Passive Network Sensing
Advanced Technology Group
February 2023

CableLabs®

A Novel Method for Passive Measurement of Network Utilization Using TCP Statistics

Prepared by
Jeremy Diamond, Software Engineer, Advanced Technology Group | j.diamond@cablelabs.com

Additional Contributors
Steve Arendt, Principal Architect & Director, Advanced Technology Group | s.arendt@cablelabs.com

Executive Summary

Active methods of testing network links for bandwidth and congestion are limited in their effectiveness. Traditional methods hog the connection and more modern statistical methods sacrifice accuracy and reliability. By measuring key metrics involved in TCP congestion control that are already available on the device it is possible to recognize congestion, determine if the congestion is caused by the device under test, and estimate bandwidth. This method may be used continuously and can function under conditions where active methods would fail.

Introduction

In network communications, it is useful to know how much available network bandwidth exists and how much is being used by a device or an application. This information is valuable for steering a device between networks, determining if a network is properly sized, etc. Methods to determine available bandwidth are typically active, where a collection of packets is sent over the network, and the maximum bandwidth for the link is calculated. Active measurements can be brute force, where the link is simply flooded with traffic and the throughput measured, or they can be more selective, where carefully timed sequences of packets are sent over the link to determine the available bandwidth.

Active measurements inject new traffic over the network, and in so doing reduce link capacity. The current paper looks at an alternative to active measurement. We propose a method to measure current link utilization from a device and to deduce whether the link is maxed out. Our method is entirely passive; it uses only traffic that a device is already transmitting on a network. We specifically use TCP traffic and leverage the queue of TCP packets awaiting acknowledgement to deduce utilization.

Background: Existing Bandwidth and Congestion Estimation Methods

The premise of a brute force test is to send a large amount of data from a source such that the network link is maxed out. A basic brute-force speed test can be done by transferring a large file and measuring the time taken to complete the transmission. Similar tools like ookla, fast.com, and perf tools are more efficient and feature rich than file transfer but fundamentally operate in the same way.

Brute-force speed tests have clear limitations. They require substantial bandwidth, incur use costs on users, and drain device batteries. Their largest limitation is that they occupy the entirety of a connection. For accurate results, the source device must also pause its network activity, lest that activity cause inaccuracies in the measurement results.

To make up for these shortcomings, statistical approaches to active link measurement to estimate the available bandwidth and other network conditions have been developed. These methods measure the effects of known behaviors of networks in various states with a set of sample packets. For example, packets of different sizes have a predictable variance in transit time based on available bandwidth. These statistical methods could send a set of sample packets that reveal the bandwidth based on variance in transmit time by size. Several such schemes exist [Band Tools] but, they all have similar potential shortcomings. The output metrics these methods produce are sometimes not directly comparable to traditional methods thus they can only be used by professionals familiar with the tool. They also require a functional connection to a server that manages the test. Thus, a connection can be of such low quality that the test cannot produce useful results. The reliability of these results is inversely related to impact on the network. The more samples sent through the network the more accurate the results, but this also increase the likelihood of impact on network performance. These drawbacks are acceptable under certain conditions where bandwidth estimate is not exposed to the end user, connections so poor they can't be tested are unimportant, and high accuracy of metrics is not a priority.

As an alternative to both traditional and statistical methods, passive measurement schemes that use the information on a single device to measure the network have been developed. Several such methods already exist [see Band Tools in the references below] however they have yet to gain popularity. One possible reason for this is, that they all establish a way for the clients to communicate with the server and use it to send information on their relevant stats. The following is a novel passive network measurement method that uses exposed information built into TCP connections. As such this method eliminates the need for a client to install additional software.

1. Passive Estimation of Bandwidth and Congestion Using Preexisting Stats in TCP

TCP or Transmission Control Protocol is a standard for reliable ordered and error checked connections. TCP contains several stats and mechanisms related to congestion and send rate, that monitor network state. In practice two of these metrics are used here to estimate the state and cause of congestion. Those metrics are TCP Wait Queue Sizes and Mean TCP Congestion Control Receive Window.

1.1. TCP Wait Queue Size

By virtue of TCP’s reliability mechanisms every packet sent is kept in memory on the sender side until the receiver verifies that they received it. The receiver communicates to the sender that a given packet was received in its acknowledgment packets. Thus, for any TCP connection there is a queue of data in memory, of variable size, that expands and contracts depending on how quickly packets are acknowledged. We refer to the size of this queue as the TCP Wait Queue Size. It would make sense that under congested conditions the relative number of bytes waiting to be acknowledged compared to uncongested connections would increase dramatically.

1.1.1. Measurement of TCP Wait Queue Size

On Linux systems several tools exist that can generate stats about the state of a given connection. The command line package ss (socket stats) has tools for measuring memory allocation associated with a given socket. The command “ss –m <ipAddr>” will output several values related to the memory used by a given socket for a given purpose, one of these being the TCP Wait Queue Size at a high degree of precision. The underlying mechanism for this measurement is a kernel interface called NetLink.

1.2. TCP Congestion Control Receive Window (R-Win)

TCP contains a congestion control protocol that allows machines to try to mitigate congestion automatically. Several of the exposed stats in congestion control communicate the state of senders and receivers. The congestion control Receive window size or R-Win of any given packet is an exposed value in the packet header. The R-Win is a signal to a sending device from the receiver of how much data the receiver can receive at a given time. This information about the receiver’s state can be used to make inferences about the source of congestion. If the receive window is large, then any detected congestion must be caused by limitations on the network or sender. Conversely, if the receive window is small during a period of congestion, then the receiver can’t handle the volume of traffic being sent. Hence, the receive window data can be combined with knowledge of the sender’s state like the TCP Wait Queue Size to infer more about the location of a bottleneck.

