Gordon Measurement methodology

Ayush Mishra

March 2019

1 Introduction

This document discusses the measurement methodology behind **Gordon**, a Network measurement tool for mapping out the entire congestion control behaviour of servers on the Internet. The measurements done by this tool are done over multiple connections made by a *wget* client. The tool has two components, the *Prober*, responsible for making the measurements, and the *Classifier*, which parses these measurements to classify them into a congestion control protocol. This document discusses only the *Prober*.

2 Mechanism

The Prober functions as a packet interceptor (Figure 1) between the sender and the receiver. It is implemented using Netfilter Queues, and counts the congestion window sizes as discussed below.

Through the course of a test, the probe makes multiple connections with the server. In each of these connections, it measures the a different congestion window - i.e. in the first connection it will measure the first window's size, in the second the second window, and so on. TCProbe starts by making a connection and dropping all the packets it receives. In this case, the number of packets it receives before getting a re-transmission of the first packet will be the size of the first congestion window - C_1. In the next connection, it will accept and acknowledge C_1 packets, and start dropping again till it receives a re-transmission for the C_1+1^{th} packet. The probe follows this strategy to measure any congestion window C_n, given by accepting $(C_1 + C_2 + C_3 + ... + C_{(n-1)})$ number of packets, and then dropping them until it receives a re-transmission.

While it does make *window measurements*, and can not accurately capture the fluctuation of rate based algorithms, it can still identify them as long as they have a unique signature in the measurements. Also, since windows are a rough estimate of the rate over the RTT, TCProbe will be able to estimate the behaviour of these rate based algorithms in the congestion control phase as well. In fact, the only thing our measurement methodology is susceptible to is random losses on the internet. We address this issue in Section 4.

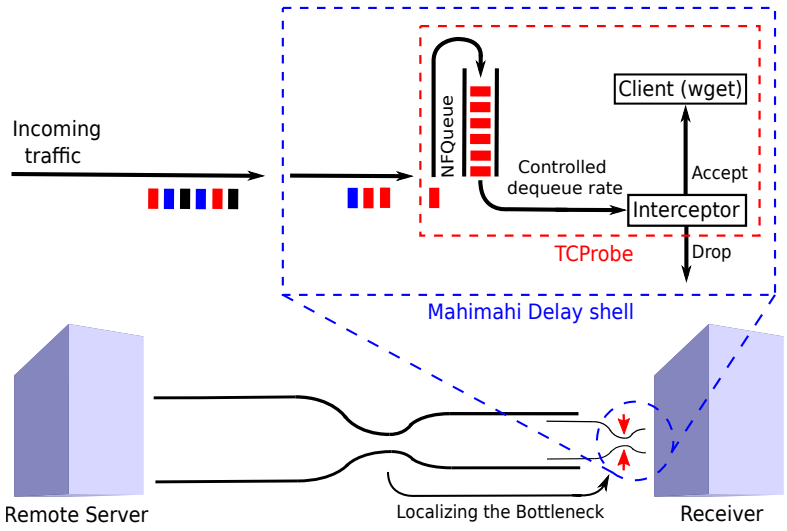

Figure 1: The Gordon Design

3 Emulating network changes

Apart from being able to count the congestion window sizes, we also wanted to emulate some common stimuli that elicit characteristic responses from certain congestion control algorithms.

Reaction to Loss. Most window based congestion control mechanism infer congestion after seeing a packet loss, and adjust their congestion window growth functions accordingly. These behaviours are often characteristic of that particular TCP variants, and measuring this change would be a good way to discern between variants. To see this change, we introduce a loss-emulation mechanism in the packet counting procedure discussed before. Say we want to emulate a loss of the 20^{th} packet. While measuring the window that contains this packet (Figure 6), we operate as usual - accepting the first 19 packets and then dropping packets till a re-transmission of he 20^{th} packet is received. In the next connection however, we accept all but the 20^{th} packet. Then, once we receive a re-transmission of our previously dropped 20^{th} packet, we continue dropping till the next re-transmission. The window that TCProbe measures in this connection, therefore become the effective loss-window of the connection - since the sender will react to the intermediate loss of the 20^{th} packet. Figure 3 shows a sample cubic flow reacting to such losses emulated by TCProbe.

