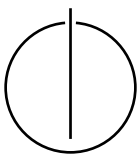
SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY —
INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Analyzing Transport Layer Characteristics of Crowdsourced Speed Test Data

Leonardo Kubilay Özuluca



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

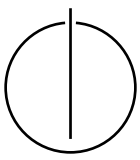
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Analyzing Transport Layer Characteristics of Crowdsourced Speed Test Data

Analyse der Transportschichteigenschaften von Crowdsourced Speed Test Daten

Author:	Leonardo Kubilay Özuluca
Supervisor:	Prof. Dr.-Ing. Jörg Ott
Advisor:	M.Sc. Justus Fries
Submission Date:	October 31st 2024



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, October 31st 2024 Leonardo Kubilay Özuluca

Acknowledgments

For my partner Pauline, who was by my side whatever happened; my friends who supported me whatever I did; and my family, without whom I could have never come this far.

Abstract

The Measurement Lab (M-Lab) Network Diagnostic Tool (NDT) dataset provides Internet measurements and presents them in an aggregated form in their database. For this thesis, we explored how M-Lab data is sampled and found that the sampling is not done distribution preserving and, therefore, introduces problems for data analysis based on the NDT data. Since M-Lab captures data on the server-side, one aspect the data does not contain is information on the client-side congestion control algorithm. While the expectation based on previous work is that most clients on the Internet use Cubic congestion control, we were unable to find a set of signals on the data that would allow inferring client-side congestion control. Inference by comparing throughput graphs, statistical analysis of RTT smoothness and packet loss estimations did not work well enough to make reliable predictions on their own. However, it could provide valuable insights when provided with additional data.

Contents

1 Introduction

1.1 Motivation

The Internet is a global communication network that has become ubiquitous in everyday life [41, 47]. However, understanding its specific intricacies and how things work within this black box of a network can be tricky. Different parts of the Internet belong to different organizations, different autonomous systems have different characteristics, and each and every device on the Internet might be different. Improving the network, its performance and its stability can only work if we take careful measurements and evaluate them. One such evaluation is being done in this thesis. For that, we use data from the M-Lab program, which takes worldwide network measurements with their NDT speed test protocol. While most crowdsourced internet speed measurements do not publicly make their data available, M-lab is a rare exception. However, even though M-Lab can provide a comprehensive data set due to its integration with Google Search, the dataset has both methodological and data quality aspects that may need improvement.

This thesis will look at the data in two different ways. In the first half of the thesis, we will take the data as mere values, and in the second half, we will look at the meaning within the data. The reason for that lies in how the M-Lab stores their data. They sample and preprocess it and present these sampled and preprocessed data parts. Our goal in the first half is to examine these sampled and preprocessed data views and compare them to each other to infer information about the sampling processes as well as the way the data is saved. We will look at specific common problems that datasets have, like duplicates and other mismatches, but also at more sophisticated comparisons like comparing the statistical distributions of values like Round-Trip-Time (RTT)s and Sender-Congestion-Window (SndCwnd) to find out problems or patterns within these sampling processes of the M-Lab project. In the second half of the thesis, we will use the data with their actual meaning and use them to infer information about the connection between client and server. We are mainly interested in two aspects. The first one is that we want to infer information about the behaviour of the connection. How it starts and ends, but also what limits the data flow. In the later part, we will then concern ourselves with the congestion control algorithm (CCA) that was used for sending the data. We know the CCA of the server, but finding ways to infer information

about the CCA of the client can be very difficult and exploring ways we could achieve that is the goal of the last section.

1.2 Research Questions

RQ 1: How is Measurement Lab sampling the speed test data and is the sampled data suitable for analysis?

When sampling data, it is essential to sample them in a distribution-preserving way. Otherwise, test results based on the data could be flawed. Testing whether this is done correctly and whether we have problems with the samples, like duplicates, length differences, or other problems, is going to be part of answering this question.

RQ 2: Is the speed test running into transport or application layer limitations?

The behavior of the connection can give us valuable information about the quality of the speed test results and gives context to the data we see. Having the connections limited might undermine the quality of the results and the corresponding data or pose problems for the general testing methodology.

RQ 3: Does the data contain hints of congestion control mechanisms that could be used to uniquely identify the client-side congestion control algorithm?

Inference on congestion control algorithms can already be quite challenging with sender-sided data, but here, we try to infer a congestion control algorithm based on receiver-side data. Testing whether we can do this in general and especially with the data at hand from the M-Lab project is necessary to answer this question.

1.3 Contributions

In this thesis, the main emphasis lies on analyzing the available data and exploring ways to use it. We developed a data pipeline for efficient data retrieval and preprocessing. For the analysis of the data quality, we used statistical measures to compare different datasets and were able to identify differences, such as duplicates or samples that were too short. We programmed queries to get information about the limitations of the speed tests and evaluated them for connection behaviour. We devised three methods to infer information about the client-side congestion control algorithm. All three of them did not work for us, but they can provide helpful information for future research.

1.4 Outline

In the following chapters, we present the findings of our analysis and go through the things we found to answer the research questions. We will start in the next chapter by presenting related work with similar or related research, comparing and differentiating our approach from theirs. After a background section discussing prior knowledge necessary to understand the analysis process, we will start with the statistical analysis of the data sampling in Chapter 4. Later, in Chapter 5, we will look at the connection behaviour and evaluate the data concerning its limitations. In the end, we will try out three approaches for estimating congestion control in Chapter 6 before we draw a general conclusion about the thesis and discuss possible future work in the final chapter, Chapter 7.

2 Related Work

Various prior work analyzed and worked with congestion control and the M-Lab data. In the following, we will look at other papers that did research related to these topics, explain them, and differentiate them from our approach to clarify the context in which this thesis was written.

2.1 Related Work for NDT Analysis

In the first half of this thesis, we will mainly concern ourselves with the quality of the data, especially the data sampling process of the M-Lab NDT dataset. Unlike the congestion control part, where there is a lot of related work, we do not know of any work related explicitly to the sampling of the M-Lab data. There are some papers where the authors describe the M-Lab project as well as the data like the one by Phillipa Gil et al. [14] or the one by Constantine Dovrolis et al. [12] where the authors also explain the bias that can be found in the data. However, specific testing with statistical tests for the quality of the M-Lab sampling process like we do are not included. There is a paper comparing the NDT data with the Ookla speed test data by Kyle Macmillan et al. [24] that uses statistical tests and tests various aspects of the data like daytime influence, client accuracy depending on the used device and the measured speed of the speed test, which we do not test for. However, this paper uses a different testing methodology than we do, as they let the NDT and Ookla client and server code run on local machines to conduct their tests. Other than that, there are also papers that try to provide context to the data. One example of this is the paper written by Udit Paul et al. [43] that presents the NDT data contextualized with technical data like access link type or device type, while the paper "Characterizing Internet Access and Quality Inequities in California M-Lab Measurements" also by Udit Paul et al. [42] contextualizes the NDT data with geographic and demographic data as well as considerations for Covid19. All of these approaches use the NDT data without testing the sampling process in the data itself. While we do not contextualize the NDT data with external data, we combine different data views from the NDT dataset to test the different data views with and against each other.

2.2 Congestion Control Related Work

In the later part of the thesis, we will explore possibilities for inferring the clients CCA from server-side data from the M-Lab NDT dataset. So, we generally try to find the senders CCA based on receiver-side captured data. Inference on CCAs is nothing new in Internet and network research. Two recent tools for CCA identification are the CCAnalyzer by Ranysha Ware et. al.[54] and Nebby by Ayush Mishra et al. [38]. Both of them can identify even the new BBR algorithm and perform exceptionally well on the global Internet. For this, the CCAnalyzer uses a middlebox that artificially creates a bottleneck where measurements can happen. Nebby uses a testing point, where artificial delay is brought into the data transfer to measure the sender's reaction. However, as both of them need some kind of middlebox during the testing process, both of these approaches describe a fundamentally different scenario than what we have because our data comes from crowd-sourced tests that already happened based on the M-Lab infrastructure that we can not change. Therefore, both of these tools try to do the same as we do but fundamentally differ from the approaches we would need to take. Nevertheless, there are also approaches for CC analysis based on M-Lab data. The paper "TCP Congestion Signatures" by Srikanth Sundaresan et al. [48] uses the RTT values from the M-Lab NDT dataset to analyze and identify CCAs similar to what we will do in section 6.3. Unlike this thesis, the paper by Srikanth Sundaresan et al. trains a machine learning model based on a decision tree classifier while we rely on simpler but better explainable statistical tests. Another paper, also written by by Srikanth Sundaresan et al. [49], also uses the M-Lab data, but instead of trying to infer the congestion control algorithm, they try to infer congestion in the network and at the interconnection points between ISPs. So even though there is research that deals with the congestion found in the M-Lab data, this research either focuses on something different or uses different methodologies and therefore differs significantly from our approach.

So, in general, we try to provide novelty in our research by thoroughly testing the M-Lab data's sampling process before proceeding with the other analysis and by statistical approaches to inference on sender-sided congestion control with receiver-side data.

3 Background

The upcoming chapter will provide background information necessary or helpful for understanding the subsequent chapters. We will start by looking at the M-lab project and the NDT protocol as a general foundation for this paper. Afterwards, we will describe and explain statistical tests and measures used in this thesis. In the last part of this background section, we will provide some background about congestion control, some commonly used algorithms and the way they work.

3.1 What are M-Lab and NDT?

The M-Lab is a non-profit open source project trying to make internet testing and its data more accessible [27]. In 2008, a group of different researchers all identified the same problems when trying to research the internet. They lacked proper, widespread server infrastructure for testing, they lacked ways for collecting data together and sharing it with each other, and they lacked resources to provide the collected data and insights to policymakers and the general public [25]. Founded in 2009, M-Lab tries to solve these problems. It has a widespread server infrastructure in dozens of countries [29], performing measurements worldwide and saving them in Google Cloud Storage and BigQuery to make them openly accessible to everyone. Nowadays, they are a substantially big independent project with many different contributors and supporters from civil organizations and educational institutions to private companies like Google, Mozilla or Cloudflare, all of which have a relationship to the internet and are interested in researching and improving it [34, 27]. According to their information, their database now contains the largest open dataset about Internet performance [27]. Moreover, it is fully open to the public and everyone interested. However, the data is limited by a processing limit of 2TB per user per day.

To measure the speed of a connection M-Lab uses their NDT. A client trying to test their internet speed gets the closest server assigned by the M-Lab and connects to it via TCP connections. The server logs and measures the connection's speed and sends them in the end to the client to see and to the Google Cloud Storage to store. From there, it will be processed further and then saved in a BigQuery database [27, 28, 22].

3.2 Statistics and Testing

When working with data, statistical tests are a powerful tool to test hypotheses in a mathematically provably way [50]. Because this thesis is about an analysis of data, a reader will be confronted with the names of several different statistical tests. To make understanding the following chapters easier, we want to explain the most important statistical tests for this thesis.