2. Experimental Setup

We performed a set of experiments to see if knowledge of both the TCP Wait Queue Size and the R-Win could be used to determine if a single network connection was at its throughput limit, and, if so, whether the limiting factor was the network or the receiver. These experiments were conducted on highly simplified networks. In the first, 2 laptops were connected to a single AP (Access Point) (802.11AC), one by Wi-Fi and one by Ethernet. In the second both were connected, by ethernet, to the same AP as before. Displayed datasets will be clearly marked by Wi-Fi or Ethernet to denote which case they represent.

To provide a source of network traffic, a series of scripts ran iperf bandwidth traffic at various speeds. In half of all iperf runs, the bandwidth was increased linearly in 5% increments from 50% to 150% of the known maximum of the link. The other half of iperf runs, were random sequences of increments from 50% to 150%. By comparing these linear and random test series effects that indicate congestion at a given step were identified. The maximum link speed was captured directly by iperf runs prior to and after the scripts were run to ensure consistency.

To monitor network conditions during the iperf runs, the scripts also ran a tshark or wireshark capture while simultaneously capturing the NetLink memory data for the target and source ip address. The Netlink data was continuously tagged with microsecond accurate epoch times for synchronization in post processing.

Iperf runs were conducted at a variety of speeds in the Wi-Fi case. The speeds were reduced by attenuation of the Wi-Fi signal using distance and obstructions.

2.1. Tests

For the purposes of this paper a step is a run of iperf TCP traffic for 30 seconds and a test is a series of steps that target bandwidths of 50% to 150% of an input value. Table T1 is a list of all tests in this analysis.

Table 1. All tests run by name, pre-test measured Mbps and Max Mbps observed during the test

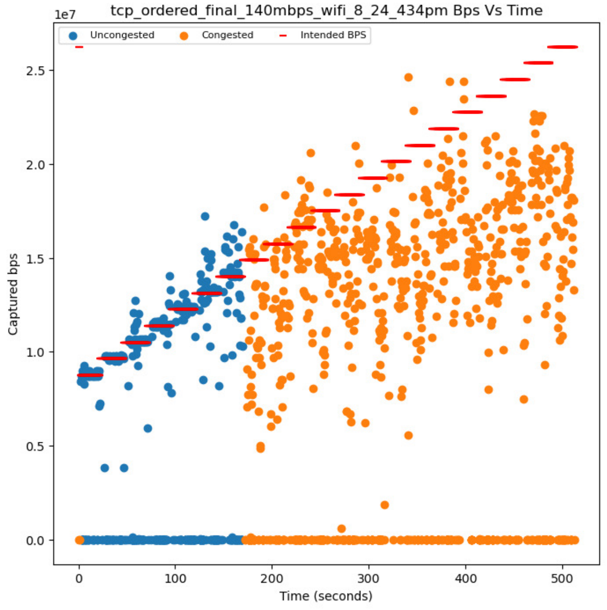
Name (tcp_<orderOfSteps>final<preTestMbps><interface><month><day><time>)	Pre-Test Mbps	Max Mbps Observed During Test
tcp_ordered_final_140mbps_wifi_8_24_434pm	140	140
tcp_ordered_final_310mbps_wifi_8_24_540pm	311	350
tcp_ordered_final_433mbps_wifi_8_23_415pm	433	600
tcp_ordered_final_940mbps_ether_8_23_415pm	940	1200
tcp_rando_final_140mbps_wifi_8_24_434pm	140	140
tcp_rando_final_310mbps_wifi_8_24_540pm	311	350
tcp_rando_final_433mbps_wifi_8_23_415pm	433	600
tcp_rando_final_940mbps_ether_8_23_415pm	940	1200

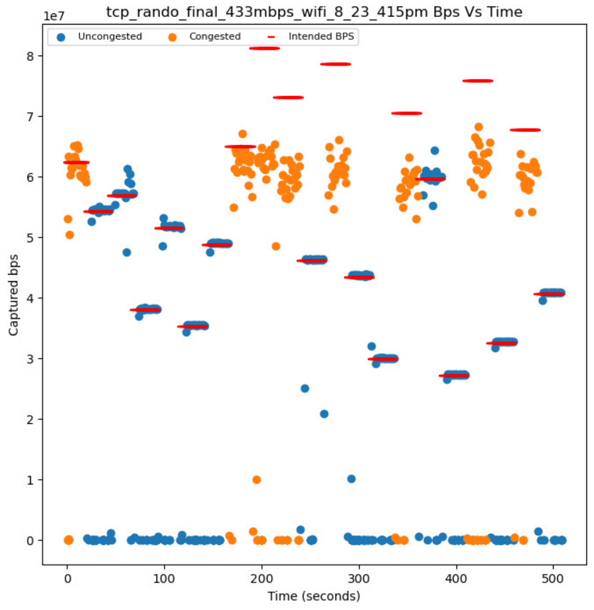
For each test, the iperf bandwidth observed before the test is not the exact same as the maximum bandwidth observed during the test on congested steps. This makes sense as the was an average over 30 seconds and as such it’s possible for the maximum to exceed that average.

