Reverse Engineering FastTCP

Monica Pardeshi
Carnegie Mellon University
Pittsburgh, Pennsylvania
mpardesh@andrew.cmu.edu

Ranysha Ware
Carnegie Mellon University
Pittsburgh, Pennsylvania
rware@cs.cmu.edu

Justine Sherry
Carnegie Mellon University
Pittsburgh, Pennsylvania
justines@cs.cmu.edu

ABSTRACT

Congestion control algorithms (CCAs) determine how many packets a sender should send through the network so that network links aren't overloaded. Because CCAs control the sending rate, they are important both for maintaining good Internet performance and for ensuring that Internet users share link capacities fairly. Akamai, a large content distribution network on the Internet, uses a CCA that is based on a published algorithm called FastTCP, but its detailed workings are unknown because Akamai's implementation is proprietary. In this research study, we observed the number of packets the algorithm sent under different network conditions to understand how the algorithm would react. In this paper, we describe the behavior of Akamai's FastTCP algorithm under different network conditions and present experimental values for the algorithm's parameters.

INTRODUCTION

Transmission Control Protocol (TCP) is a protocol that allows a sender and a receiver to exchange data. The data is broken into packets, which the sender gradually sends through the network to the receiver. Once a packet has been sent, there are a few things that could happen. If the receiver receives the packet, then she will send an acknowledgment to the sender to announce that the packet was received. The time it takes for the sender to receive the acknowledgement once the packet has been sent is called the round trip time (RTT). If the sender doesn't receive an acknowledgment within a certain time limit, the sender will assume that the packet was dropped and needs to be resent.

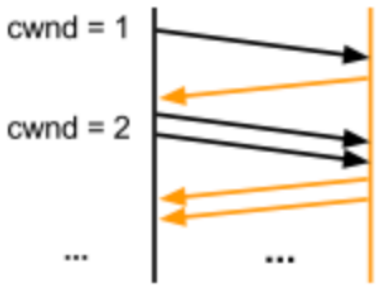


Figure 1: in TCP, a sender (left) sends cwind packets at a time to the receiver (right)

If the receiver is not able to process a packet immediately, then the packet will sit at the end of the receiver's router queue. If the queue is full, then the packet is dropped. How should the sender decide how many packets to send at a time? An entire class of algorithms has been developed to answer this question, namely congestion control algorithms. Congestion control algorithms determine the window size, or the number of packets a sender should send to the receiver so that network links aren't overloaded. CCAs are important for two main reasons. First, they determine Internet performance: if packets are sent too quickly, they will likely be dropped because the network does not have enough capacity and becomes overloaded. If packets are sent too slowly, download speeds will be slow. The second reason is that CCAs control Internet fairness--if multiple senders are sending data across the same link, how can they send at rates where they all use the same amount of bandwidth?

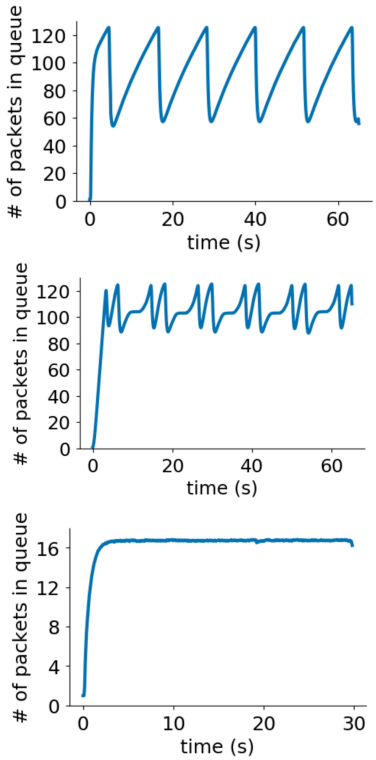


Figure 2: queue occupancies of three different CCAs (from top to bottom: Reno, Cubic, FastTCP)

