Listening with TSDE (Transport Segment Delay Estimator)

Kathleen Nichols
Pollere, Inc.

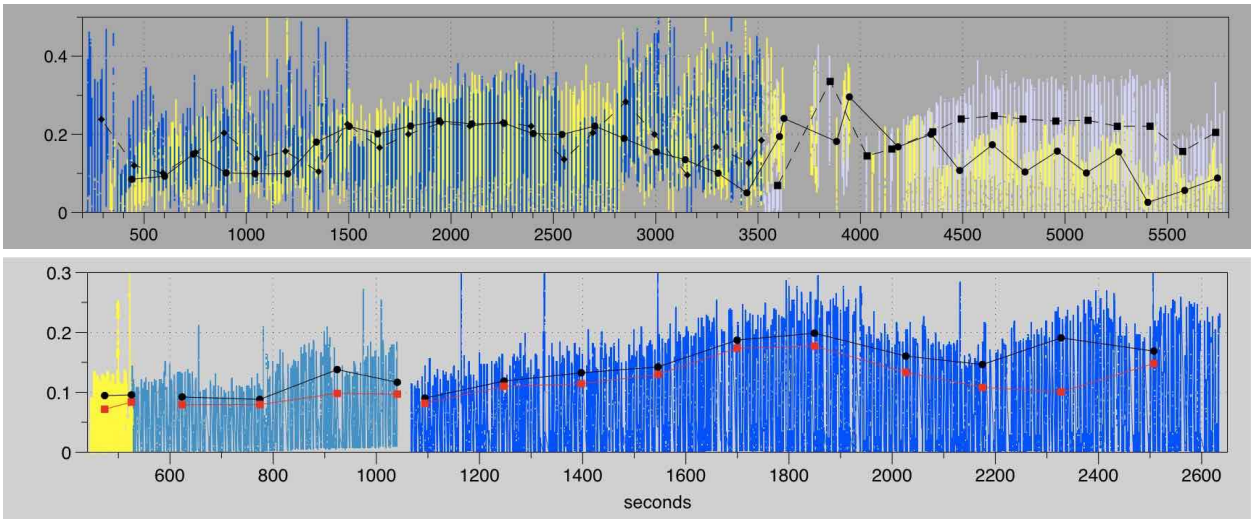
Basic Information

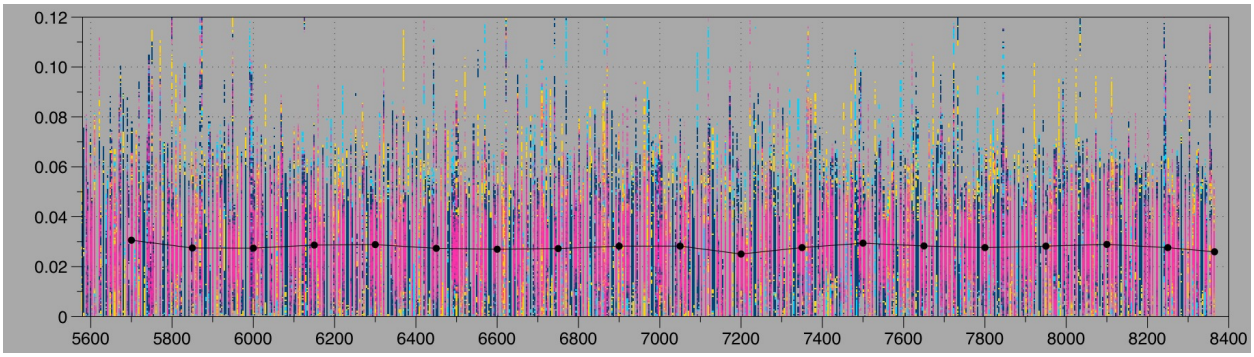
Pollere has been working on TSDE under an SBIR grant from the Department of Energy. In the process of debugging and validating the code, we deployed it in a device and tried it out on a home network. We found the data interesting and thought others, particularly in the *bufferbloat* community, might also. This note primarily covers data collected in the second quarter of 2016. Errors and misconceptions are all mine.