3. Analysis and Results

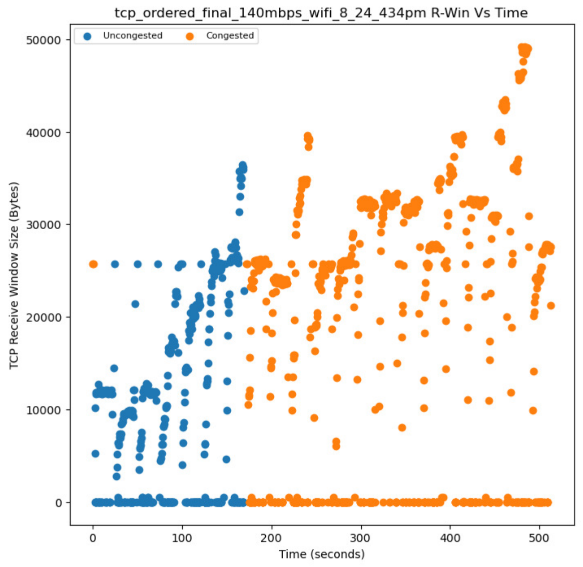
The throughput, R-Win, and TCP Wait Queue size were all averaged over each second and graphed on a scatter plot.

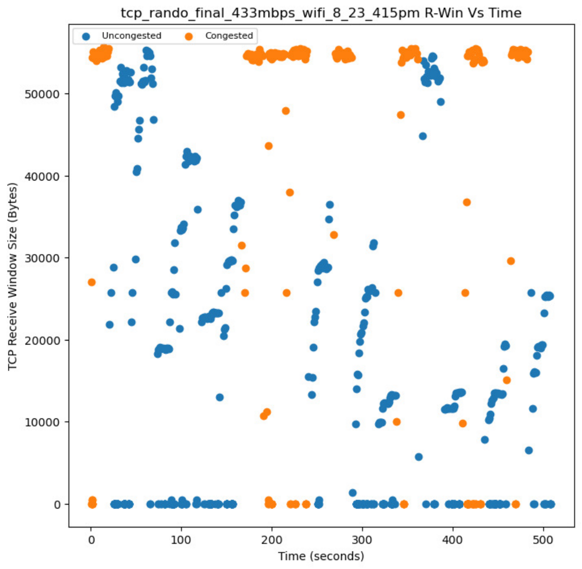
Figures 3&4 Throughput (bps) Scatter Plot Examples