BACKGROUND

Here we will describe how some well-known CCAs work. Perhaps the simplest CCA is Reno. Reno has two phases: slow start and congestion avoidance. During slow start, Reno starts with window size 1. Then, it doubles the window size until it notices packet loss. At this point, Reno enters congestion avoidance. When there is packet loss, the algorithm cuts the window size in half, then increases the window size by one until it notices loss again. This results in the sawtooth pattern in Figure 2. Prior work has shown that Akamai sometimes uses Reno but it is not known when. The main CCA that we researched is FastTCP.

$$w \longleftarrow \min\left\{2w,\ (1-\gamma)w + \gamma\left(\frac{\text{baseRTT}}{\text{RTT}}w + \alpha\right)\right\}$$

Figure 3: the window update function that FastTCP uses

The published FastTCP algorithm updates its window size according to the formula in Figure 3, where α is the maximum number of packets in the queue at any given time and γ is a value between 0 and 1 that tells how much to weight new samples. This results in a smooth curve, as shown in Figure 2. FastTCP updates its window size every 20 ms.

PROBLEM STATEMENT

Our goal was to answer two questions: 1) under which conditions does Akamai use Reno, and under which conditions does it use the "normal" FastTCP algorithm? 2) When Akamai uses FastTCP, how are the parameters configured? In order to answer the first question, we ran experiments on 10 different websites using our testbed and varied the bandwidth, RTT, and queue size. We observed queue occupancy. Our values for bandwidth ranged from 2 mbps to 512 mbps, our RTT values ranged from 20 ms to 512 ms, and our queue size values ranged from 16 packets to 512 packets. In order to find experimental values for the window size parameters, we used the same test bed and used network conditions that produced FastTCP curves. Then we observed the maximum queue occupancy.

METHOD

In order to determine the behavior of the algorithm, we set up a testbed with three nodes: a client, a server, and a BESS node.

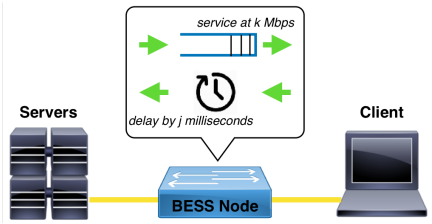


Figure 4: our testbed setup

BESS is a software switch that allows us to control the queue size, network latency, and bandwidth. We set BESS's bandwidth to low values to ensure that this bandwidth would be the lowest in the network. By making BESS the bottleneck, the queue size reflected the window size of the sender, and measuring BESS's queue occupancy allowed us to see the window size. The bottleneck queue size measures the number of packets the sender puts in the network because it is processing the packets the slowest, thus all arriving packets line up in the queue. For example, the queue occupancy of Reno mirrors the "additive increase, multiplicative decrease" window size pattern, as shown in Figure 2. Cubic, another CCA, also begins by doubling its window size in the slow start phase. During the congestion avoidance phase, Cubic adjusts its window size according to a cubic function. Indeed, the graph of Cubic's queue occupancy looks like the graph of a cubic function, as shown in Figure 2.

RESULTS

We ran experiments on 10 different websites that are known to be served by Akamai with different network conditions. For the sake of brevity, we have included graphs from just one website for each set of conditions. In the first round of experiments, we varied the bandwidth.

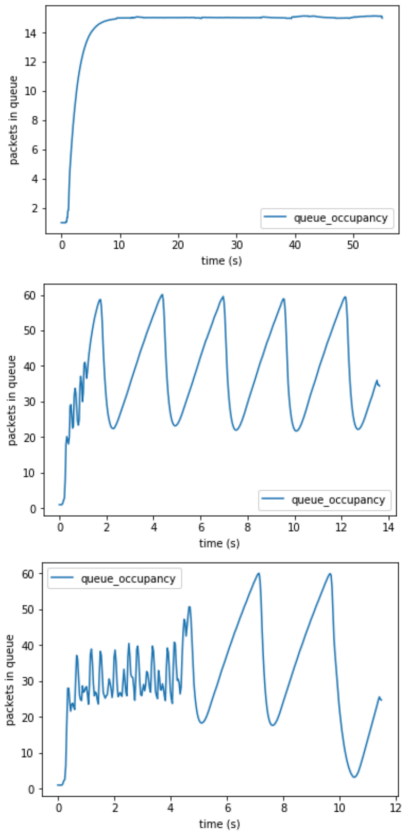


Figure 5: queue occupancy for two different websites with RTT = 35 ms, queue size = 64 packets, and bandwidth = 2, 14, 17 mbps from top to bottom