Change in Bottleneck bandwidth.
Rate based congestion control schemes like BBR often ignore loss as a sign of

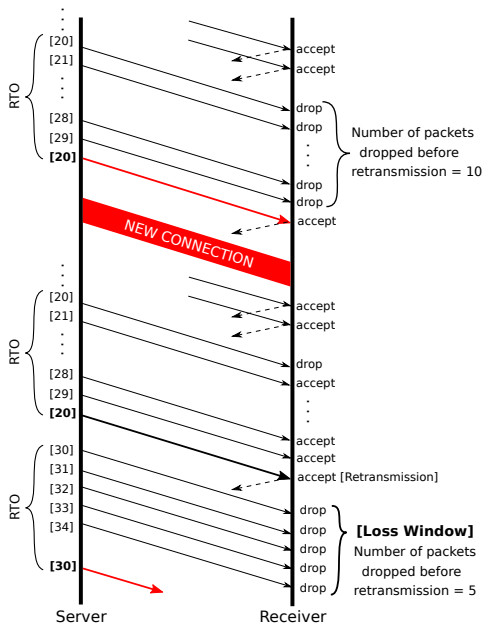

Figure 2: Emulating a loss

congestion, and instead adjust their sending rates according the the bandwidth that they receive at the bottleneck. For example, congestion control algorithms like BBR try to match the bottleneck bandwidth with periodic probing for change in bandwidth. This behaviour should ideally reflect in the measured congestion window sizes. To see this behaviour, TCProbe controls the rate it which it dequeues packets from the queue. This effectively does two things - firstly, with a low enough rate, it allow TCProbe to move the bottleneck of the connection locally to an observable space where it can see the flow interact with the Buffer. Secondly, now that the end-point congestion control algorithm optimizes for this local bottle-neck, it can vary the this bottleneck bandwidth and see the congestion control algorithm's reaction. Figure 4) shows a sample BBR flow's measured congestion window sizes while interacting with this localized bottleneck.

Inflating the RTT. For reasons discussed in Section 4, The probe also inflates the RTT between itself and the Server. To do so, we utilize a Mahimah delay shell. Overview of the complete design in Figure 1.

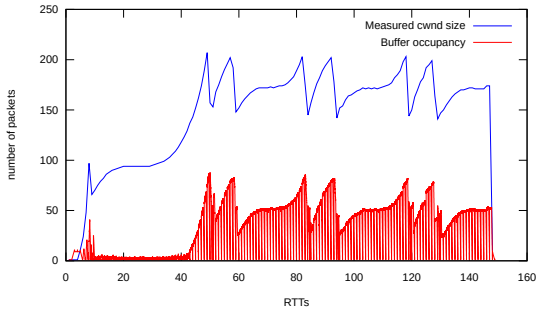

Figure 3: The observed congestion windows for a sample Cubic flow reacting to Gordon's emulated losses when 1) the congestion window rises above 80 packets for the first time and 2) Every time the buffer occupancy exceeds 80 packets.

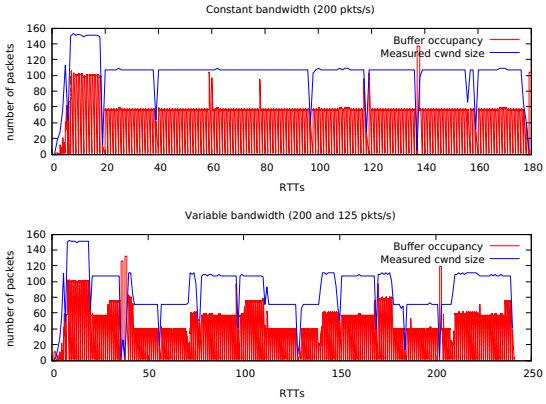

Figure 4: A sample BBR flow's congestion window measurements when the local bottleneck provisions a constant rate of 473 kbps (or 200 packets per second, top) as compared to when the bottleneck oscillates the rate between 473 kbps and 296 kbps (125 packets per second) every 3000 packets received (bottom). The connection used an MTU size of 296 bytes.

4 Parameterization

Drop Points. Since most of our key differentiation between algorithms happens in the congestion control phase, our methodology for the test at a high level is that we let the congestion window grow to a certain size, force the flow to back off, and then observe how it reacts. Be it window based or rate based protocols, where a flow enters it's congestion control phase sets rough bounds on the window size/rates the sender uses for the next few RTTs. This, effectively also decides on how long a flow would last (given the fixed size of the web page that has been requested)

For example, Bic and Cubic reduce their congestion window size ($cwnd$) by a factor of β and try to reach the cwnd size at which they last saw a loss (L) as soon as possible - limiting their congestion window sizes to $[L.\beta, L]$ for a few RTTs.