3.2.1 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov (KS) test is one of the most important tests for this thesis, as it lets us compare two samples and test the hypothesis that they are from the same distribution. We will look at a geometrical intuition of where the test value comes from. After conducting the test, we get two values, the D-value and the p-value. The p-value

is mathematically derived from the D-value and is the tool we will use to interpret the results. However, to understand how the test generally works, we will look at how the D-value is derived. This D-value is defined as the maximum absolute distance of the cumulative distribution function (CDF). Every distribution has a probability distribution function (PDF) and a CDF. The PDF is the function that plots for each value how high its probability is. Upon taking the integral of it, one gets the CDF, which describes for each value x how high the probability is to get a value of smaller or equal x. This is also visualized in figure 3.1 for the normal distribution.

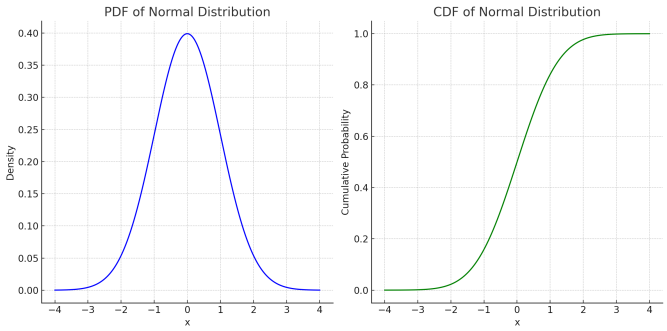


Figure 3.1: Illustration of PDF and CDF.

If we have two samples, we will get two PDFs and two CDFs. Out of these two CDFs, we take the maximum absolute difference. This is visualized in figure 3.2.

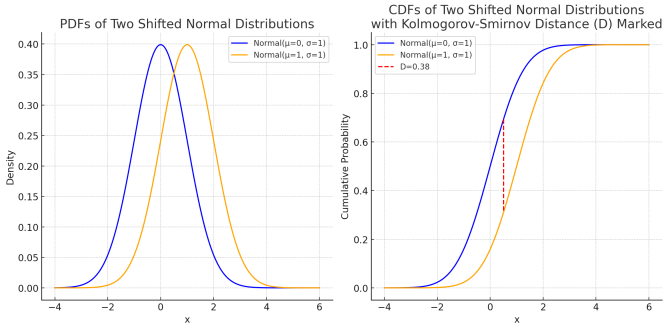


Figure 3.2: Illustrations of KS test distance metric for two distributions.

The KS test is a great tool for comparing distributions as a whole as it takes not only the form but also the shifting of data greatly into account. It is, therefore, rather robust against outliers, which do not change the entire distribution. This can greatly help when analysing unclean data. One drawback is that the KS test can have problems with duplicate values as they might shift or change the entirety of the distribution and, therefore, influence the results of the KS test [17, 11, 46].

3.2.2 Mann-Whiteney U Test

Similar to the KS test, the Mann-Whiteney U (MWU) test is used for comparing two samples with each other, and it helps us to augment and support the results of the KS test. Even though the MWU test does something similar to the KS test, it does it in a different way, and just like before, we will not dive deep into the underlying mathematics but instead get an intuition for how it works.

It does not use the PDF or CDF but instead a rank sum. Instead of looking at the samples as distributions like in a CDF, it looks at them in a discrete way and assigns rankings. The MWU test first combines the two datasets and sorts the combined set. When sorted, each value gets a rank for its position in the sorted list. After assigning the ranks, the values are separated again into their different samples. Now, if both samples are from the same distribution, both of them should have approximately the same amount of low-, middle- and high-ranking values on average. For both samples, the sum of all of their ranks is computed and adjusted for the size of the sample. This sum of ranks is then used in some complicated math to get to the p-value, but essentially, this sum of ranks is the backbone of the MWU test.

In figure 3.3, we can see a coloured example to explain this visually. We start with two samples, here in blue and orange and combine them. We sort the combined list

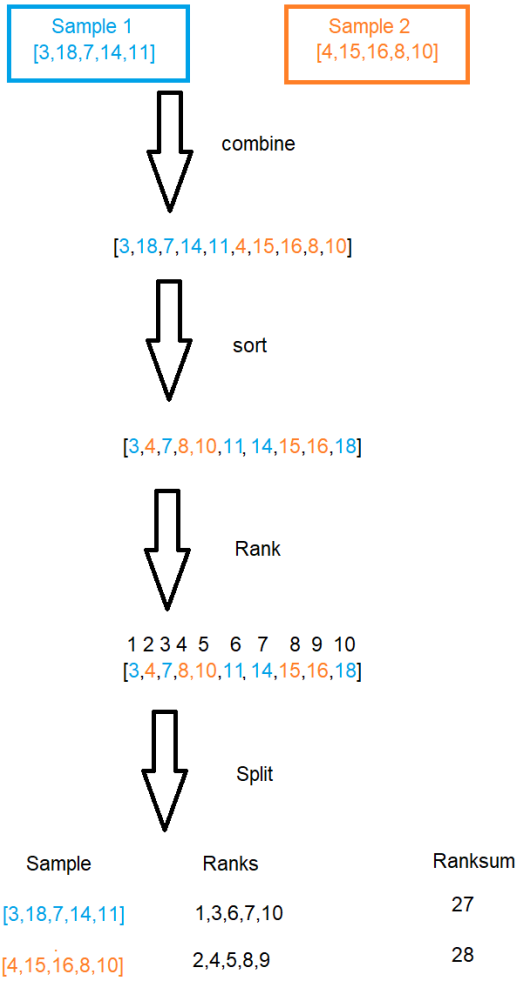


Figure 3.3: Example for calculating the rank sum in the MWU test.

and assign positioning ranks to the values. Afterwards, we split the samples again in the original separation and sum up the ranks for each sample. This way, we get two rank sums that are the basis for the MWU test to see how similar the samples are.

As the test reduces the samples to mere rank sums, it is impossible to infer where or how they differ. However, because the complete samples get reduced to ranks and rank sums, the test is robust against outliers. For our use case in the thesis, it will be used together with the KS test to test samples for similarity [3, 10, 36].

3.2.3 Binomial Test

The binomial test is a tool for testing binary data. The binomial test tests that a specific event comes with a fixed and given probability. A typical example of this would be a fairness test for a coin. A coin is either "Head or Tail", and the probability for "Head" is supposed to be 50% if it is a fair coin. After flipping the coins several times, we can conduct a binomial test to determine how likely it is for the coin to be fair. The binomial test is based on a short formula:

$$\Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

where k is how often our event happened, so how often we got "Head", n is how often we flipped the coin in total, and p is the probability we expect under a true hypothesis; in the case of a fair coin, this is 0.5. With this formula, we get the probability of the event happening k-times. By summing up over different k, we can get the probability of having a result at least as extreme as k. This probability is what we use in the binomial tests for decision-making. [45, 9, 40].

3.2.4 Shapiro Test

Testing for a normal distribution of data is often necessary, as many statistical tests require normally distributed data for testing. Generally, there are two common ways that normality is tested graphically and analytically. Graphically, we can look at data and visually look for a bell curve in the plot, or we can directly plot it in a q-q graph and look for a linear graph, which would indicate a normal distribution. Even though this is a very reliable way of determining a normal distribution, it has scalability problems. For augmenting the manual visual analysis and for better scalability, there is also a way to test for normality analytically. The Shapiro test or Shapiro-Wilk test is one such test for normality and is the test we will use in this thesis. It computes a so called W-value based on a comparison of the sample values and the values we would expect

if the sample was normally distributed and uses that to compute a p-value that we can interpret just like the one of the other tests [18, 53].

3.2.5 Interpreting p-values correctly

As we are working with statistical tests, we will get a p-value as a result most of the time. As with many other things in statistics, it is crucial to correctly interpret the p-value to make sure not to make any wrong assumptions, especially because the p-value is often used as the criteria for making decisions. When testing with a statistical test, we usually have a null hypothesis. For example, two samples were both sampled from the same population. When conducting a test, the resulting p-value is the probability of seeing a result at least as extreme as the given one under the assumption of a true null hypothesis. So when testing two concrete samples with a KS test, the resulting p-value represents the probability of getting two samples that are at least as different as the two given ones under the assumption that they are from the same population. If this probability is very low, we can safely assume our assumption is wrong. This does not mean that it really is wrong, but just that we see in the data evidence that made our assumption very unlikely, so we reject it in favour of the alternative hypothesis. This also does not mean that the alternative is right, just that we saw evidence in favour of it. When using the p-value to make decisions, we rely on a significance value. This value acts as a threshold, below which we reject the null hypothesis. Commonly used values for this threshold are 0.01, 0.05, and 0.1, depending on the criticality of the application. In our TCP data analysis, we will use a significance value of 0.1 and 0.05. Therefore, any test with a p-value below 0.05 or 0.1, respectively, rejects the null hypothesis in favour of the alternative.

3.2.6 Modality

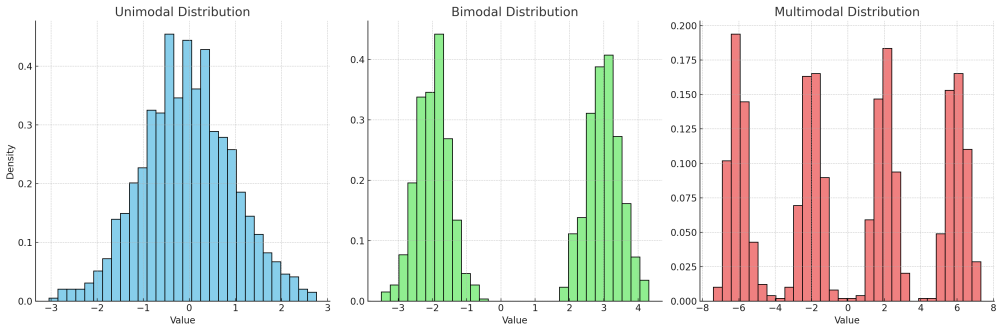


Figure 3.4: Example for unimodality, bimodality and multimodality.

Modality in statistics refers to a characteristic within the data concerning its distribution. Data can often be found clustering around specific values. The number of these bundles we have in the data is called modality. In easy terms, one could say that if we have one bundle of data, it is unimodal; if it is two, it is bimodal; and if the data is spread out in several clusters, it is called multimodal. An example of that can be seen in figure 3.4.

3.2.7 Boxplots

A boxplot is a visualization tool that is designed to represent an different aspects of a distribution in a straightforward manner. It consists of just three components: a box, a red line within the box, and whiskers at the top and bottom. The box contains the middle 50% of the values, while the whiskers extend to the largest and smallest values, each containing the top and bottom 25% of the data. The red line represents the median, the value that sits in the middle of the dataset with half of the data above and half below. Any potential outliers that are significantly distant from the box are shown as dots. In figure 3.5 is an example of a boxplot that plots the data that is just the numbers from 2 to 12.