Changes in bandwidth produced the most interesting results. We observe that when the bandwidth reaches around 15 mbps, the graphs look like Reno. We would expect instead that Reno would be used at low bandwidths. To test this hypothesis, we also tried running some experiments with higher bandwidths. When we increase the bandwidth, it is less likely that BESS is the bottleneck, so it is difficult to tell whether our graphs still mirror the sender's window size, so we do not know if Akamai switches back to FastTCP at higher bandwidths.

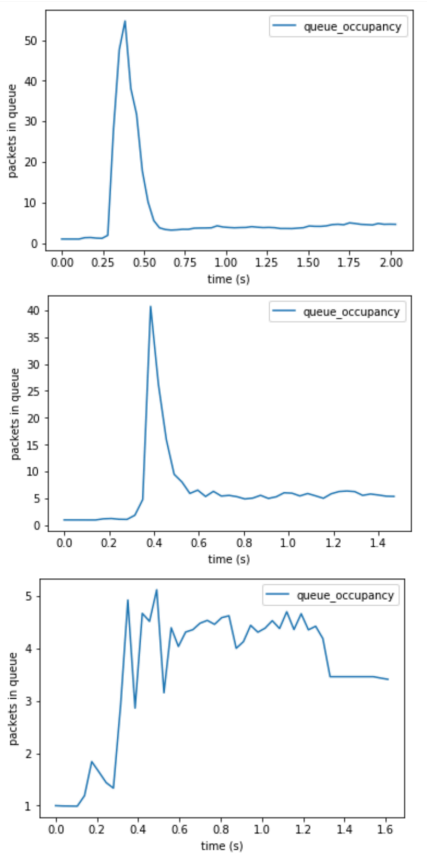
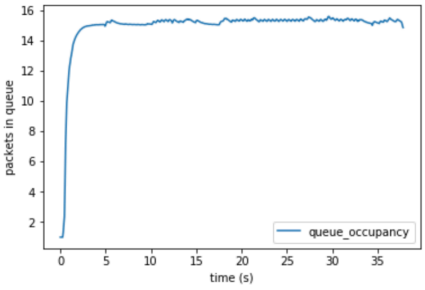


Figure 6: at higher bandwidths, the queue occupancy does not look like that of any known CCA. From top to bottom, the bandwidths are 128, 256, 512 mbps

In the next set of experiments, we varied the RTT.



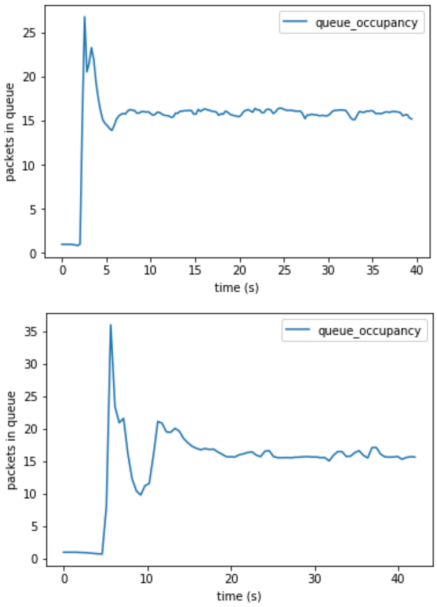
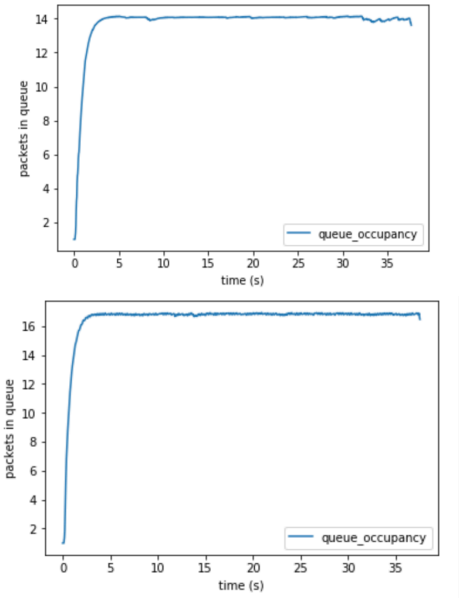