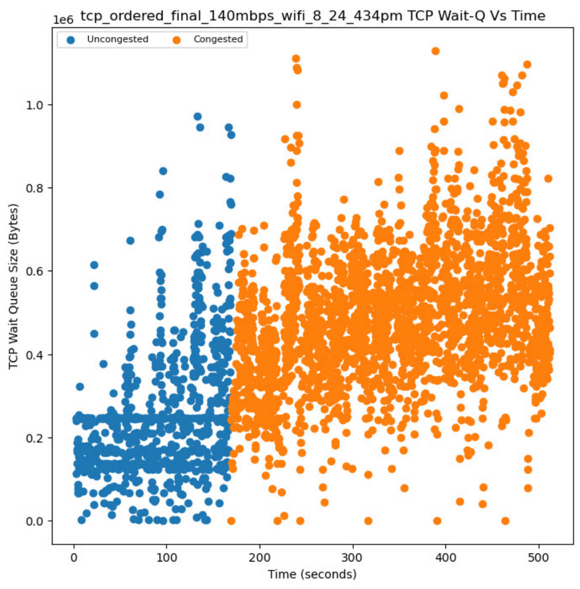


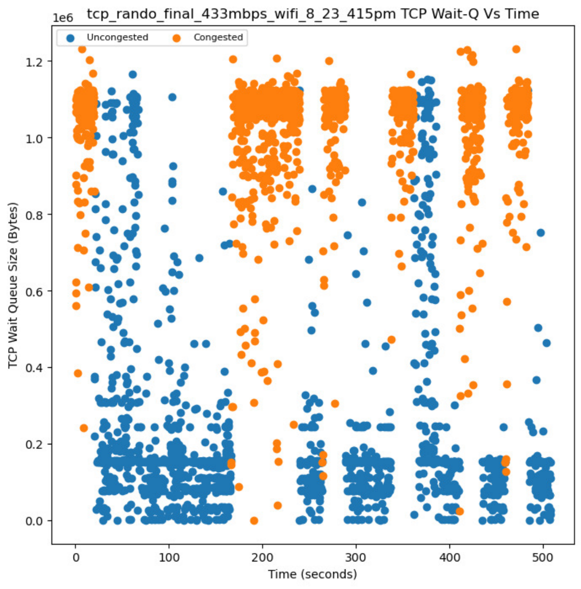
Figures 5&6 R-Win Scatter Plot Examples





Figures 7&8 TCP Wait-Q Scatter Plot Examples

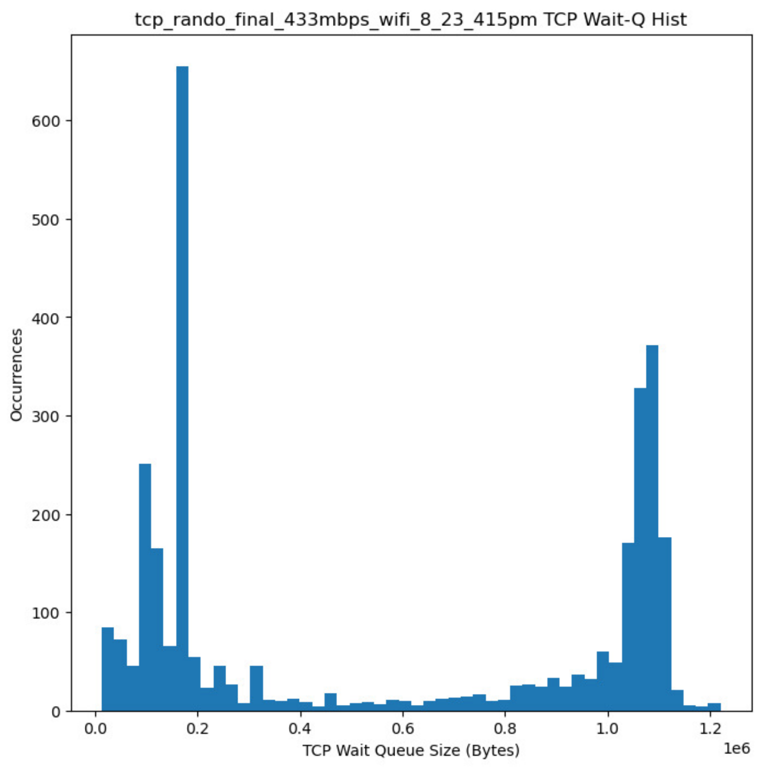




3.1. Use of TCP Wait Queue Size in Congestion Detection

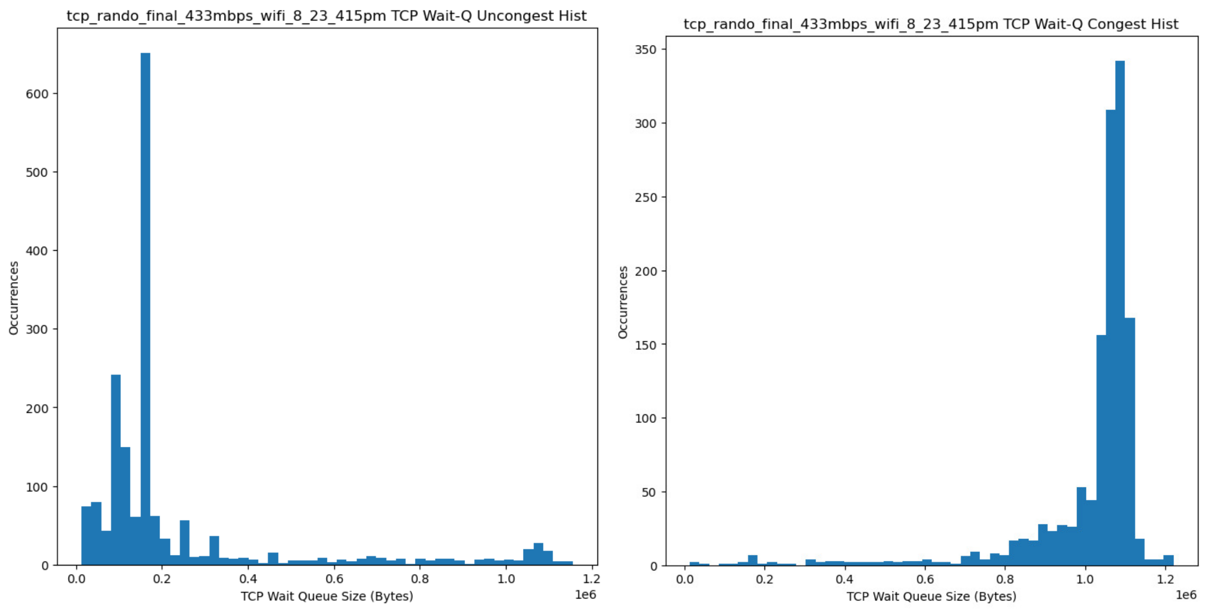
As can be seen in figures 7 and 8 in both the congested and uncongested steps there is an extreme degree of scatter in TCP Wait Queue size and the maximum value was present in each step. To clarify this data, a histogram of each test was plotted. There are 2 major peaks in these histograms as can be seen in figure 9 which correspond to a bimodal distribution.

Figure 9: Example of TCP Wait-Q bimodal distribution



If we separate the data for congested times from the data for uncongested times, the bimodal distribution cleanly splits in two. In the histogram for only the uncongested steps only the lower peak is present, while in the histogram for the congested steps only the upper peak is present. This is shown in figures 10 & 11.

Figures 10&11: TCP Wait-Q histograms for congested and uncongested steps



If we assume that the total range of the TCP Wait Queue size is constant in time, then the above data suggests that the median of the TCP Wait Queue sizes will be below the midpoint of the total range for uncongested conditions, and above the midpoint for congested conditions. **This provides a remarkably easy metric for measuring whether a network is congested or not.** To formalize this, we propose the metric M3R:

M3R = Median/(Max - Median),

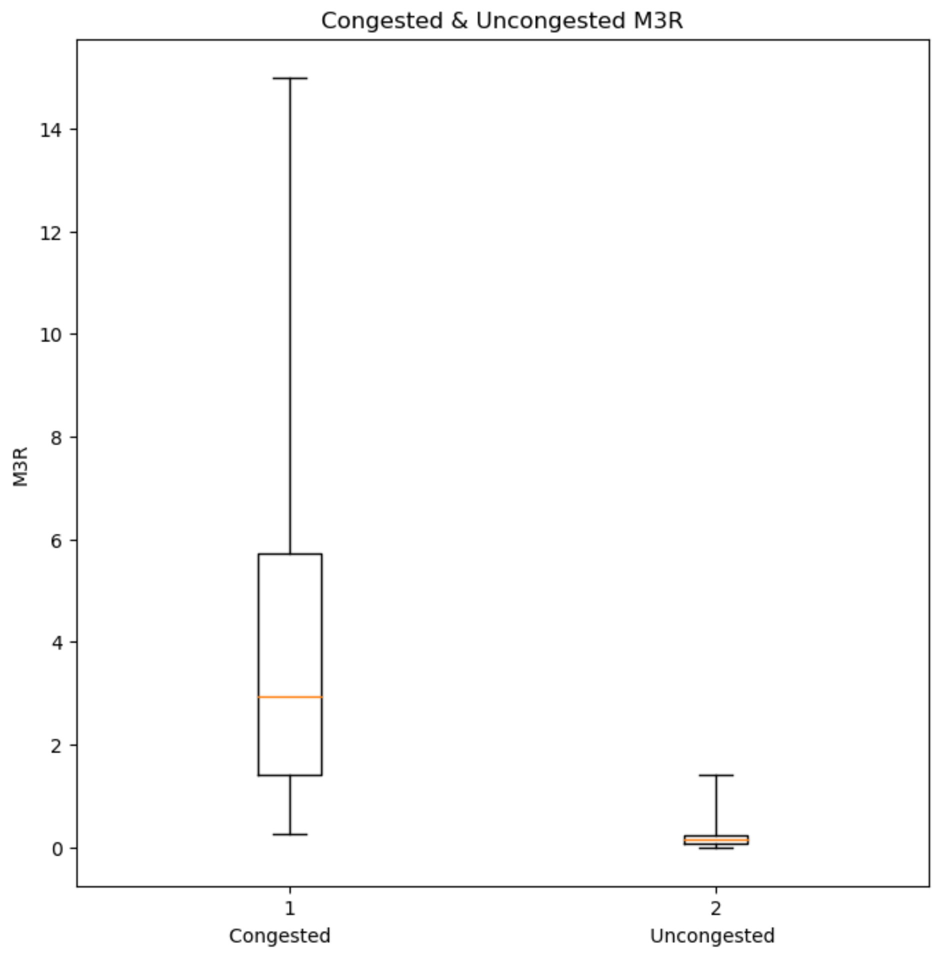
where the Median and Max refer to histograms of the time sequence of TCP Wait Queue Size. M3R will be below 1 during uncongested conditions and above 1 during congested conditions.

To put this metric into practice, we need to know the total range of TCP Wait Queue Size. The minimum and maximum TCP Wait Queue size are defined by the sending system, with the minimum being 0. The maximum could potentially be found as a setting on the sending system, but it turns out to be straightforward to determine it from the TCP Wait Queue Size measurements themselves. The TCP Wait Queue Size contains sufficient scatter that the maximum of the range is present for at least some samples in the histogram, so one can simply assume the range maximum is the maximum TCP Wait Queue size in the histogram.

For each step of the dataset, M3R values were generated and separated into congested and uncongested subsets, and a box plot was made for each subset as shown below. In the box plot, the median is marked by the red line, the upper and lower quartiles set the edges of the box and the whiskers extend to the highest and lowest outliers. Clearly M3R does an excellent job of distinguishing between the congested and uncongested case. There is a small amount of overlap between the congested and uncongested sets around M3R = 1. The upper quartile of the uncongested subset and lower quartile of the congested set were averaged. This effectively is the center of the unpopulated region between the 2 clusters of M3R values. If the above hypothesis was correct that value should be close to 1. If all steps are included this value is 0.819. This result is improved to 1.05 after compensating for outlier data (see section 3.4.2).

Figures 12&13 M3R Box Plots for every step of every test in the dataset.

Note Congested M3R has 2 values above 5000 which are excluded from this chart



Similarly, we can be look at the predictive power of 1 as a dividing threshold for M3R. Across the entire data set of 158 steps there is 1 uncongested step with an M3R over 1 and 10 congested steps with an M3R under 1. This represents a false positive rate of 0.63% and a false negative rate of 6.33%.

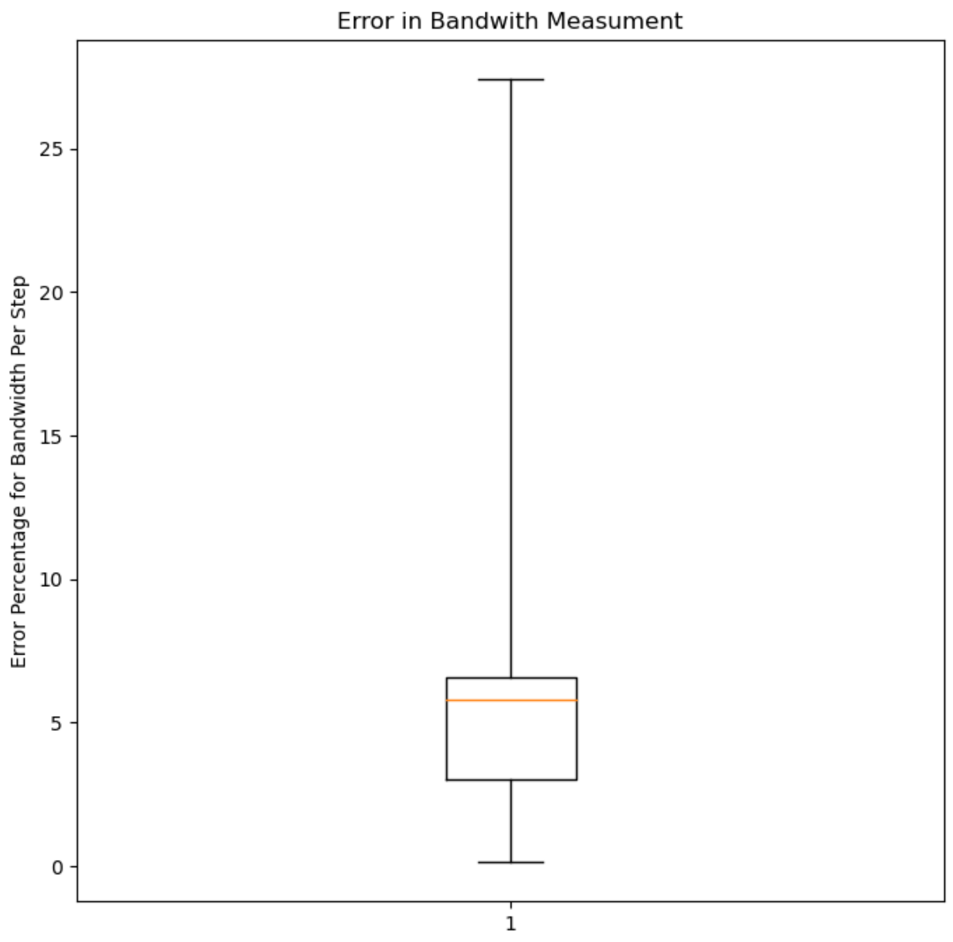
Table 2, True False Positive Negative Count for Congested and Uncontested Guessing, All Steps, M3R = 1