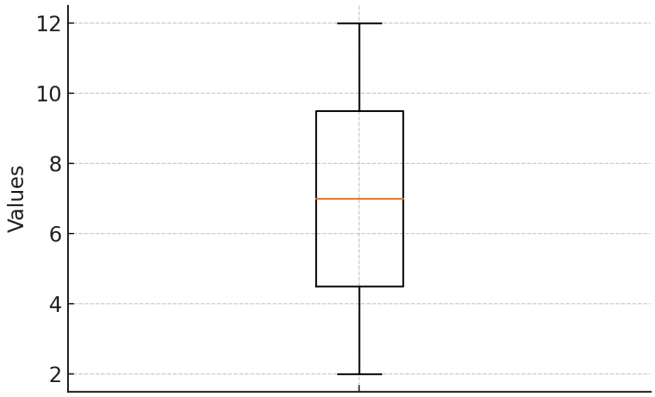


Figure 3.5: Example of an Boxplot.

3.3 Measures for Non-Statistical and Binary Data

Until now we covered how we can test and compare statistical data. The KS test uses utilizes the CDF of distributions, the MWU test uses sorting and rank sums, but what if that is not possible? Binary data for example we can neither rank nor can we compute

a CDF for it. For data where this was the case we use in this theses four different ways of comparing to samples. These four are the Hamming Distance, the Ovelap Match percentage, the maximum cross correlation and the average sliding window correlation. We will quickly go through them here so that they are clear for the rest of the thesis.

3.3.1 Hamming Distance

The Hamming Distance is a widely known metric in Computer science to compare binary data. The general principal is very easy. Given two samples or two lengths of equal lengths, we compare corresponding entries with each other. We compare the first entry with the other lists first entry, the second entry with the other lists second entry and so on. The number of unequal entries is the so called Hamming Distance [35, 15].

3.3.2 Overlap Match percentage

Overlap match percentage is a metric that compares corresponding entries in two samples or lists. It compares the first element of $list_1$ with the first element of $list_2$ and then the second element of $list_1$ with the second element of $list_2$ and so on. Every time the two compared entries are identical, we have a match. The overlap match percentage is just the percentage of these matches [6].

3.3.3 Maximum Cross Correlation

Until now we only looked at ways to compare two lists or samples as they are, we did not consider that they might be shifted to each other. The maximum cross correlation takes this into account. The correlation is a mathematical value we can compute for two lists to determine how similar they are. With cross correlation we now try every possible shift for one of the lists to see whether they get closer or not. The maximum cross correlation is the maximum of all of these correlation values we get from the cross correlation [7, 2, 8].

3.3.4 Average sliding window correlation

The previous methods all looked at comparing the entire lists with each other, but sometimes we have lists that match insanely well for some parts and slightly bad for other, but simply checking the entire list might lose this information. With the average sliding window correlation we can check with a window just a small part of the list and calculate a correlation. We do this for all of the possible placements of the window and average them out. This way we get the average sliding window correlation [55, 52].

3.4 Congestion Control

The word "congestion" in the modern world is often used to describe traffic jams. A condition where too many people try to use a road with their cars, which results in no one being able to get the result they expected: fast arrival. In the context of computer networks and the internet, congestion happens if too many network packets are sent via the same link, and the result is similar to traffic jams: everything gets slower. If the congestion is very bad, we will even see packet loss, something that happens because the capacities of buffers and memories are finite and limited. This was already a problem in the very early days of the internet in the mid-1980s, so in the year 1988, Van Jacobson, at that time a scientist at the University of California, published a paper with the title "Congestion Avoidance and Control" that is the foundation of internet congestion control up to this day [20]. Even though a lot has changed since the 1980s, the general concept of congestion control remains the same; we try to detect congestion on the network and adapt our sending rate to avoid congestion. Nowadays, two of the main congestion control algorithms in use are Cubic, a representative of the loss-based CCAs and BBR, a relatively new approach that brings many advantages with it. In the following sections, we will describe what their most prominent characteristics are and how they behave so we can distinguish them later in the thesis.

3.4.1 Important Words and Phrases

Word	Explanation
Packet	A packet is a bundle of data that is sent via the network.
Acknowledgment	An acknowledgment is a type of message that a receiver gives a sender to signal that a packet successfully reached them.
Acknowledgment Number	The number sent with the acknowledgment to signal how many bytes were received in total and what bytes are expected next.
RTT (Round Trip Time)	The Round Trip Time is the time between sending a packet and receiving the reply for the packet, i.e., the time for an entire round.
BiF (Bytes in Flight)	The Bytes in Flight are the amount of bytes that are currently sent but not yet acknowledged.
SndCwnd (Sender Congestion Window)	The Sender Congestion Window is the maximum number of bytes that can be in flight without knowingly risking network congestion.
MSS (Maximum Segment Size)	The Maximum Segment Size is the maximal size the transport layer data and its payload can have.
ISP (Internet Service Provider)	The company that provides peoples and companies with their internet access and handles the upstream connectivity.
ECN (Explicit Congestion Notification)	An ECN is a bit that can be set in a pcket sent by a network entity to inform a sender manually about the existence of congestion without him having to infer by himself.
Pcap (Packet Capture)	A file in which the network packets of a connection can be stored.

3.4.2 Loss-based and Cubic

Loss-based CCAs were among the first to be invented, with TCP Tahoe being invented in 1988 and TCP Reno shortly after in 1990. The concept is generally straightforward: if we detect loss, we reduce the SndCwnd, if not, we increase it. What sounds simple as a concept becomes more difficult in practice. On the one hand, we want to use as much bandwidth as possible as much of the time as possible so that we do not waste network resources; on the other hand, we want to avoid network congestion as well as possible to increase throughput. To do that, loss-based congestion control algorithms generally divide the connection into two phases: a slow start and a congestion avoidance phase. In the slow start phase, the sender exponentially increases the SndCwnd with each

received acknowledgement until packet loss occurs to test out the available bandwidth in the network. Once packet loss occurs, the SndCwnd is halved, and we continue with the congestion avoidance phase. In this phase, we try to use as much bandwidth as possible while avoiding congestion, so we increase the SndCwnd with every RTT a bit to use potentially unused bandwidth, but not as extreme as in the slow start phase because we want to avoid congestion. If packet loss eventually occurs in the congestion avoidance, we will reduce the SndCwnd again abruptly and significantly before raising it again slowly. This will result in a shark tooth pattern as visible in figure 3.6.

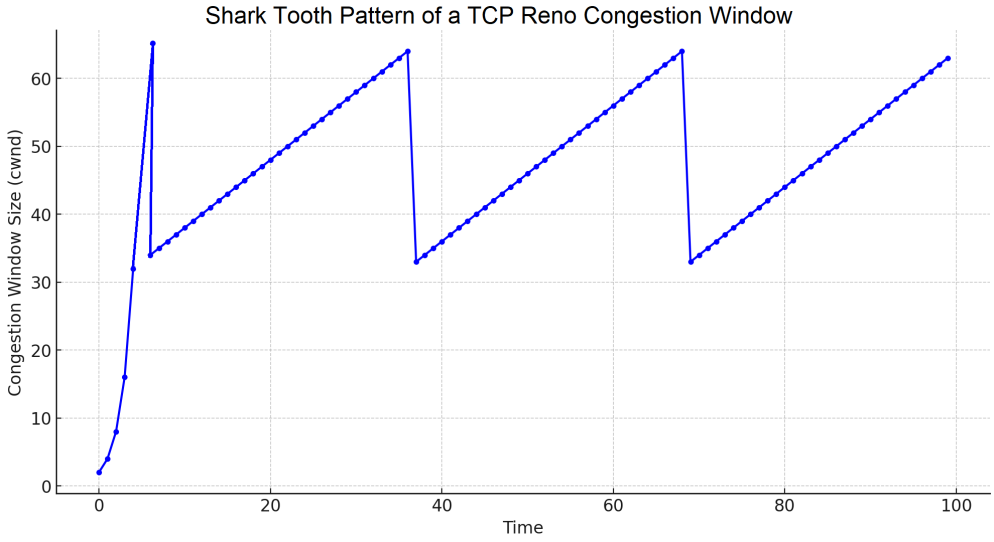


Figure 3.6: Common TCP Reno Cwnd development with shark tooth pattern [1].

One of the most common loss-based CCAs nowadays is TCP Cubic. Since 2006, it has been the default CCA for the Linux kernel and is widely used on the Internet [4]. The most significant change for us from the aforementioned TCP Reno is it using cubic functions instead of linear functions in the congestion avoidance phase. This helps increasing the throughput and its ability to use unused bandwidth. Unlike Reno, Cubic also keeps track of the last congestion event that happened and the SndCwnd at that time, the Wmax, and only keeps rising until that point most of the time. It rises above that point only sometimes to test out potential open bandwidth.

3.4.3 BBR

As loss-based CCAs rely on actual packet loss to react to congestion, they fill the network buffers along the way to the brim until one packet gets lost. They operate,

therefore, with full network buffers and high RTTs. This is not only bad for the throughput but also increases latency. To avoid this, BBR uses an entirely different approach, model-based congestion control [5]. Instead of waiting for packet loss to happen, BBR tries to estimate the Bottleneck Bandwidth and the RTT and maximize throughput while keeping RTTs low and buffers empty. How exactly it does that is not important for this thesis. However, generally, one can say that BBR tries to estimate the RTT and the bandwidth at the bottleneck of the connection and adjust the amount of data in flight to operate as close to the optimum as possible. Therefore, we would not see a shark tooth pattern in the throughput of a BBR connection.

3.5 Other Transport Layer Concepts

3.5.1 TCP Info and TCP storage in M-Lab

When connecting two entities on the internet, e.g. a server and a client, we often use TCP. To function correctly, TCP collects and computes a lot of information about the connection. This information is saved in the TCP info struct [21, 44, 37]. The M-Lab NDT servers use TCP connections for the testing process and takes a snapshot every 10 milliseconds. These are afterwards thinned out by a factor of 10 to 1 while ensuring the last one is always included [23] and used as the basis of the `tcpinfo` view in the BigQuery database of the M-Lab project [23, 30, 33].

3.5.2 Receive Window Limitation

As we have different machines on the internet with different hardware and software and, therefore, different capabilities when handling incoming data, ensuring that we do not overwhelm a receiver is a crucial part of internet communication. To accomplish that, both parties, in a connection, communicate their so-called receive window, which is the maximum amount they can receive at a time. If the throughput and the sending rate are artificially kept low by the sender in order not to exceed the receiver's receive window, we call the connection receive-window-limited (rwndlimited) [30, 21].