Dropping too early would mean window sizes would be really small, decreasing the precision of our experiments. Dropping too late would mean large window sizes that will cause short flows that wouldn't be long enough to make any observations. Where we emulate a drop also decides on the shape we get to see, as seen in Figure **??**. Ideally, we want a drop point that allows a connection that can sustain for a long time, as well has give us measurements that are able to capture all the features that help us identify a particular algorithm.

Through our measurements, we found out that most connections have a starting window of 10 packets. This means that we can expect a typical slow start window sizes like 10, 20, 40, 80, 160 and so on. We decide to emulate the first drop when the cwnd size exceeds 80 packets. This gives us a reasonable precision and allows us to monitor anywhere between 2000-3500 packets over a period of 30 RTTs (depending on the algorithm). Using an MTU size of 296 bytes would mean needing file sizes between 300-600 kb - which are readily available on most websites today. This also allows us to capture most of the features of the congestion control behaviour of an algorithm within 30 RTTs.

Setting the Bottleneck bandwidth. Making flows like BBR back off, on the other hand, is more tricky since they don't react to a loss event. However, since BBR flows try to match the bottleneck bandwidth, squeezing the pipe at the localized bottleneck should be enough to manipulate such flows. Ideally, we want the rate based flows to back-off around the same window size as a window based flow to capture the same amount of congestion control phase behaviour, but not the exact same value - such that we can differentiate between a back-off caused by a loss event, or the bottleneck bandwidth.

Since our measurements are window based, it is hard to calculate an accurate bottleneck bandwidth for a target window size, however, we can make some estimations about this rate by setting a target buffer occupancy.

For a target buffer occupancy B, the parameters D (Emulated one-way delay) and $1/d$ (Bottleneck bandwidth - each dequeue event happens every d ms) $B = 4D/d$, . Because of this, the measurements are done at a 125ms one-way delay, with a rate of 200 packets/sec (d=5ms). This rate is later (after receiving the first 1500 packets) changed to 125 packets/second to observe how such flows react to a change in bandwidth. The proof for this is as follows:

by *Little's law*, the number of packets in the bottleneck buffer can be given by

$$B = R.d$$

Where, R is the arrival rate of the packets at the buffer, and $1/d$ is the departure rate from the buffer (i.e. the bottleneck bandwidth). For BBR, the target R

is set approximating the bottleneck bandwidth as the arrival rate, which can be estimated as the measured window size *cwnd* divided by the average perceived RTT. Hence,

$$R = \frac{cwnd}{2.D + \frac{d.B}{2}}$$

Here, the perceived RTT is twice the emulated one-way delay D, plus the average queuing time in the buffer. Replacing this equation in the *Little's law* expression, we get a quadratic equation as follows:

$$2D.B + \frac{d.B^2}{2} = cwnd.d$$

solving this equation for B, we get:

$$B = \frac{2D \pm \sqrt{4D^2 + 2d^2.cwnd}}{d}$$

For one-way delays much larger than d, we can regard $2d^2.cwnd$ as negligible. Which leave us with:

$$B \simeq \frac{4D}{d}$$

Both the parameters D and d can be controlled by the probe, and are set to $D = 125$ and $d = 5$ for a maximum buffer occupancy of 100 packets. This buffer occupancy actually shows up as around 50 packets in the buffer, because out of the 100 packets, 1 BDP worth packets in the connection are actually on wire, and not in the bottleneck. This is a side-effect of modelling the entire connection (network pipe plus the bottleneck) as a single buffer in the *Little's Law* derivation.

5 Adapting to Internet tests.

Dealing with noisy measurements. Our measurement mechanism is vulnerable to only one thing - losses. This makes our methodology susceptible to errors/noise in our measurements. However, it is possible to eliminate this noise to a certain extent. If we study this problem closely, we can see that where the random loss happens can characterize what kind of noise we get in our measurements. Figure **??** shows that when the packets are lost in the Accept window or the Drop window of the connection, the noise is always negative. It is only when the re-transmit marker packet is last that we end up counting the entire re transmitted window twice and have positive noise. the positive nose can be combated by aborting a test every-time we see any but the first Drop window packet re-transmitted.