Ture Pos	True Neg	False Pos	False Neg
75	72	1	10

3.2. Bandwidth Prediction During Periods of Congestion

It is assumed that the amount of data transmitted during congested steps is the maximum capacity of the link. By taking the average bps of data transmitted during congested periods we can make highly accurate estimates of the maximum bps of the link. The mean error in all estimates of link capacity via this method is 5.8%. Even considering outliers, less than 10% of measurements have an error rate above 10%.

Figure 14 Error In Bandwidth prediction for every step of every test in the dataset



3.3. Use of R-Win to Predict the Cause of Congestion

The TCP receive window tells the sender how much data the receiver can ingest. Thus, over any given step of the experiment, mean receive window size can be used as a signal for whether the receiver is overwhelmed. We use this to determine whether congestion is due to the receiver being unable to ingest more data or if it is due to the network itself.

3.3.1. Scale Factors

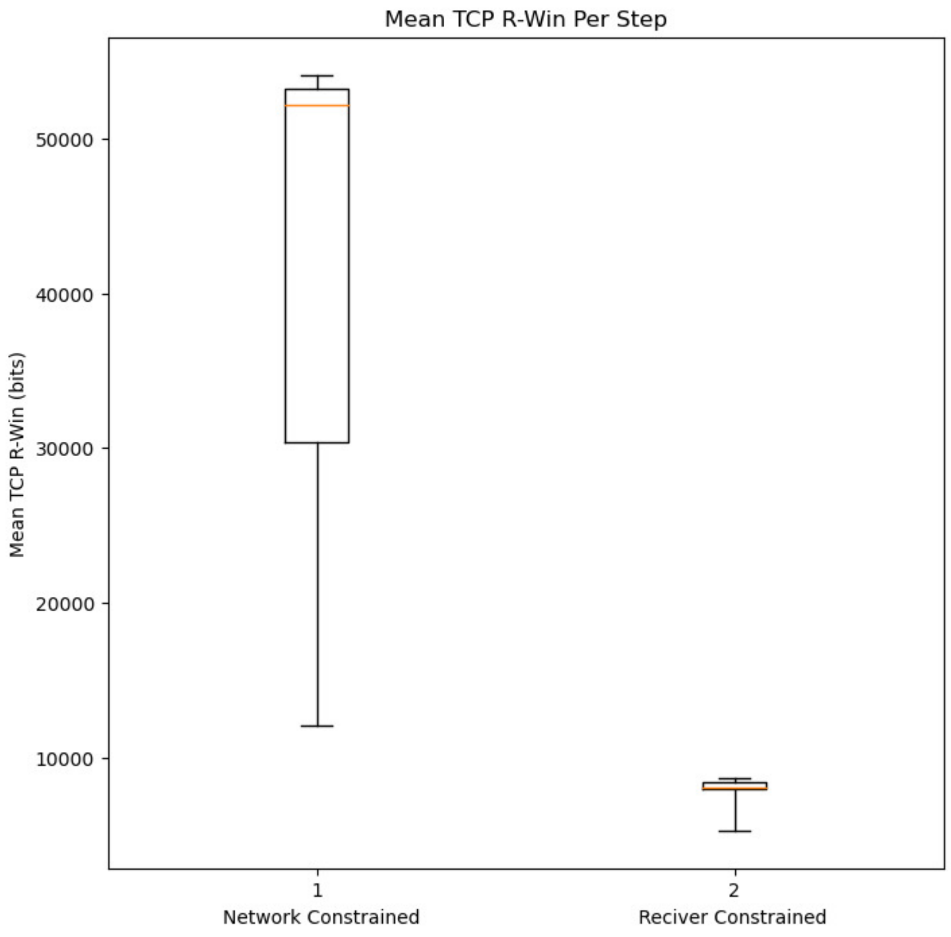
The maximum receive window size on any given TCP connection is negotiated during the initial TCP connect and acknowledgment sequence. TCP negotiates a scaling factor for the received window size. The scale factor is remembered for the duration of the connection. Each packet contains a receive window size that both sender and receiver understand is to be multiplied by 2 to the power of the scale factor (Calculated Window Size = R-Win * (2^Scale Factor)). This is often called the calculated window size. A complication is that the negotiated maximum window size does not necessarily correspond to the maximum receive window size used in practice.