3.5.3 Application Limitation

When establishing a TCP connection between two parties, the data that is being sent is often produced or handed down by the applications that run on a machine. If the throughput and the sending rate of a sender is capped by not getting enough data to sent by the application, we call this connection application-limited (applimited). This can indicate that a connection is affected by a non-network related bottleneck [30, 21].

4 The Data and Its Sampling Process

In this part of the work, we will look at what data is stored by M-Lab, its quality and the way it is being used and sampled. We look at different parts of the datasets where the same information about the connections is stored and that contain samples, as well as compare differently sampled data to infer information about the sampling.

4.1 The BigQuery Database and Its Different Views

After the raw test results are processed, the test results are saved in the BigQuery database. In this section, we will have a closer look at this database, its views and the data stored within it.

The M-Lab does many different tests, like NDT, Measurement Swiss Army Knive (MSAK), Paris traceroute and other tests. As we will use the data from the NDT tests, we will only look at the database parts connected to this test. We will only look at the newest version, NDT7, and ignore the older protocol versions' data. In figure 4.1, there is a schema of the important data views for this thesis.

In general, the NDT data is sorted into three different parts, `ndt_raw`, `ndt_intermediate` and `ndt`. The `ndt_raw` folder contains unsampled, unfiltered and nearly unprocessed data. It is supposed to mirror the raw pcap and test data as close as possible. The `ndt_intermediate` contains the extended datasets. These contain each and every row from the raw datasets in `ndt_raw` but are extended by additional information that the M-Lab knows about the test, like metadata about the client and server or filter metrics, like whether the test had completed or had any errors. The `ndt` folder contains filtered data from the `ndt_intermediate` extended views using the filter metrics added to filter out bad, defective tests and leave only ones that are supposed to have worked properly.

The main data we will analyze will be the `ndt.ndt7` and the `ndt.tcpinfo` data views [33, 32]. The extended views we will use for additional checks on whether a test is valid. From the `ndt_raw`, we will only use the pcap view, which provides links to the raw pcap files in the Google Cloud Storage (GCS).

The different views in `ndt_raw`, and `ndt_intermediate` are never sampled. They

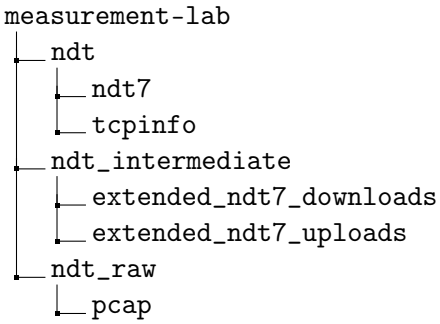


Figure 4.1: Directory tree of the BigQuery data views that are relevant for us.

contain the complete data. What is interesting, however, is the structure of the `ndt7` view. It contains a "raw" section that includes TCP info snapshots. One would typically imagine that the TCP info snapshots in the `tcpinfo` view are the same as the TCP info snapshots that are in the raw section of the `ndt7` view, but this is not the case. The same speed test has two different samples of the TCP info snapshots. The quality of this sampling and whether or not we can see any patterns or irregularities in this sampling process will be the main topic of this chapter.

4.1.1 How Much Data Do We Have?

As we are limited to 2TB of data daily, we only used `ndt7` and `tcpinfo` data from 01 December 2023. To ensure we still have enough data to run statistical tests we counted them with a SQL query. The results can be seen in table 4.1. We have 14.7 million speed tests in just that one day. This is large enough to prevent the risk of bias due to the lack of data. More interesting are the numbers for download and upload tests. For some reason, we do not have matching numbers. Even though we should have equal amounts of upload and download tests, we see 8.4 million download tests and only 6.3 million upload tests. So we have more download than upload tests. However, this is no problem for our analysis as we never rely on absolute numbers but rather fractions of the data having a specific characteristic.

Table 4.1: Number of download and upload tests on 01 December 2023.

Type	**Number of tests**
Total	14,709,944
Download	8,419,491
Upload	6,290,453

4.2 The Google Cloud Storage and the Raw Pcap Files

The GCS is a Google service used for automatic and large-scale file saving and backup. In the case of M-Lab, the GCS saves the raw and unprocessed data. Unlike the BigQuery database, the GCS saves without redundancies or samplings, as this is simply used for saving the raw data [31]. The raw data that is the most relevant for us is the collection of raw packet captures. They are captured by the server while conducting the test. How they are saved in the GCS, how we will preprocess them for the upcoming statistical tests, and how much data we have are the topics of this section.

4.2.1 The Pcap Preprocessing Pipeline

To turn the pcaps into a format that is easily and efficiently machine readable for when the actual statistical testing is done, we need to preprocess them.

When downloading the data from the GCS, we first get ".tar" files that combine several pcaps compressed. The tar files usually consist of 30-250 pcap files (depending on the size of the pcap files) that each represent one test. These need to be unpacked first. After that, we filter out problematic tests that had errors or stopped early based on the filter flags from the extended views in the BigQuery database. With this, roughly 40-45 percent of tests get filtered out. The remaining packet captures get further processed by the data pipeline. A Python script goes through the entire pcap files, separates the "client to server" and the "server to client" packets from each other and saves for each the acknowledgement numbers, the RTTs and the Bytes in Flight (BiF) as well as the maximum RTT and BiF for each. It also computes the duration of the entire test based on the time difference between the first and last packet. For better machine readability and for saving space these get saved as a compromised parquet file for later processing in the statistical testing pipeline.

4.2.2 How Much Pcap Data Do We Have

Regarding the raw packet captures, there is about 3TB of data for each day. This is infeasible to process in this thesis. Instead, we focused on a representative sample of the data. To ensure that we do not skew our results with a potential location bias, we take a sample of 1GB of data from each server site to ensure that each location is included in the analysis. As there are 71 server sites, this results in around 71GB.

4.3 The Testing Process

As we saw in section 4.1, different parts of the M-Lab databases contain the same data parts for the same speed tests. These are the TCP info snapshots that are saved as different samples in the `tcpinfo` view and the raw section of the `ndt7` view. When we sample data, so when we take a smaller subset of data for each speed test, we have to make sure to take a representative sample that does not destroy the distribution of values. Whether the sampling is done, distribution preserving or not, how well this is done, and what kind of patterns we can see are what we will try to test. So, in the following, we will take the TCP info snapshots from both views and compare them with statistical tests to infer information about the sampling process.

4.3.1 Choosing the Right Test

When choosing the right test for testing our data, there are a few things to consider. These include the pairing of the data, the original distribution from which the samples came, and the general goal we are trying to achieve by the test. Our goal is to compare two samples with each other and see whether they came from the same distribution. The data from the `ndt7` view and the `tcpinfo` view are unpaired, and a test with a Shapiro test revealed that the underlying data is not following a normal distribution.

So, we need a test that can compare two different datasets for their statistical distribution and handles unpaired data that is not normally distributed. With this in mind, the best test for our use case is the so-called KS test. This test takes two samples, calculates how different they are based on their CDF and tests the assumption that they are sampled from the same population. It takes into account the overall distribution and compares them, which makes it a good fit for our use case. In practice, it turned out that making decisions based only on the KS test was difficult. So, to support this, we can take the Mann-Whiteney-U test. This test also takes in two distributions and tests them under the null hypothesis that they have the same distribution. Unlike the KS test, the MWU test does not use the CDF but instead uses a rank sum of the dataset to compare the central tendencies of the data. These tests together form the foundation of our testing.

4.3.2 Structure of the Testing Pipeline

As discussed in the previous sections, the best tests for our use case are the KS test and the MWU test. We have an array of values for each speed test from the `ndt7` view and one from the `tcpinfo` view, which we test against each other with the KS and MWU

tests. The result will be a p-value for each of the speed tests. Each of these p-values gets used to test the hypothesis of whether the two samples are similar in their distribution. These decisions get aggregated by a binomial test to condense them to one p-value for the general hypothesis that the entire sampling process was generally done distribution preserving. This is also visualized in 4.2.

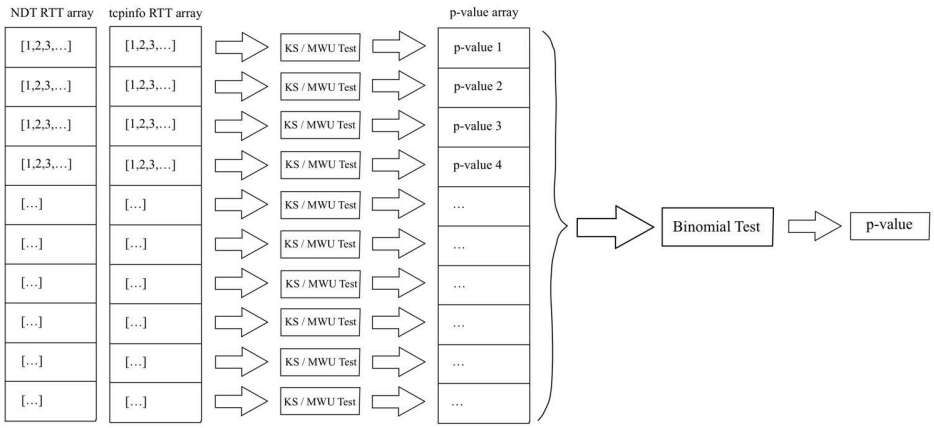


Figure 4.2: Schema of the Testing Pipeline with the example of RTT measurements.

Do we need Correction terms?

In statistics, correction terms are an essential tool to ensure that tests give correct results when e.g. data is too small, or assumptions do not always hold. In our case, we have a pipeline consisting of several statistical tests, a case in which commonly correction terms like the Bonferroni correction [16] are applied to ensure that the tests do not bloat the false positive rate. Nevertheless, our pipeline is comparably small. We are doing at maximum two tests in a row (either a KS test and then a binomial test, or a MWU test and then a binomial test), which is so small that the impact on the false positive rate should be marginal. As our test results are very clear and do not seem to have any problems with a false positive rate, it was decided not to include a correction term [13].

4.3.3 Extra Tests for Applimited Testing

When testing the data that we have, we were always able to compute distributions and compare them. A list of RTTs can start with low RTTs and have them increase over

time as buffers might fill. However, for the applimited data, this is not the case. The applimited data is just a list of ones and zeros. There is no distribution in it. This makes it effectively impossible to use the tests we used before, the KS test and the MWU test, to test this data. However, for binary data like this, there are other tests that we can use. The general testing pipeline stays the same, the only thing we exchange is the KS/MWU bracket in the schema 4.2 and exchange it with what we call a combined similarity metric. We did, in general, different tests for the datasets and then weighted them. We used the Hamming distance, the overlap match percentage, the maximum cross correlation and the average sliding window correlation. The first two focus on the match with the present data, while the second two also consider shifting in the data that could otherwise skew the data. These four tests each give a value that is being combined to get the combined similarity metric. This will then be used like a p-value in the rest of the testing pipeline by the binomial test.

