to demonstrate this, we test using our standard setup with a fixed number (4) of background flows while increasing the RTT and bandwidth. The results of these simulations are in Figure 5.4. We observe that RCP's gain over TCP grows as bandwidth-delay product grows. At the median, RCP completes flows about twice as fast as TCP across all these bandwidth-delay products.

RED behaves quite well at the median, allowing TCP to complete flows as fast as RCP in Figures 5.4b and 5.4a. However, RED performs significantly worse at the 99^{th} percentile. It is never better than Fair Queueing, and is significantly worse in Figure 5.4f, where it is even slower than standard TCP for flows larger than 50KB.

Fair Queueing is somewhere between standard TCP with an increased CWND and RED at the median. However, it performs significantly better at the 99^{th} percentile, where it gets better, relatively speaking, as the bandwidth-delay product increases.

Simply increasing the CWND of TCP does give a measurable benefit does give a benefit over standard TCP at the median. However, in Figure 5.4e, we observe that increasing the initial congestion window degrades TCP's performance. Since the 99^{th} percentile is experiencing several timeouts, it seems reasonable that increasing the congestion window can increase the likelihood of experiencing multiple timeouts and the number of timeouts experienced.

Chapter 6

Further Research

Our simulations indicate that much of the benefit of RCP and similar green-field protocols for small flows can be achieved with Fair Queueing, a local change that does not require deployment to the whole network. However, there is still a significant gap between FCTs with Fair Queueing and FCTs with RCP. Can this gap be closed?

A significant part of the remaining gap is due to fundamental limitations that TCP Slow Start imposes. TCP Slow Start is required for TCP-friendliness. However, with Fair Queueing in the network, flows are isolated in a way that makes TCP-friendliness unneccesary. How can we take advantage of this to make short flows finish faster?

Chapter 7

References

[1] Appenzeller, G., Keslassy, I., and McKeown, N. *Sizing router buffers*, vol. 34. ACM, 2004.

[2] Breslau, L., Estrin, D., Fall, K., Floyd, S., Heidemann, J., Helmy, A., Huang, P., McCanne, S., Varadhan, K., Xu, Y., and Yu, H. Advances in network simulation. *Computer 33*, 5 (may 2000), 59 –67.

[3] Brutlag, J. Speed matters for google web search. `http://services.google.com/fh/files/blogs/google_delayexp.pdf`, 2009. Accessed: 11/26/2012.

[4] Dukkipati, N., Kobayashi, M., Zhang-Shen, R., and McKeown, N. Processor sharing flows in the internet. *Quality of Service–IWQoS 2005* (2005), 271–285.

[5] Dukkipati, N., and McKeown, N. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev. 36* (January 2006), 59–62.

[6] Dukkipati, N., Refice, T., Cheng, Y., Chu, J., Herbert, T., Agarwal, A., Jain, A., and Sutin, N. An argument for increasing tcp's initial congestion window. *ACM SIGCOMM Computer Communications Review 40* (2010), 27–33.

[7] Floyd, S. HighSpeed TCP for Large Congestion Windows. RFC 3649, RFC Editor, December 2003.

[8] Floyd, S. RED (Random Early Detection) Queue Management. `http://www.icir.org/floyd/red.html`, 2008.

[9] Floyd, S., Allman, M., Jain, A., and Sarolahti, P. Quick-Start for TCP and IP. RFC 4782, January 2007.

[10] Ha, S., Rhee, I., and Xu, L. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev. 42*, 5 (July 2008), 64–74.

[11] Jacko, J. A., Sears, A., and Borella, M. S. The effect of network delay and media on user perceptions of web resources. *Behaviour & Information Technology 19*, 6 (2000), 427–439.

[12] Katabi, D., Handley, M., and Rohrs, C. Congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM Computer Communication Review* (2002), vol. 32, ACM, pp. 89–102.

[13] Kelly, T. Scalable TCP: improving performance in highspeed wide area networks. *SIGCOMM Comput. Commun. Rev. 33*, 2 (Apr. 2003), 83–91.

[14] Moktan, G., Siikavirta, S., Sarela, M., and Manner, J. Favoring the short. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on* (2012), IEEE, pp. 31–36.

[15] Qazi, I., and Znati, T. On the design of load factor based congestion control protocols for next-generation networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE* (april 2008), pp. 96 –100.

[16] Qazi, I., Znati, T., and Andrew, L. Congestion control using efficient explicit feedback. In *INFOCOM 2009, IEEE* (2009), IEEE, pp. 10–18.

[17] Ramachandran, S. Web metrics: Size and number of resources. `https://developers.google.com/speed/articles/web-metrics`, 2010.

[18] Souders, S. Velocity and the bottom line. `http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html`, 2009.

[19] Tan, K., Song, J., Zhang, Q., and Sridharan, M. A Compound TCP approach for high-speed and long distance networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* (april 2006), pp. 1 –12.

[20] Xia, Y., Subramanian, L., Stoica, I., and Kalyanaraman, S. One more bit is enough. *Networking, IEEE/ACM Transactions on 16*, 6 (dec. 2008), 1281 –1294.