A Test Drive

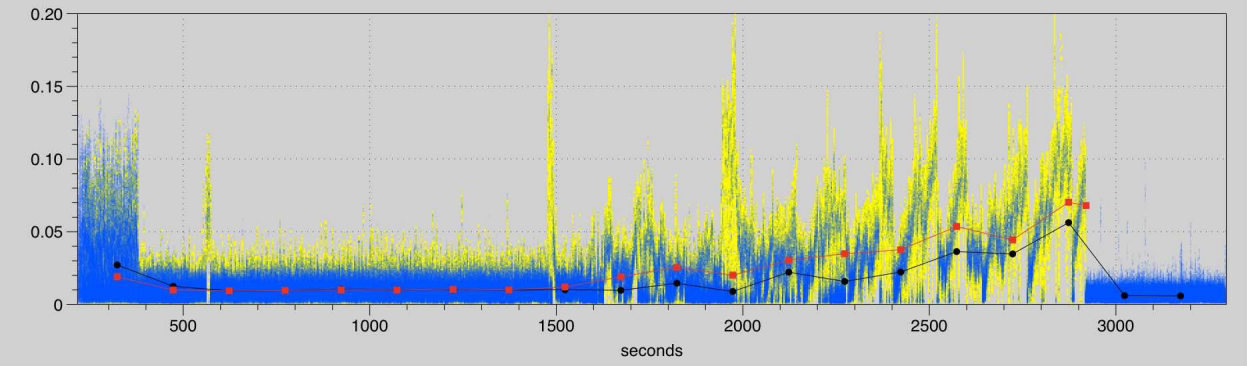
Development and validation originally focused on packet traces, so it was pretty cool to have a live device available. We hooked this up in our house and started streaming video. The box was connected well inside the home network so experienced network (particularly wifi) delays inside the house. There are a number of delay metrics possible, but we focused on the downstream queuing delay, or delay variation, from the server to our measurement point close to the client. When the first tests were done, the downstream bandwidth available was 30-40 Mbps. TSDE's GUI was fairly basic, so we saved its output to analyze and plot later.

First is the big picture(s). These plots cover the entire video stream. The delays are what the packets see, so a median value is the median delay the packets saw, not the median delay of the queue over time. These cover Netflix (NF), GooglePlay (GP), Amazon (AZ), and there's also Apple iTunes (IT) but it's done as a file transfer so is quite different. Here's what *appears* to be going on, but please correct me if I'm wrong. NF, GP, and AZ are chunk-oriented, reading a chunk from storage and then sending it, but the sending part is quite different. Netflix interleaves two flows of video and two flows of audio at a time, at least most of the time. Each video stream is 3.2 Mbps. GP uses a single 6.8Mbps flow at a time, though the flow changes over time. This AZ has four video flows and each is about 2 Mbps. (An AZ a few days later was mostly four flows.) The round trip delay to the servers was 14-15 ms for NF and GP, but 70ms for AZ. The median delay values for each 2.5 seconds are shown as black dots; the red dots on GP are 25th percentile. Below are NF, GP, and AZ in order.