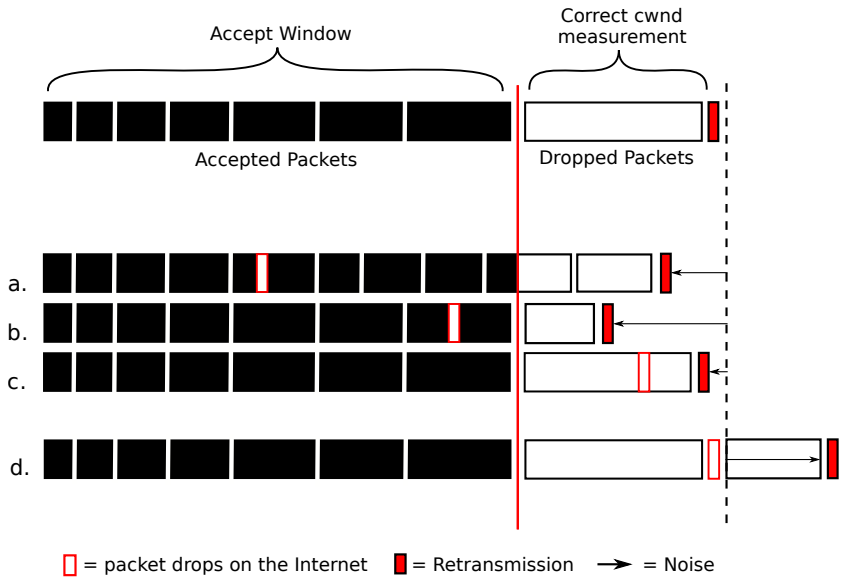

Figure 5: How noise is always negative

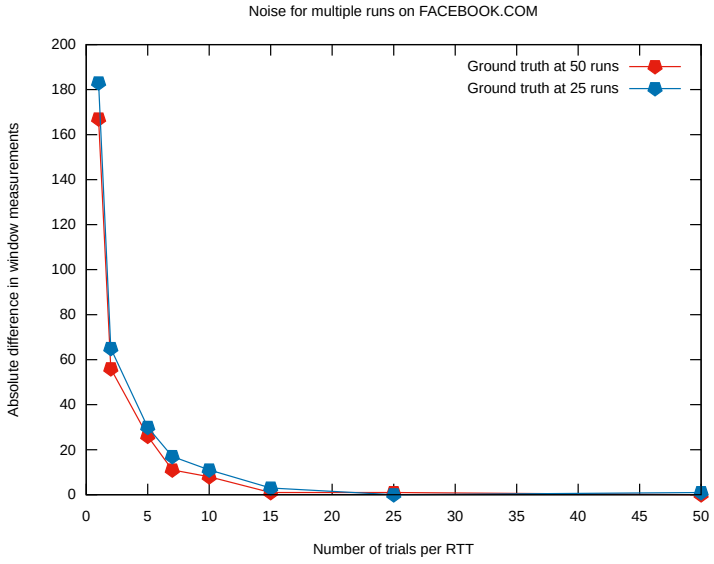

Figure 6: Sensitivity analysis for number of trials for eliminating negative noise from measurements.

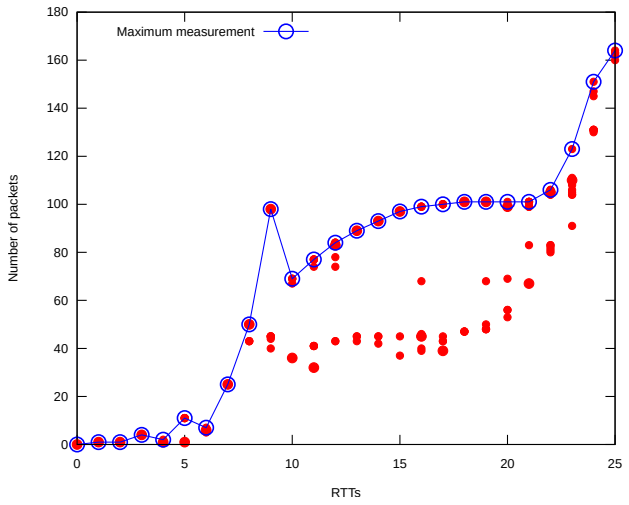

Figure 7: Measurements for Reddit.com

We remove the negative noise to be robust to losses in other regions by taking the maximum window measurement as the congestion window that RTT. We tested multiple per RTT sample spaces for minimizing noise for the lowest number of trials (Figure.**??**), and found 15 trials per RTT to give us reasonably low amount of per-RTT noise.

Figure 7 shows the window measurements for **reddit.com**, with 10 trials done per window measurement. We can see that this is sufficient to ignore any erroneous measurements, and trace what looks like a clean Cubic congestion window evolution graph.