4.3.4 Interpreting the Test Results

In the following section, we will go over the test results and their implications. The results are presented in tables with five columns, "Type", "Test", "Significance", "Rejection Rate" and "p-value". The type represents what type of data was used; the Test and Significance columns represent what test and significance level were used to test the given data. The rejection rate column describes how many percent of the speed tests in the data rejected the hypothesis that the `ndt7` views data and the `tcpinfo` views data have the same distribution. Finally, the p-value column contains the p-value based on which we can evaluate the general hypothesis of the data sampling process that it is done distribution preserving.

Table 4.2: RTT results KS.

Type	Test	Significance	Rejection Rate	p-value
all	KS	0.05	6.33%	0.0
all	KS	0.1	9.47%	1.0

In the first table, table 4.2, we can see the results for all of the test data, tested with the KS test on two significance levels, 0.05 and 0.1. We can see that the p-value is 0.0 for the first one and 1.0 for the second one, complete opposites. An identical test with identical data on two significance levels can have different outcomes, but they should at least be similar. Having the p-value completely turned is seldom. As the result contradicted itself, the test was repeated with the MWU test.

Table 4.3: RTT results KS and MWU.

Type	Test	Significance	Rejection Rate	p-value
all	KS	0.05	6.33%	0.0
all	KS	0.1	9.47%	1.0
all	MWU	0.05	6.31%	0.0
all	MWU	0.1	10.14%	0.02321300071353757

When including the same tests for the MWU test, we can see the results from table 4.3. For a significance of 0.5, we get the value of 0.0; for a 0.1 significance, we get 0.023. Both values are close to zero, so they are at least similar. However, with the knowledge that this test rounds a value down to flat 0.0 only if the number is smaller than ten to the power of -100, we know that they are still vastly different in magnitude. Expecting that our data would not be skewed and our tests would work fine, we tried partitioning the data to see whether there were subsets with different outcomes that lead to contradicting results. Most partitions don't give very meaningful data, but partitioning the data into upload and download tests gives us the result visible in table 4.4.

Table 4.4: RTT results.

Type	Test	Significance	Rejection Rate	p-value
all	KS	0.05	6.33%	0.0
all	KS	0.1	9.47%	1.0
all	MWU	0.05	6.31%	0.0
all	MWU	0.1	10.14%	0.02321300071353757
download	KS	0.05	2.84%	1.0
download	KS	0.1	5.97%	1.0
download	MWU	0.05	3.41%	1.0
download	MWU	0.1	7.12%	1.0
upload	KS	0.05	11.41%	0.0
upload	KS	0.1	14.55%	0.0
upload	MWU	0.05	10.53%	0.0
upload	MWU	0.1	14.52%	0.0

We can see a clearer picture when separating the data into upload and download

tests and evaluating them separately in table 4.4. The upload tests all have a p-value of 0.0, so they all get rejected. The download tests all get accepted with a p-value of 1.0. So, we have a clear difference between the two subsets of the data, the upload and the download tests. This marks the first result of this thesis, as we found mathematically provable statistical evidence for differences in data regarding the upload/download test type being saved. This is not just for RTT values, but also for the SndCwnd, as we can see in table 4.5.

Table 4.5: SndCwnd results.

Type	**Test**	**Significance**	**Rejection Rate**	**p-value**
all	KS	0.05	4.23%	1.0
all	KS	0.1	5.84%	1.0
all	MWU	0.05	7.64%	0.0
all	MWU	0.1	11.74%	0.0
download	KS	0.05	1.96%	1.0
download	KS	0.1	3.77%	1.0
download	MWU	0.05	3.61%	1.0
download	MWU	0.1	7.64%	1.0
upload	KS	0.05	7.53%	0.0
upload	KS	0.1	8.86%	1.0
upload	MWU	0.05	13.48%	0.0
upload	MWU	0.1	17.69%	0.0

In table 4.5, we can again see the results for the same tests done with the SndCwnd. We can see that when we take all of the data, downloads, and uploads together, we have contradicting results again. This time, the KS and MWU tests contradict each other in their p-values. Upon partitioning the data into upload and download, we see again that all of the p-values of 0.0 are for the upload data, while the download data has, throughout all tests, a p-value of 1.0.

While RTT and SndCwnd are fields that change quite often, the rwndlimited data is, for most of the tests, just an array of 0s. As two arrays that are both just 0s are entirely identical, most of the data should be identical. This means that the rejection rates should be significantly lower. This should especially be true for the upload data. As the NDT servers are much more likely to have no rwndlimitation, more tests should be just 0s, so the upload tests should have lower rejection rates. This allows us to plausibly

check our data and test pipeline as high rejection rates or ones similar to the other tests would indicate high false positive rates in the testing pipeline and could invalidate the other findings with this testing pipeline. Luckily, we see the data we would expect to see with a working pipeline in table 4.6. We see that all of the p-values are 1.0. We see that all rejection rates are low compared to the ones we saw in the previous tables, and we saw what we expected for the upload tests: exceptionally low rejection rates.

Table 4.6: Rwndlimited results.

Type	**Test**	**Significance**	**Rejection Rate**	**p-value**
all	KS	0.05	0.63%	1.0
all	KS	0.1	1.11%	1.0
all	MWU	0.05	2.31%	1.0
all	MWU	0.1	3.86%	1.0
download	KS	0.05	1.06%	1.0
download	KS	0.1	1.87%	1.0
download	MWU	0.05	3.90%	1.0
download	MWU	0.1	6.51%	1.0
upload	KS	0.05	0.01%	1.0
upload	KS	0.1	0.01%	1.0
upload	MWU	0.05	0.01%	1.0
upload	MWU	0.1	0.01%	1.0

Last but not least, we have the applimited data. This data is not only often just an array of 0s, like rwndlimited data. It also contains a "1" if the connection is applimited at the time of the TCP info snapshot. This means it contains data that is just 0s and 1s and very often just 0s, which made it challenging to analyze like the other datasets with a KS or MWU test. Instead, it was tested using our own custom similarity metric, which was described earlier in 4.3.3. This means we need to use different significance thresholds, but the rest stays the same, so we can just interpret the rejection rates and p-values like before. As this data is often just an array of 0s like the rwndlimited, we would also expect it to be similar to it in regards to it having low rejection rates overall and, especially for the upload tests. We can see what was expected by looking at table 4.7. We see very low rejection rates overall and, especially for the upload tests, rejection rates under 1%.

Table 4.7: Applimited results.

Type	Test	Significance	Rejection Rate	p-value
all	custom	0.1	0.56%	1.0
all	custom	0.3	0.90%	1.0
all	custom	0.5	3.24%	1.0
download	custom	0.1	0.95%	1.0
download	custom	0.3	1.44%	1.0
download	custom	0.5	4.95%	1.0
upload	custom	0.1	0.00%	1.0
upload	custom	0.3	0.12%	1.0
upload	custom	0.5	0.74%	1.0

4.4 Finding Reasons for Rejection Rates

The last part was about whether sampled and unsampled data differ, and we derived that, especially for upload tests, they seemed to be different for RTT and SndCwnd. This part will now be about finding out why upload and download tests might differ and if the reasons why tests get rejected are the same for upload tests and download tests, just in different frequencies or if they actually have different reasons.

4.4.1 What Do We Test

As the reasons for the rejection rates could be various different problems that can occur with the data, knowing what to test for can be difficult. Instead of showing directly what tests we did and what the results are, we want to show some of the data that led us to what we tested and give an intuition for the problem at hand.

To find issues and patterns in the data, we filtered out the rejected upload tests and visualized them. As the TCP info snapshots do not have timestamps, we chose two different ways of visualizing. The first one is a naive way, where we take the position in the array for each value as x-value and spread them out equidistantly. The second one is a more sophisticated approach where we chose the acknowledged bytes of each snapshot as x-value.

We can see very clearly that the graphs are not very similar in the visualization with the relative array position as the x-axis in figure 4.3. They resemble each other in general shape and the magnitude of the values, but they are shifted. However, the

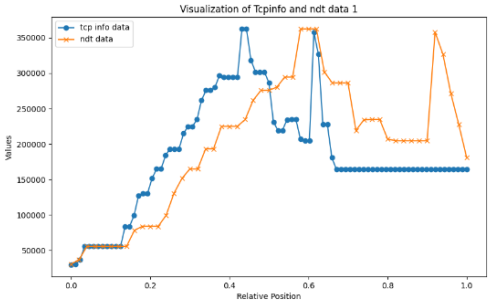


Figure 4.3: Visualization of `ndt7` and `tcpinfo` RTT data with relative array position for x-axis.

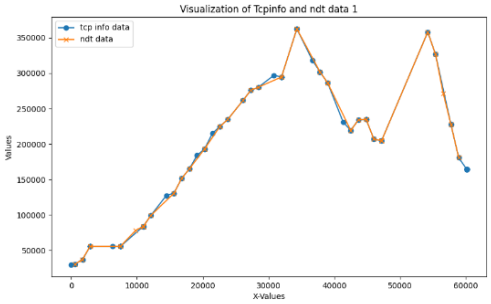


Figure 4.4: Visualization of `ndt7` and `tcpinfo` RTT data with Bytes_Acked for x-axis.

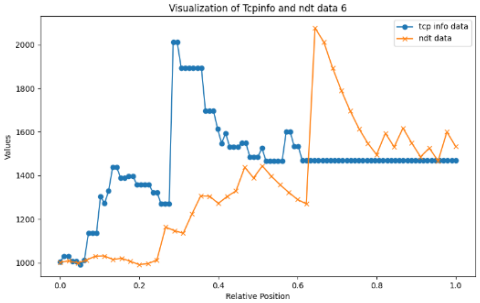


Figure 4.5: Visualization of `ndt7` and `tcpinfo` RTT data with relative array position for x-axis.

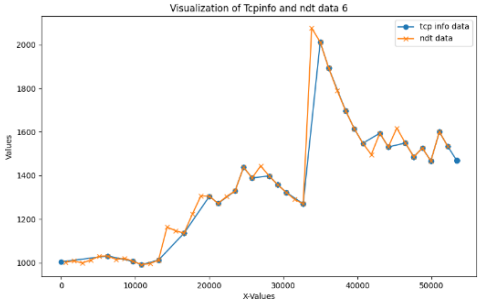


Figure 4.6: Visualization of `ndt7` and `tcpinfo` RTT data with Bytes_Acked for x-axis.

graphs align when using the Bytes_Acked for the x-axis in figure 4.4. The same can be seen for other speed tests in figure 4.5, 4.7, and 4.9 with their corresponding figures 4.6, 4.8, and 4.10 . We saw a recurring pattern in the data.

What comes to mind very quickly when looking at the figures is that the blue graph, the TCP info graph, seems to have a long tail at the right, where the values just seem to stay the same, and that vanishes when looking at the corresponding figure that uses the Bytes_Acked for the x-axis. In figure 4.9, this tail at the end consists of nearly the entire later half of the testing data. This later half getting eliminated when mapping it with Bytes_Acked means that these data points not only have the same RTT values but also the same Bytes_Acked values; these values are duplicates. This is what we decided to test; whether we can reliably remove the duplicates and get better test results for the data.