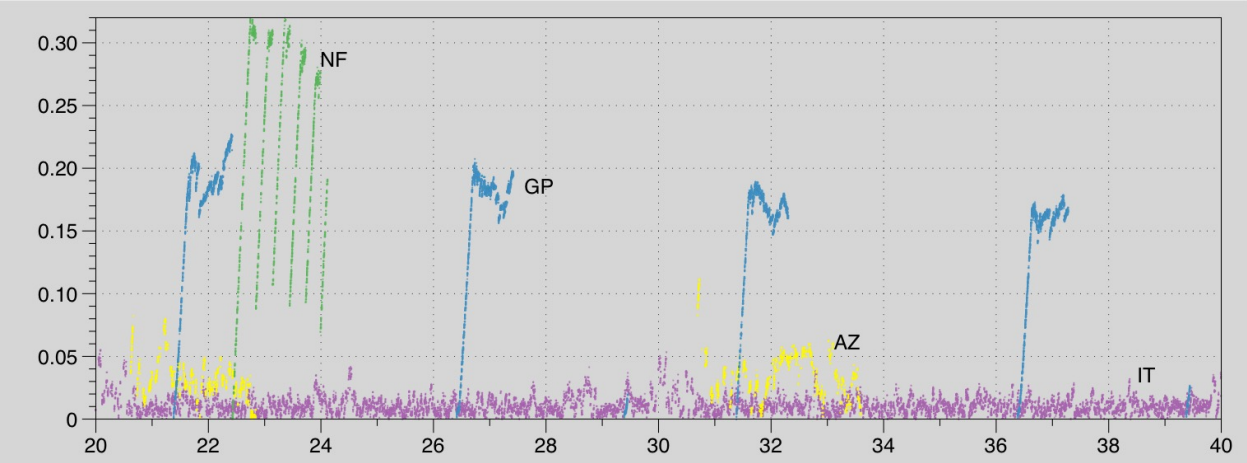




Below is the IT streaming. This is really strange because one stream is to the iPad that "owns" the content (median in red) and the other is to the Apple TV (median in black) that was showing it. This isn't supposed to happen. It appears that the client applications for AZ and IT are advertising smaller windows than NF and GP, avoiding the larger delays.

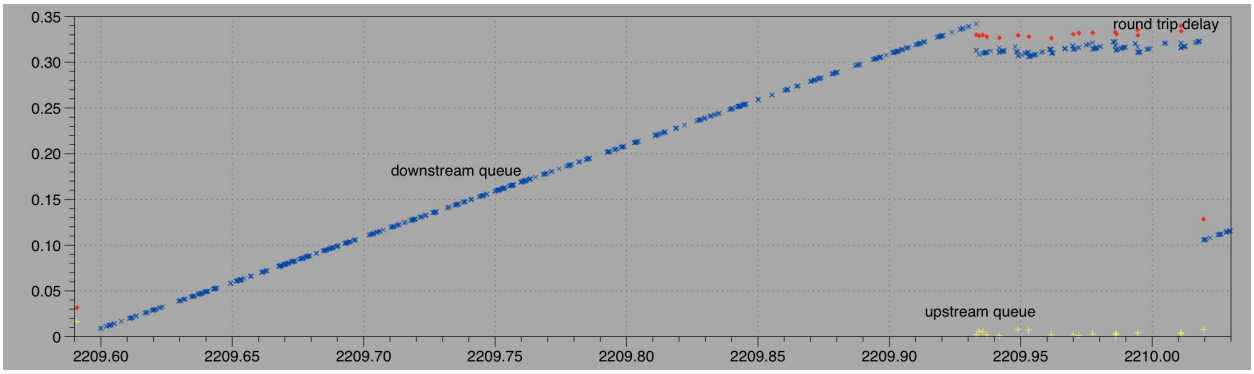


Zooming in to see what happens in individual transfers was pretty interesting (to me, anyway). The usual behavior for NF is each stream transmits its chunk at 20 second intervals, the two streams are interleaved so there's a chunk sent about every 10 seconds and sending takes about 3 seconds. There is about 5 seconds between each chunk transfer of GP and each chunk takes about 1.3 seconds to send. The AZ streams usually each send at 10 second intervals, but 1) sometimes one misses an interval and 2) they aren't interleaved, but all burst at the same time. (Note another AZ stream a few days later was more interleaved.) In summary, GP transmits a chunk for about 1.3 seconds every 5 seconds, NF simultaneously transmits two flows that each transmit for about 2.5 seconds every 20 seconds, AZ simultaneously transmits four flows that each transmit for about 3 seconds every 10 seconds, and IT just does a steady file transfer. The NF flows are usually nicely interleaved while the AZ flows all happen at the same time, but other experiments with both have shown the opposite behavior. Taking one flow from each and normalizing time to show what the downstream delay (and activity) looks like:

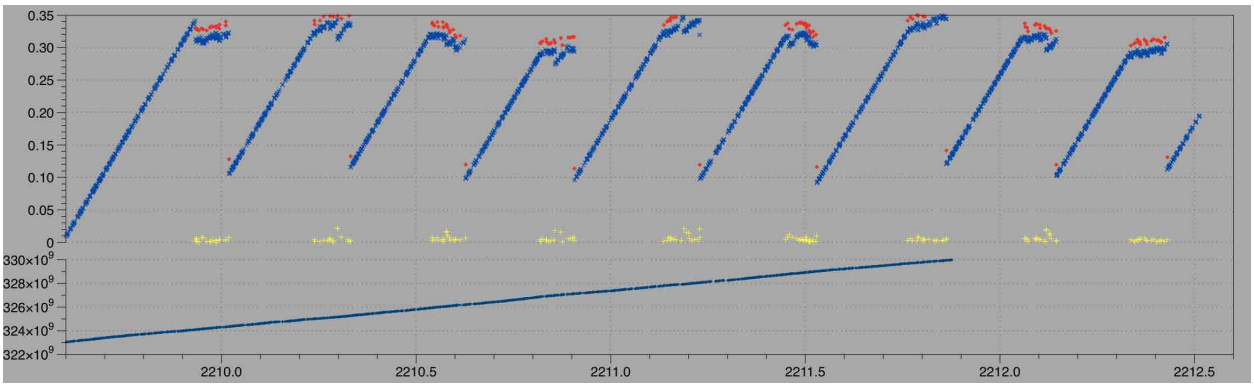


Over this period, NF would have one burst from the other flow at around 32 seconds and there would be three other flows active when the AZ one is active. There's plenty of idle time on the links (except for IT with it's smaller, steady queue), but some huge delay swings. Makes one wonder what would happen to a VoIP on the same link.

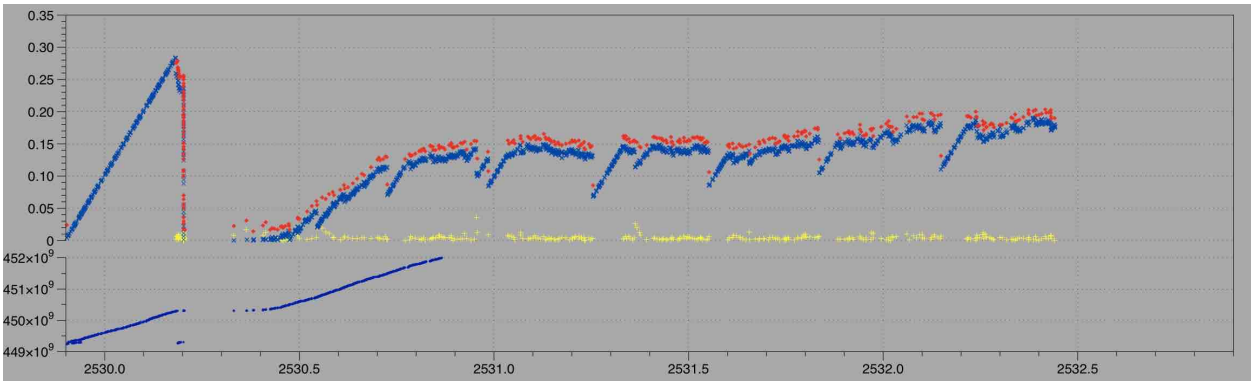
I'm going to pick on NF since they create the most interesting data. Both NF and GP do an initial burst, but GP then settles into a somewhat normal TCP connection. NF does repeated bursts in each of its sending phases. We zoom in on on of those little NF "shepherd's hooks" from the previous plot. (Yeah, they are green above and blue below.)



Each point is a per packet delay. The ramping up is due to dumping a burst of about half a megabyte of data into the downstream path. The linear delay increase shows how each packet queues behind more packets in front of it. The delay flattens out when the maximum sending window was reached. The big drop in delay at the end means that nothing was added to the queue for about 200 ms. So, is this burst pattern a good thing? Below are two NF chunk transfers with very different outcomes, a three second time slice of each. It aids understanding to look at the evolution of the sequence numbers early in the stream so that's shown beneath the time plots. Blue is the downstream queue, yellow is the upstream queue and red is the round trip delay. The first one is the "normal" case and sequence numbers increase linearly.