Figure 7: queue occupancy for two different websites with queue size = 64 packets, bandwidth = 5 mbps, and RTT = 64, 256, 512 mbps from top to bottom

We observe that the algorithm overshoots and then levels out at around 16 packets. This is because FastTCP only updates its window every 20 ms. If the RTT is much larger than 20 ms, then the algorithm will not know that it has sent too many packets until much later.

Finally, we varied the queue size.



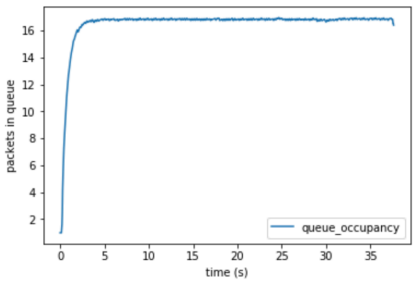


Figure 8: queue occupancy for two different websites with bandwidth = 5 mbps, RTT = 35 mbps, and queue size = 16, 64, 256 packets from top to bottom

The window size does not seem to be affected by queue size, as in each plot, the window size flattens out at around 16 packets.

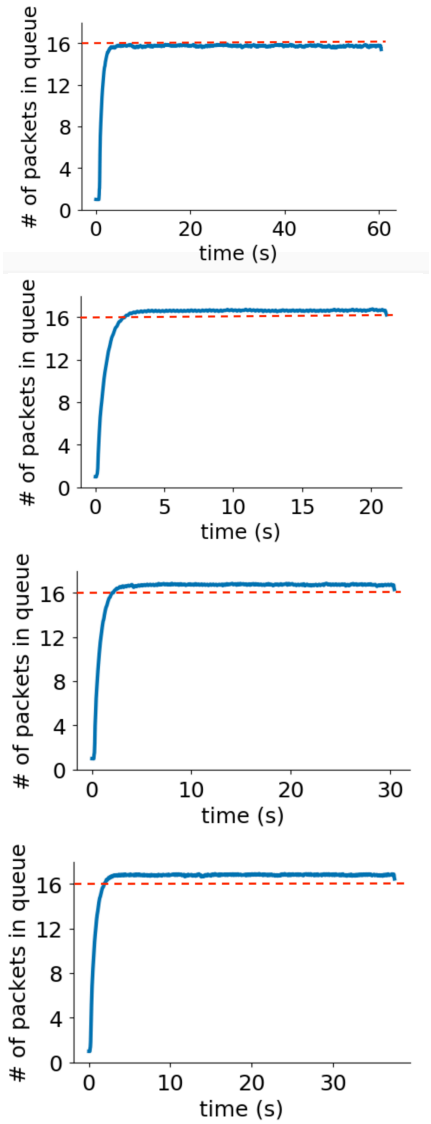


Figure 10: Max queue occupancy is always about 16 packets

Our second goal was to find experimental values for the parameters of FastTCP. By observing the graphs with FastTCP curves, we determined an experimental value of $\alpha = 16$.

CONCLUSION

Akamai seems to use standard FastTCP under most network conditions. As the bottleneck bandwidth increases to around 15 mbps, the algorithms switches to Reno. When the bandwidth becomes too high, the window size does not look like any known CCA. We also determined that an experimental value for α is 16.

FUTURE WORK

In the future, we hope to ask Akamai whether our understanding of their CCA is accurate. We also hope to run experiments with multiple flows to see how FastTCP flows interact with other flows and to see whether FastTCP is fair. Finally, we hope to determine an experimental value for γ.

ACKNOWLEDGMENTS

I would like to thank Ranysha Ware and Justine Sherry for their invaluable guidance and support.