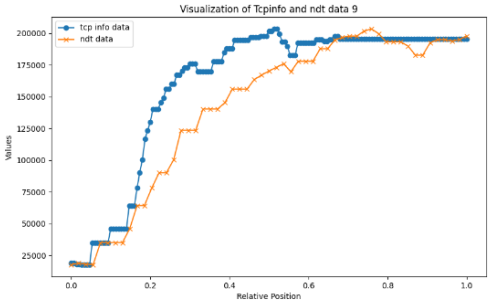


Figure 4.7: Visualization of `ndt7` and `tcpinfo` RTT data with relative array position for x-axis.

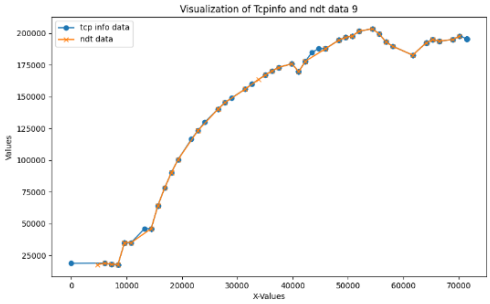


Figure 4.8: Visualization of `ndt7` and `tcpinfo` RTT data with Bytes_Acked for x-axis.

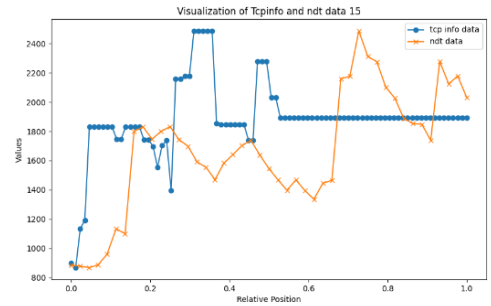


Figure 4.9: Visualization of `ndt7` and `tcpinfo` RTT data with relative array position for x-axis.

Figure 4.10: Visualization of `ndt7` and `tcpinfo` RTT data with Bytes_Acked for x-axis.

4.4.2 Designing a Test for Duplicates

In the last part, we looked at visualisations of the datasets that were rejected in our testing pipeline and formed the hypothesis that they were rejected because of duplicates in the data. Our null hypothesis for the upcoming tests is that we are given proper speed test data without duplicates. Our alternative hypothesis will be that we do have duplicates and that removing them will significantly reduce the rejection rate.

For this test, we copied the dataset and removed the duplicates in one copy and in the other not. Then, we will use the same KS and MWU testing pipeline as before to get rejection rates and p-values for both and compare them.

For this to work, we need to consider a few things. The first thing to consider are natural duplicates. Different TCP info fields are differently volatile. For some fields

like rwndlimited or SndCwnd, it is relatively normal to stay the same over at least a short period of time, while other fields like RTT are changing very frequently. For this experiment to work, we would need to use a volatile field like RTT to minimize the chance of natural duplicates. To further minimize the risk of removing natural duplicates, we join the RTT values first with the corresponding Bytes_Acked and SndCwnd values, remove only those rows that are the same in all values and then select just the RTTs out of them again to use for the pipeline.

When taking all the discussed considerations into account and running the test, we get the results we see in table 4.8. We can see that we were able to reduce the rejection rates for the KS test with 0.05 significance from 11.41% to 0.04%. This is a reduction of 99.5% of the rejections. The smallest gain we got was for the MWU test with 0.1 significance, but even here, we got from a 14.52% rejection rate to 0.40%, which is a reduction of 97.5% of the rejections. These results demonstrate quite clearly that removing the duplicates in the data significantly reduces the rejection rates. So much so that we can assume to have found the main reason for rejections among upload tests.

Table 4.8: Results for removing upload test duplicates.

Type	**duplicates**	**Test**	**Significance**	**Rejection Rate**	**p-value**
upload	duplicates	KS	0.05	11.41%	0.0
upload	duplicates	KS	0.1	14.55%	0.0
upload	duplicates	MWU	0.05	10.53%	0.0
upload	duplicates	MWU	0.1	14.52%	0.0
upload	no dupl	KS	0.05	0.04%	1.0
upload	no dupl	KS	0.1	0.09%	1.0
upload	no dupl	MWU	0.05	0.12%	1.0
upload	no dupl	MWU	0.1	0.40%	1.0

However, the data we have looked at is only about upload tests. For the download tests, we would need to look into table 4.9. There, we can see that the download tests barely had any benefits from the removal of duplicates. The download rejection rates were low to begin with, compared with the original upload rejection rates, but they seem quite high compared with the upload rejection rates after duplicate removal. Additionally, considering that the download rejection rates barely changed from the duplicate removal, there is likely another reason. Finding this reason is, therefore, what we will try in the next section.

Table 4.9: Results for removing download test duplicates.

Type	duplicates	Test	Significance	Rejection Rate	p-value
download	duplicates	KS	0.05	2.84%	1.0
download	duplicates	KS	0.1	5.97%	1.0
download	duplicates	MWU	0.05	3.41%	1.0
download	duplicates	MWU	0.1	7.12%	1.0
download	no dupl	KS	0.05	2.81%	1.0
download	no dupl	KS	0.1	5.88%	1.0
download	no dupl	MWU	0.05	3.33%	1.0
download	no dupl	MWU	0.1	7.05%	1.0

4.5 Finding Reasons for Download Rejection Rates

In the last section, we found out what caused the high rejection rates of the upload tests, duplicates, and that this is not the reason for most of the download tests' rejections. This still leaves the question: what caused the download test rejection rates?

We start again by looking at the visualizations of the data. Here, the most promising was looking at the data visualized with the Bytes_Acked for the x-axis. When filtering the data for download tests with rejections for the KS and MWU tests and visualizing, one trend showed up frequently. When looking at four of these visualizations in figure 4.11, 4.12, 4.13 and 4.14, we can see that the problem does not lay with inaccuracy or bad sampling, `ndt7` data seems to stop earlier than the `tcpinfo` data. In all four visualizations, both graphs seem to be more or less similar up to the point where one of them, the `ndt7` one, stops. We could see this phenomenon not only in these four visualizations but also in many others.

Having speed test data with different lengths in the database poses a big problem, as the length is an integral part of what the data can tell us, and different lengths could indicate other, more severe problems. Different lengths in the speed test data could indicate anything from problems with the sampling or the storage of data to problems with the processing of data at the M-Lab servers or even problems with the measurement processes themselves. Verifying the validity of the claim that the test data has different lengths is therefore very important and the topic of this section.

4.5.1 TCP Info Sampling Distance and Other Considerations

Before we start analyzing, an important question to consider is whether the tests are shorter because of the sampling distance of the raw TCP info snapshots. The last

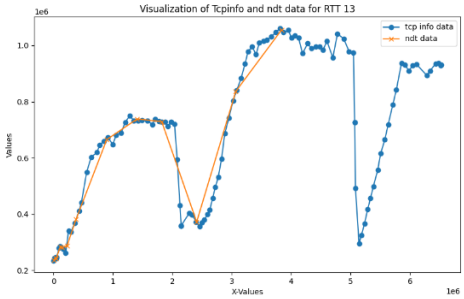


Figure 4.11: Visualization of NDT and TCP info RTT data for rejected downloads with Bytes_Acked for x-axis.

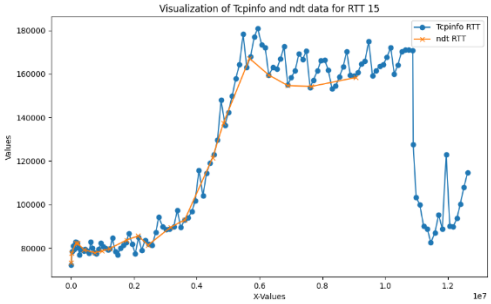


Figure 4.12: Visualization of NDT and TCP info RTT data for rejected downloads with Bytes_Acked for x-axis

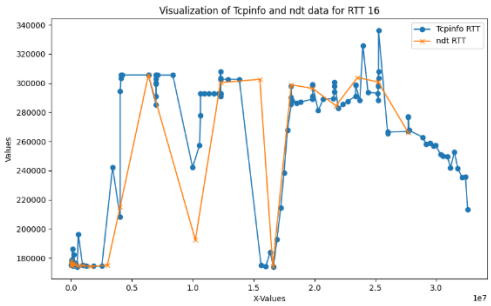


Figure 4.13: Visualization of NDT and TCP info RTT data for rejected downloads with Bytes_Acked for x-axis.

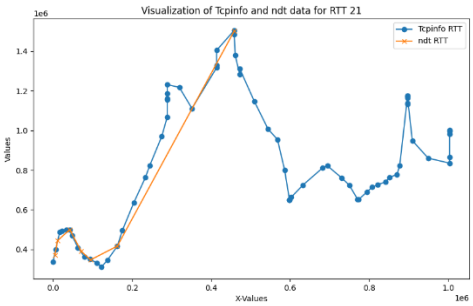


Figure 4.14: Visualization of NDT and TCP info RTT data for rejected downloads with Bytes_Acked for x-axis.

snapshot that is saved should be a maximum of 10 milliseconds old. Tests are five to ten seconds long, so even from a test that is just five seconds long, the sampling distance is at a maximum of 0.2%. Suppose we can lose at maximum 0.2% of the data and, on average, even less because of the sampling distance. In that case, we should be able to reliably use the snapshots for time measurement as they should represent the length of the speed test data quite well. Moreover, the sampling interval should not be the reason for this length discrepancy. Nonetheless, we want to take into consideration that things like the sampling distance, different sampling for `ndt7` and `tcpinfo` database views and other things could influence the length by a few percent so that we only count length differences that are above a certain five percent and test with various thresholds

to make sure we get reliable information out of the data.

4.5.2 Cutting the Lengths by Acknowledged Bytes

An efficient and fast way of testing whether tests have different lengths is to cut the longer ones and compare them again. If the rejections we saw happened because of the different lengths, we should see a decrease in the rejection rates. As a metric for the cutoff value, we chose the acknowledged bytes. We join the RTT data with it and cut off the `tcpinfo` RTT values corresponding to Bytes_Acked that are higher than the highest Bytes_Acked value from the `ndt7` views RTTs. Afterwards, we can test the data again using our KS and MWU testing pipeline and compare the results.

When looking at table 4.10, we can see the difference between the rejection rates of our normal test and our tests when we truncate the `tcpinfo` views data to the length of the `ndt7` views data. We can see that the truncation significantly affected the rejection rates, e.g. the KS test with 0.05 Significance went from 2.84% to only 1.55% rejection rate, which is only slightly more than half of the original value. This is not as impressive as the change we saw when we found the upload tests' problems with duplicates, but nonetheless something that supports the theory that the download tests have problems with test lengths. When looking at the other rows, we see that the rejection rates fell over all of the tests. This does not prove by itself that the download tests actually have a problem with different test lengths being saved, but it shows us interesting evidence in favour of that theory. Enough evidence for us to decide that this is something we should look deeper into.