For example, the 2 devices in this data set always have a scale factor of 6 thus the maximum possible calculated window size for every file in this data set is 4194240 bits or 4.19mbits. However, the maximum observed calculated receive window size is 2072832. Receivers will regularly negotiate receive window scaling factors that allow for a possible maximum value massively outside the range of anything they will ever request. As a side note, on Linux systems, both the preferred scale factor and the maximum requested size are configurable.

3.3.2. TCP Receive Window Size Separation

For our test runs, we kept track of the maximum window size that occured over the history of the connection. Below is box and whiskers plot of the mean TCP R-Win size for each congested test run step. The data are separated into steps whose congestion was caused by the network or limitations of the sender and steps whose congestion was due to the receiver. The former all had receive window sizes in the upper four quartiles of the range for that connection, while the latter all had received window sizes in the lowest quartile of the historic range for that connection. The two subsets did not overlap on a single sample.

Figure 15 Mean TCP R-Win Size Separated By Constraint, for Every Step of Every Test in the Dataset



Based on this data, it is clearly possible to determine whether congestion is caused by an overwhelmed receiver or the network/sender in all congested cases.

3.4. Outlier Data

Table 3 is a full list of all outlier steps ordered by test name and occurrence. There are two classes of outlier data present in the data set. A single step that does not conform to this congestion prediction method and many steps in the most attenuated Wi-Fi tests.

Table 3: Outlier results

Test Name	Step Number	M3R
False negative		
tcp_ordered_final_140mbps_wifi_8_24_434pm	8	0.9847328244
tcp_ordered_final_140mbps_wifi_8_24_434pm	9	0.8084252758
tcp_ordered_final_140mbps_wifi_8_24_434pm	15	0.9265545209
tcp_ordered_final_310mbps_wifi_8_24_540pm	8	0.2706526867
tcp_rando_final_140mbps_wifi_8_24_434pm	8	0.7318718381
tcp_rando_final_140mbps_wifi_8_24_434pm	13	0.8146778275
tcp_rando_final_140mbps_wifi_8_24_434pm	19	0.8030797101
tcp_rando_final_140mbps_wifi_8_24_434pm	9	0.8135506003
tcp_rando_final_140mbps_wifi_8_24_434pm	20	0.8034188034
tcp_rando_final_140mbps_wifi_8_24_434pm	17	0.9729355181
False positive		
tcp_rando_final_140mbps_wifi_8_24_434pm	5	1.422845691

3.4.1. Non-Conforming Step

An interesting entry in the above table is step 8 of test tcp_ordered_final_310mbps_wifi_8_24_540pm. This step that should be congested but has an M3R of less than 0.3. This step exhibits less variation in TCP Wait Queue size than its surrounding neighbors. Either the M3R method is not detecting an inherent feature of congestion (which seems unlikely given its success in other cases) or the test environment was accidentally modified for that step's, possibly via a change in Wi-Fi attenuation or a change in RF interference.

3.4.2. Highly Attenuated Wi-Fi

All remaining outlier data occur in tests tcp_ordered_final_140mbps_wifi_8_24_434pm and tcp_rando_final_140mbps_wifi_8_24_434pm. It seems the congested steps across these tests are hard to distinguish from uncongested steps (see figure 7 for a visual intuition of this). These test used highly attenuated Wi-Fi, and from the M3R perspective, they appear always congested, even when passing significantly less data than the connection is capable of transmitting.

As seen in table 3, most of these cases are congested steps with an M3R above 0.8 and below 1. This indicates a bias in highly attenuated Wi-Fi to a minor false negative on congested steps. When using an M3R of 0.8 to test congestion, no false positives are introduced. The accuracy of both bandwidth guessing and cause guessing is only mildly affected. If these tests are removed from this analysis the midpoint M3R guessing method discussed in section 3.1 returns a value of 1.05. A further discussion of the causes of these outlier data points can be found in section 4.5.2.

3.5. Results

In summary, the methods described in the above sections can predict whether a TCP receiver is congested 93% of the time, which can likely be made more accurate if known causes of outlier data under highly attenuated Wi-Fi conditions are properly controlled for. When it is determined that the link is congested, it is possible to predict the available bandwidth on the link to within 7%. Lastly, these methods can predict if the receiver is overwhelmed or the cause of the congestion is elsewhere in the network, potentially 100% of the time.

4. Continuing Research

Although this method produces consistent results for the datasets presented, additional research is needed to make this a universal method for all interface types, and arbitrary data. Additional features are needed to make inferences about other stats than just congestion and maximum bandwidth.

4.1. Prediction of the Amount of Congestion

As of now this method reports congestion as a binary value: congested and uncongested. The tool would become much more powerful if it were able to report congestion as a percentage on a known scale. Not only would it provide an immediate understanding of the amount of strain on the network, but it would also enable continuous monitoring of bandwidth as opposed to sampling only when fully congested, as described above. It is possible this could be determined by analyzing the ambiguous and uncongested cases with more sophisticated methods.

M3R has been shown to be highly predictive of congestion in this binary mode. In this dataset M3R rarely enters a range spanning approximately from 0.5 to 0.9 but it is possible that range communicates the degree to which the network is congested and by extension would allow for continuous utilization predictions.

4.2. Overlooked Cases

The following cases have not been tested and our M3R congestion detection algorithm needs to be verified for them.

- Wi-Fi 4 and 6, cellular, Ethernet at other speeds – all not yet tested but we suspect M3R will work
- Multi stream networks
- Real-world traffic -- The current results are based on artificial data generated with perf tools. It is possible that this model breaks down for real world traffic, though there is no reason to suspect this is the case.