Setting MTU sizes and getting large enough web-pages. Other than this, we also want to be able to sustain a long enough connection to get meaningful measurements for a server. To do so, we use an MTU size of 296 bytes - that was small enough to sustain a long connection, as well as large enough to allow a successful SSL handshake with most of the servers on our our list. We also crawled all our target domains for the largest web-page they could serve - giving us the best chance to make sufficient measurements t identify and classify the congestion control behaviour of web servers.

Figure 8 illustrates the response of commonly used congestion control variants to our measurement methodology. From the results, it is clear that Gordon is accurately able to map out the expected congestion control behaviour of these algorithms.

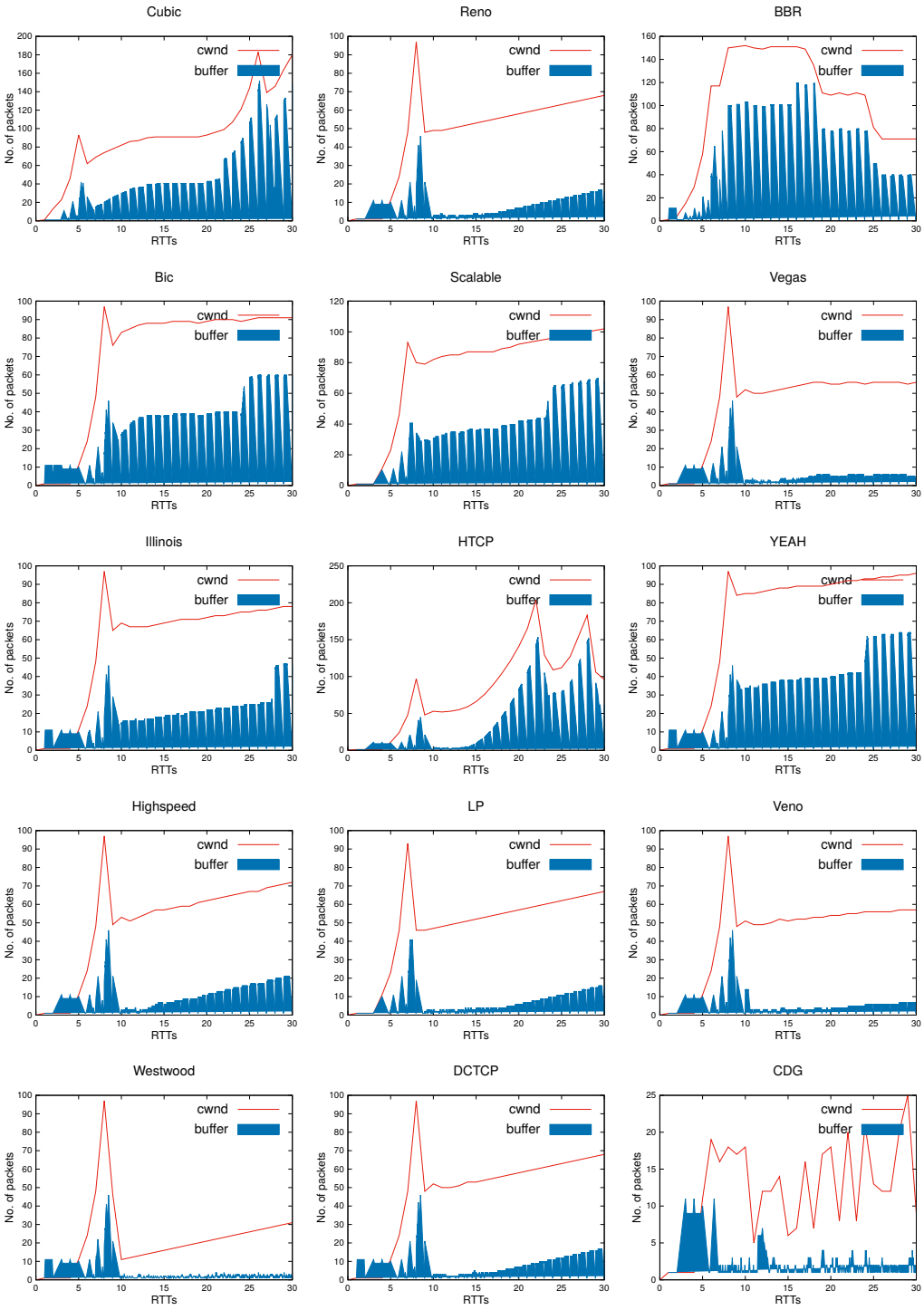

Figure 8: Control tests.