Table 4.10: Results for removing download test duplicates.

Type	truncation	Test	Significance	Rejection Rate	p-value
download	normal	KS	0.05	2.84%	1.0
download	normal	KS	0.1	5.97%	1.0
download	normal	MWU	0.05	3.41%	1.0
download	normal	MWU	0.1	7.12%	1.0
download	truncated	KS	0.05	1.55%	1.0
download	truncated	KS	0.1	3.59%	1.0
download	truncated	MWU	0.05	1.84%	1.0
download	truncated	MWU	0.1	4.46%	1.0

Before we proceed, and at least as a double-check, we should also do this test with the upload test data. In table 4.11, we see that the rejection rates also significantly decrease

for the uploads. What seems confusing initially makes sense when looking at how this test truncates. Not always, but very frequently, the last few `tcpinfo` values are not in the `ndt7` data. This means that the more extended tests with duplicates at the end can remove their duplicates if they have a Bytes_Acked value higher than the maximum Bytes_Acked of the `ndt7` data. Moreover, we have already seen that removing these duplicates has a significant effect. As we remove duplicates here more as a side effect and only when they have a higher Bytes_Acked value than the maximum of the `ndt7` Bytes_Acked value we do not always remove them, which results in a lower reduction of rejection rates than if we actually tried removing duplicates like in table 4.8, but this again strengthens our findings from before.

Table 4.11: Results for removing upload test duplicates.

Type	truncation	Test	Significance	Rejection Rate	p-value
upload	normal	KS	0.05	11.41%	0.0
upload	normal	KS	0.1	14.55%	1.0
upload	normal	MWU	0.05	10.53%	0.0
upload	normal	MWU	0.1	14.52%	0.023213000713537 57
upload	truncated	KS	0.05	1.95%	1.0
upload	truncated	KS	0.1	2.80%	1.0
upload	truncated	MWU	0.05	2.79%	1.0
upload	truncated	MWU	0.1	4.79%	1.0

4.5.3 Pcap Based Test Design

To test the claim that the `ndt7` data stops earlier than it should, we are using a completely new test design. The most objective view of testing length are the raw packet captures that are saved in the pcap data in the GCS. We will compare the length of the `ndt7` data in the BigQuery database with the length of the packet capture and then decide based on the results. For that we are going to introduce new metrics, the ackratio and the timeratio. The ackratio is simply the quotient of the pcap maximum Bytes_Acked and the `ndt7` maximum Bytes_Acked. The timeratio similarly is the quotient of the pcap duration and the `ndt7` duration.

$$\text{ackratio: } \frac{\text{total pcap Bytes\_Acked}}{\text{total ndt Bytes\_Acked}} \quad \text{and} \quad \text{timeratio: } \frac{\text{pcap duration}}{\text{ndt duration}}$$

These ratios will give us a way of determining how much longer or shorter the pcap test data was compared to the ndt7 view's test data. E.g. an ackratio of 1.05 would mean that in the pcap 5% more bytes were acknowledged. A timeratio of 1.05 would mean that the pcap data is 5% longer regarding how much time the tests took. However, if we have a part with an ackratio of 1.05 and a part with a timeratio of 1.05, this does not mean that they are the exact same part of the data. It could be two disjunct parts of the dataset. For this, we created a flag called "both" for each test, indicating that a certain pcap is longer in both timeratio and ackratio. These metrics are all computed for each pcap and, in the end, evaluated for how many percent of the data have at least a certain ack- or timeratio.

4.5.4 Interpreting the Results of the New Test

When looking at table 4.12 we can see the ack-, time- and both percentages for four different thresholds: 1.05, 1.1, 1.2 and 1.5. The percentages in each column describe how many percent of the data have at least a ratio as high as the given threshold. The first thing that is directly visible is that the numbers are significantly higher than what would be expected if there was no problem with the testing length. When we look at the ack percentage for an ackratio of 1.05, we can see that more than 70% of the tests are afflicted. This means that we have more tests where the pcap is at least 5% longer than tests where this is not the case. We already assumed that we would need to consider things like the sampling distance and other problems, which is why we start with looking at ackratios above 1.05 and not 1.00. However, even if we consider our assumption too low and we should start looking at ackratios above 1.2, the numbers are still very high. An ack percentage of 33.54% means that one in every three tests has at least more than 20% discrepancy between pcap data and ndt7 data. The results are even more surprising when looking at ackratios of 1.5. 50% more bytes being acknowledged is high enough that we expected this number to be close to zero, even if our null hypothesis was wrong. Instead, we have 12.59% there, which means one in every eight tests is at least 50% longer.

Table 4.12: Percentage of download data above a given length threshold.

	1.05	**1.1**	**1.2**	**1.5**
ack percentage	70.61%	51.42%	33.54%	12.59%
time percentage	67.56%	41.26%	20.38%	4.90%
both percentage	59.18%	35.63%	17.07%	3.05%

It could be that the discrepancies between ndt7 data and pcap data could also be

much smaller in reality and that and the discrepancies in the values we see come from us measuring the time and length differently than M-Lab. This would explain some of the discrepancies if the e.g. M-Lab starts their time measurement after the Syn/Ack part of the connection and we start with the first packet that was exchanged, but that would not explain discrepancies in the range of 50% more or less in time or acknowledged bytes. We interpret this as strong evidence in favour of `ndt7` data and pcap data having different lengths. Whether this is the main reason behind the rejections of download test data or just part of the reason is a different question and one that is very difficult to answer. However, with the results at hand, we can say that it should at least be some sort of reason for it.

4.6 Conclusion for Data Sampling Analysis

In this first part of the thesis, we looked at data stored in the database of the M-Lab project, identified places where identical or sampled data was saved and compared sampled and unsampled data. We discovered that the data from `tcpinfo` and `ndt7` views are sampled so differently that a statistical test would frequently not recognize them as data from the same distribution. We found that the reason for that seems different for upload and download test data. The upload data seemed to struggle with duplicates in the data, while the download tests seemed to have problems with different test lengths in test duration as well as the number of acknowledged bytes. Overall, these observations mean that analyzing the M-Lab data and, specifically, the available transport layer data requires careful consideration of the data source. Based on our results, we recommend using the `tcpinfo` data and using a data processing step to remove duplicate `tcpinfo` entries.

5 M-Lab's Speed Test Methodology from a Transport Perspective

In the first part of the thesis, we looked at the data collected by the M-Lab in a non-semantic way. We wanted to know whether datasets were similar or whether we had duplicates or not, but we did not care about the information in the data itself. In the following we will look at precisely these information that are in the data. In this chapter, we will look at the connection behavior, the establishment and teardown of the connections and the way that the connections are limited.

5.1 Connection Establishment and Teardown

To get their Internet speed tested, clients connect via WebSockets to the Server. How exactly this connection happens to work in pactice and what problems we identify with that approach is what we want to have a look at, at the beginning of this chapter before we go to the connection limitations.

What strikes our attention is the status of the WebSockets and the connection after the test. The WebSockets remain after the test still in status "101 Switching Protocols", which indicates that the connection to the server was successful. However, after the test, this should change to "Closed", "Aborted", or something similar to indicate that the test is over. Upon inspecting the WebSocket messages, we were able to see that the last message contained a TCP info statement, which contains a "status" field. If the connection was closed, this field should be "0", but instead it was "1", indicating that the connection is still open. Lastly, upon looking into the packet captures, we can see that the connections are not getting closed properly but instead ended by packets with RST flags, which stands for "Reset" and is usually used to instantly end a TCP connection when a problem occurs. These RST flag packets occur not only at the end of the packet capture but also between the two tests when the download test ends and the upload test starts. This means that something often goes wrong at some point in the connection, like one of the parties not replying anymore after the test ended or that M-Lab is using them for purposes they should not, like for replacing regular connection endings. Either way, the connections leave the WebSockets in status 101 and the TCP info struct in status 1, indicating that the connection is not closed correctly.

5.2 Analysis of Receive Window Limitations

In the first part, the data part, we just looked at the rwndlimited values for comparing sampled and unsampled data. Now, we are actually going to use it to infer information about the limitations of the TCP connections. The data we get from the M-Lab project are all server-side. Being limited here would mean, that the server is not sending more data because of the client's communicated receive window. The first question is, how often are the connections limited by the receive window? This is relatively easy to test, as we just count how many tests have at least one entry in their rwndlimited array bigger than zero, as zero indicates no limitation and any other value indicates a limit. In table 5.1, we can see the results of this analysis.

Table 5.1: Percentages of Receive window limitation.

Type	**rwndlimited**
All	36.49 %
Download	63.76%
Upload	0.13 %

As we can see in the table, around 36% of the tests are receive window limited. However, this is only the case for the general view. When dividing it into download and upload tests, we can see that more than 63% of the download tests are rwndlimited while barely any upload tests are limited. The low values for the upload tests come from the fact, that the server is sending barely anything during the upload test. Reaching the receive window and being limited by it is therefore very unlikely, just as we see in the data. For the download tests, we see that a significant portion of them is rwndlimited. This means for the clients, that a significant portion of them is getting their download throughput not limited because of their internet connection, but because of the specific receive window limitation that their own device is posing on them. What we would still need to look at is the distribution of the rwndlimited. Until now, we have just taken the connections where at least one snapshot indicates rwndlimitation. But a connection with just one snapshot out of one hundred snapshots being limited is less harmful than a connection that is limited through the entire connection time.

What we see in table 5.2 is a collection of data about the percentage of how much of a connection was rwndlimited. For this, we counted how many snapshots were rwndlimited and divided it by the total number of snapshots a connection has. For the upload data, nearly all values are 0.0, as we only have 0.13 % in general, but we see that the maximum is 100%, which indicates that there are some connections that are entirely

type	dataset	min	max	lower quartile	median	upper quartile
rwndlimited	upload	0.0 %	100 %	0.0 %	0.0 %	0.0%
rwndlimited	download	0.0 %	100 %	0.0 %	86.67 %	100%

Table 5.2: Boxplot statistics for rwndlimited values.

rwndlimited. Nonetheless, for the upload tests, this might be a bad way to show the data. This table is more suited for the download data. We see that the minimum is 0.0% and the maximum is 100%, indicating that the rwndlimitation can range from no limitations throughout the entire connection to 100% of the entire connection. The lower quantile is 0.0, and the upper quantile is 100%, this means that the lower 25% are entirely without limitation, and the upper 25% are completely limited. The middle 50% are between 0.0 and 100%, with the median being 86.67%. We also plotted the data as CDFs, showing us how many percent of the data is smaller than a given value. We can see both plots in figure 5.1 and 5.2. On the x-axis, we can see the values we want to track, so how many percent of the connection is affected by rwndlimitations. On the y-axis, we can see how many percent of the speed test we had had the value on the x-axis or a smaller value.