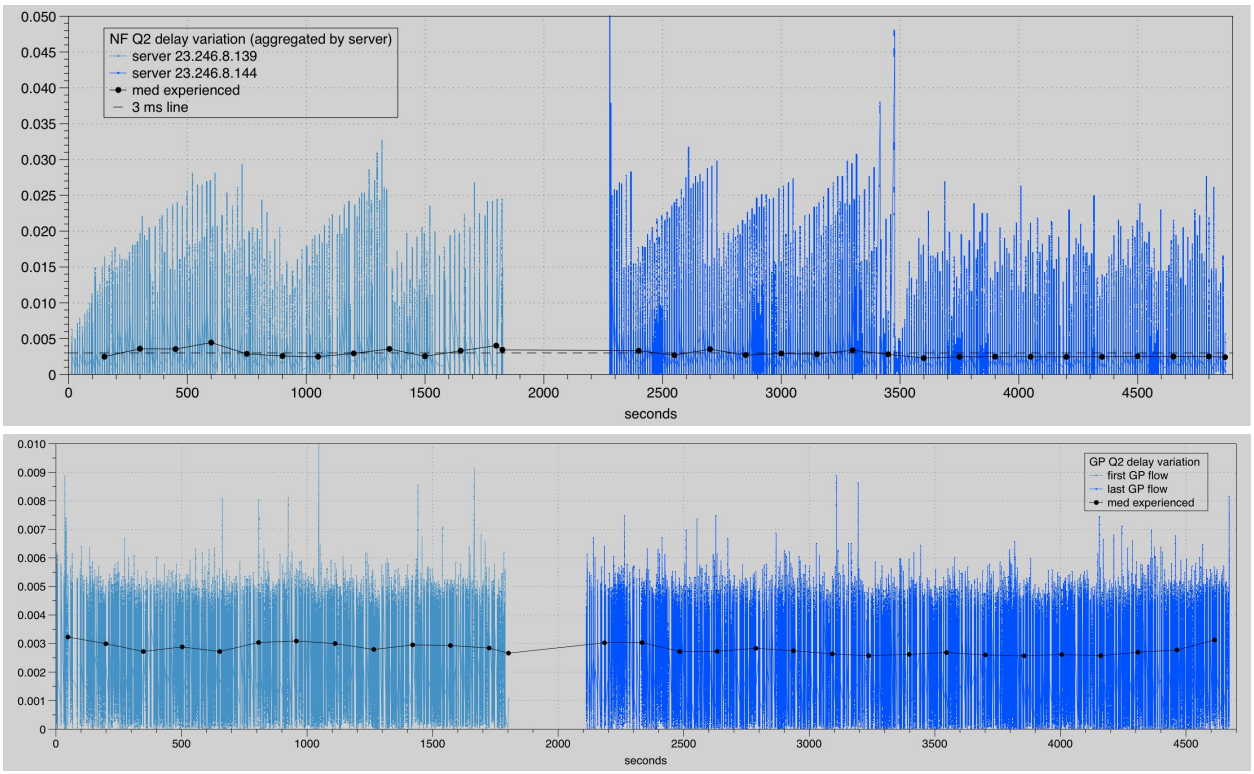
In this chunk transfer, there is an early packet loss, which is actually a reordering that the sender treats as a loss, so there is both a retransmit and a timeout. This seems to cause the connection to go into standard slow start behavior instead of the big bursts. What's interesting is that despite the early loss and smaller window in effect for most of the transfer, the chunk transfer finishes more quickly.