4.3. Cause of Congestion Split Over Interface Type

Another significant limitation of this data is that the receiver was only overwhelmed with sends during the congested steps of ethernet tests while during all congested steps of Wi-Fi tests the limited network was the source of congestion. As a result, our cause prediction is split across two distinct interface types. It is recommended that further testing be done such that cause prediction can be done over a single interface type to ensure the method described in section 3.2 is not interface dependent.

4.4. Highly Attenuated Wi-Fi Cases

It is not known at this time what causes the apparent continuous congestion when Wi-Fi is highly attenuated. It is also not known how valid it is to lower the threshold to compensate for the likelihood of highly attenuated Wi-Fi producing minor false negatives. Finally, the exact point at which Wi-Fi becomes so attenuated that this effect manifests is unknown. A large corpus of highly attenuated Wi-Fi data would be useful in answering these questions.

4.5. Applications

This work has been done by post-processing captured data. In application, this method would be implemented to run continuously. The algorithm is simple enough to likely require minimal resource use. For example, on Linux devices, the most practical and efficient implementation would be a kernel module, though all the needed information is available in user space.

Once implemented, the M3R method will have numerous applications ranging from client steering to diagnosis of network conditions. The passive nature of the method will minimize its impact on device as well as the network itself.

Conclusions

The methods discussed here have enormous potential to allow continuous monitoring of congestion, its causes, and the bandwidth of connections. Congestion was correctly identified 93% of the time and there are multiple ways to improve that accuracy to above 98% are known. The cause of congestion was correctly predicted in 100% of cases. Bandwidth availability was predicted to within 5% of the known value more than 70% of the time and to within 10% of the known value more than 95% of the time. Finally, by taking the position of the sender these methods scale to multi-client scenarios, without encountering the hidden terminal problem.

APPENDIX A References

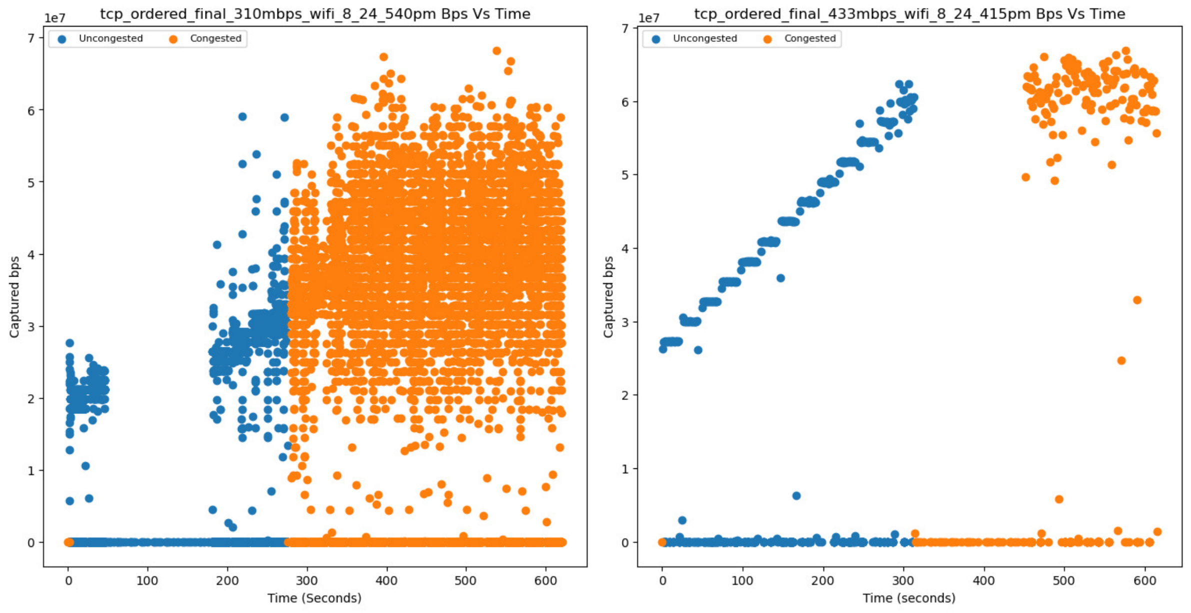
[Band Tools] Mukta Airon and Neeraj Gupta, Bandwidth Estimation Tools and Techniques: A Review, https://www.researchgate.net/publication/332874492_Bandwidth_Estimation_Tools_and_Techniques_A_Review_Mukta_Airon_orcidorg0000-0002-0637-0382_and_Neeraj_Gupta_orcidorg0000-0002-9650-4988, Preprint, October 2017, K. R. Mangalam University, Gurgaon, Indi

[MatPlotLib Boxplot] matplotlib.pyplot.boxplot Documentation, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

APPENDIX B Removed Steps

During test tcp_ordered_final_310mbps_wifi_8_24_540pm 65% step (3rd step) and test tcp_ordered_final_433mbps_wifi_8_23_415pm 120% step (14th step) iperf retries occurred and as a result no traffic was run for extended periods. These steps were omitted from the following analysis for consistency's sake.

Figure 1&2 bandwidth scatter plot of test with retry steps


tcp_ordered_final_310mbps_wifi_8_24_540pm Bps Vs Time
1e7
Uncongested
Congested
Captured bps
Time (Seconds)
tcp_ordered_final_433mbps_wifi_8_24_415pm Bps Vs Time
1e7
Uncongested
Congested
Captured bps
Time (seconds)