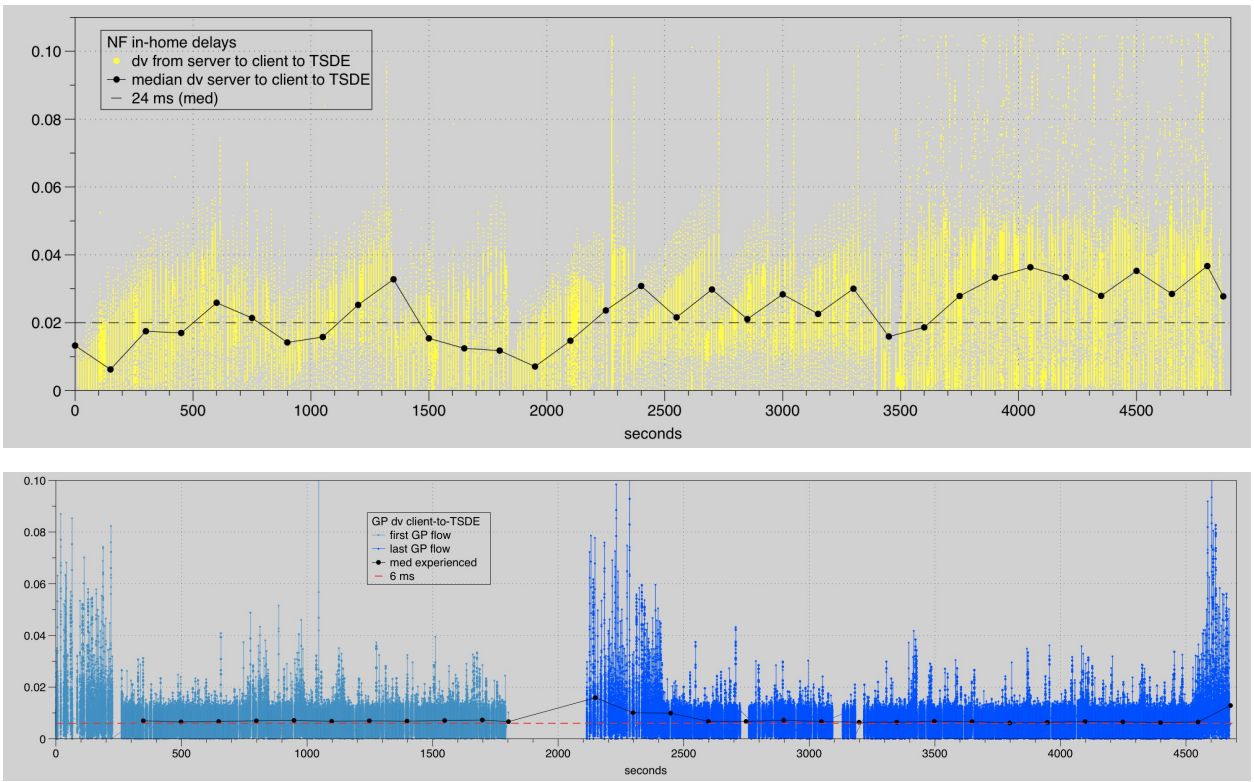
Test Drive on a Faster Track

Recently, we started getting about 178 Mbps downstream on speedtests and reran the NF and GP experiments. The transfer patterns haven't changed at all, but the delay pattern has. For NF, I aggregated the interleaved delays (by server) to make the downstream delay the focus. An interesting thing about this experiment is that the interleaving didn't always stay so nicely spaced as previously. Despite a few longer delays (much shorter than at the lower bandwidth), the median delays stay right around 3 ms downstream. The video client is on an Apple TV with a 100 Mbps interface so now the bottleneck is in the house. The downstream GP delay is even more boring, but has the same roughly 3ms downstream delay.

The NF pattern shows that the window is not being reset when the connection idles.



It's a little harder to get a good picture of the client-side delay, but we can get the delay variation from the server to the client and back to the TSDE for NF and the client to TSDE delay variation for GP. In both cases, there's more delay on the client side now, NF is more affected since it's on the Apple TV with its 100 Mbps interface.



About TSDE

Current approaches to measuring network delay largely rely on injecting packets for short periods, adding information to packets, deploying cooperating agents, or costly timing infrastructure. What's missing is a way to extract the packet delays seen by applications using the Internet. Under a grant from the Department of Energy under Award Number DE-SC0009498, Pollere has developed a reference Transport Segment Delay Estimator (TSDE) software module and prototyped this patent pending technology in a Intel Atom-based device configured as a transparent network bridge.