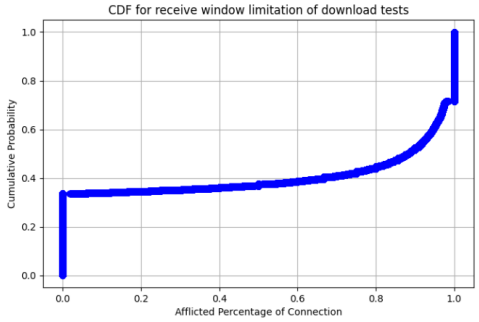


Figure 5.1: Receive window distribution CDF downlaod

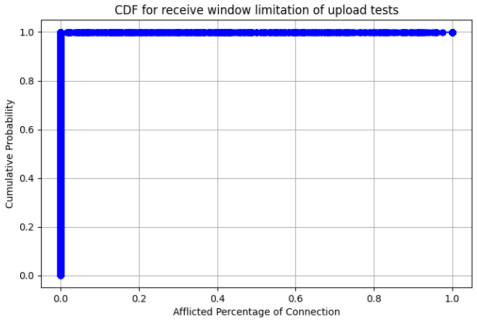


Figure 5.2: Receive window distribution CDF upload

Both CDF graphs show us the distribution of the data. For the upload data, we see in figure 5.2 what we already saw at the beginning; most of the connections are not affected by receive window limitations. In the graph for the download distribution, in figure 5.1, we see on the left that around 35% of the connections are not affected by receive window limitations (in this graph, it is visible as an affection of 0% at the x-axis). This fits with our result from above that around 64% of the connections have at least some rwndlimitation. We also see that around 30% of the connections are affected to 100%, they are rwndlimited through the entire connection time. In the middle between

them, we see a smooth curve with all values represented; this can tell us that we do not have any rapid spikes or something similar that would indicate any problems. This is also the general conclusion we get out of this section, which is that the rwndlimitations are only really present in the download tests but do not pose any problems for the testing methodology there.

5.3 Analysis of Application Limitations

The applimited data is similar to the rwndlimited data but different in meaning. A connection being applimited means that the sender could send more with the available bandwidth but does not get more data to send from the application. So while the rwndlimited data gives us a way to see when the server limits the connection as the receiver, the applimited data gives us a way to see when the server limits the connection as the sender. Similar to the previous rwndlimited section, we will first look at how often application limitation occurs in our data, so how many connections have at least one TCP info snapshot indicating application limitation. The results are visible in table 5.3.

Table 5.3: Percentages of application limitation.

Type	**applimited**
All	65.49%
Download	39.59%
Upload	99.97%

We can see in the table that around 65% of the tests are afflicted by application limitations. What is very clearly visible is the high percentage of upload tests with applimitations. This should not be a big problem, as the data comes from server-side TCP info snapshots, so the application limitation is based on the fact that the server was limited. As the client should mostly send and the server should wait and acknowledge, it makes sense that in the upload test, the server's application limits its sending rate. The download tests application limitation, however, is concerning. 40% of the download test connections being applimited is very bad, considering that the server should be the one sending, so we should look into it further. For that, we should look at how much of the connection is usually affected by the limitations. Connections that are only slightly limited at the very beginning might not be as much of a problem as connections being limited 100% of the connection time.

Figure 5.3 and 5.4 show two CDFs that visualize the distribution of applimitation

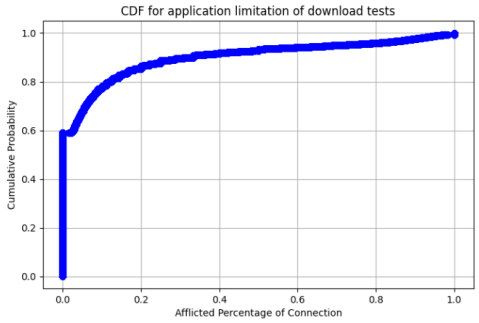


Figure 5.3: Application limitation distribution CDF downlaod

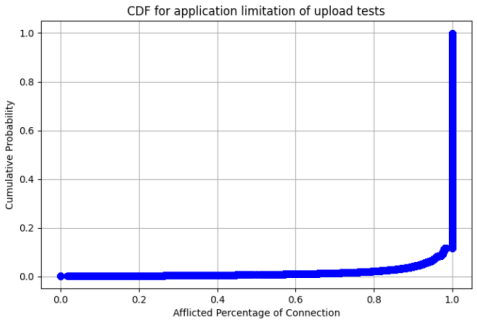


Figure 5.4: Application limitation distribution CDF upload

affection percentages. For the upload tests, we can see that most of the connections are affected nearly the entire connection time, which makes sense for the upload tests. For the download test, we can see that around 60% are not afflicted with application limitation (x-axis value = 0), which corresponds to the values we had above in the table that around 40% of the connections have any application limitation. Unlike the table before, here we can see how they are distributed. We can see that we do not have many tests that are affected through 100% of the entire connection time. Around 90% of all of the tests we looked at had less than 30% of the connection afflicted. As server-side application limitation in the download, test could potentially skew the test results of the speed test, and most of the speed tests have no or just very small portions of the connection affected by application limitation, we would expect this to be some sort of unexpected or unwanted behaviour. However, finding reasons for this and quantifying the impact on the connection would need to be done in future work.

5.4 Connection Behavior Conclusion

In this chapter, we looked at the data from the M-Lab project to gain information about the connection behaviour. We were able to see that the client opens two web sockets, one for the download and one for the upload, but realized that the connections are often not correctly ended and just end with rst flags. We also explored the rwndlimitation and applimitation of the connections and realized that the rwndlimitation poses no problem, while the applimitation might be a problem for the download test. However, exact quantifications of the impact would need to be done in future work.

6 Inference on Congestion Control Algorithms

The congestion control (CC) of TCP connections is a crucial aspect of modern Internet communications. As we developed several fundamentally different CCAs over the years, identifying the used CCA from a given connection became increasingly important. In this part of the thesis, we want to use the data we collected and analyzed in the first part to look at different aspects of congestion control and whether we can either identify the used CCA or at least see hints of it based on the data we have.

6.1 Server and Client Side

Our data is server-side collected data from the M-Lab speed tests. In these speed tests, we have two sides communicating in with each other, the client and the server. Therefore, we have two potentially different CCAs, the server's and the client's. Because the server mainly sends during the download tests and the client mainly during the upload tests, we can see the server-side CCA mainly during the download and the client-side CCA mainly during the upload tests. However, there is more we know about the server-side.

Unlike the clients, which could all potentially be different from each other, all M-Lab NDT servers are programmed to be the same for all connections. This includes the CCA. In the case of the most modern version, NDT7, the used CCA for the servers is BBR [22]. This tells us all we need to know about the server's CCA. If we want some more or more reliable information, we can see in the BigQuery database for every connection the TCP info structs as well as the BBR-info struct, which tells us not only that BBR was used (because the BBR-info struct should be empty if BBR was not used) but also how the CCA behaved. One could say that for the servers, we have all the data we could want.

For the client-side, it looks different. Identifying the client-side CCA is significantly more complex than the server-side one. Unlike the servers, the clients are all different machines with different operating systems and specifications and, therefore, potentially different default CCAs. This makes it more difficult, but it is not the only problem.

Even though we have CC information in the BigQuery database for all tests, we can use nothing from that because all the data we measure is for the server-side CCA. We can measure SndCwnd and can tell when packet loss occurred and have all sorts of data we would like to use to identify the CCA, but we have all of this data only for the server, not for the client. Without data, we have nothing to analyze, which makes it inherently nearly impossible to identify the CCA. Trying to identify the possibility and testing out various ways for this to still work is, however, precisely what we will try in the following sections. Instead of focusing on specific identification of the CCA, we will focus on the general question of whether it is possible to identify the CCA from the data we have. For this, we try out some approximations and more unorthodox ways in the identification and focus mainly on the identifying differences between Cubic and BBR because we expect those two to be the main CCAs used.

6.2 Looking at Throughputs

As we have a limited amount of information about the client-side for CCA inference, we need to try using whatever we have. One thing about the client we do have is the data it sends out because this data will reach the server and will, therefore, be in the server-side measurements we have. More than what the client sends to the server, how, when, and how much the client sends to the server will help us. By computing the throughput, we should be able to see these things. We will use a simple approach: The CCAs are very well differentiable via how they behave upon packet loss. Loss-based CCAs will strictly reduce the SndCwnd while BBR will not. However, when the SndCwnd, so the amount the sender can send without knowingly risking more congestion, rapidly decreases, the amount of actually sent bytes should typically also rapidly decrease. We can neither know about packet loss nor the SndCwnd behaviour, but these decreases in the amount that is being sent can be seen in the throughput of loss-based congestion controlled connections, while connections with BBR should not show this tendency.

We know that the server always uses BBR. So, we could look at how the throughput behaves when the server sends, which is the case in the download tests. The throughput of one such test is in figure 6.1. On the other side, we conducted our own NDT speed test and used Cubic on our local machine; the throughput is visible in figure 6.2. We can see that the throughput of the BBR using server rises initially and stays relatively constant. Our Cubic using client instead has a fundamentally different graph. It is less steady, it changes a lot and rapidly decreases at times. We can safely say that it is significantly more volatile in general.

The general idea is now, if we look at other throughput graphs of clients, and we see

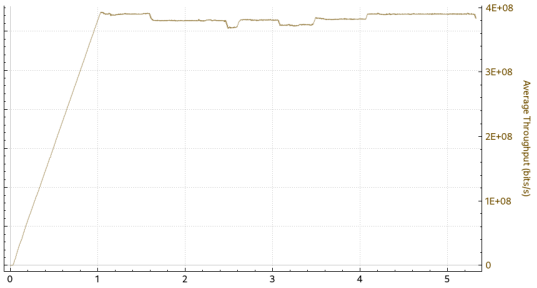


Figure 6.1: Throughput of BBR using server.

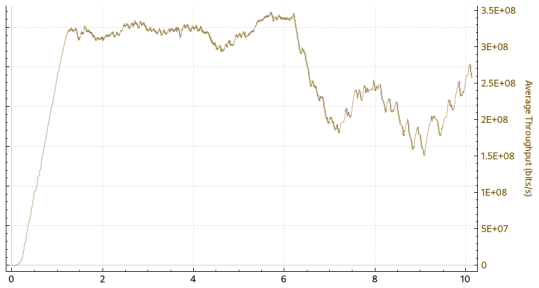


Figure 6.2: Throughput of Cubic using Client.

behaviour like in figure 6.1, then we can expect the client to have used BBR, and if we see behaviour like in figure 6.2 we can expect the client to have used Cubic.

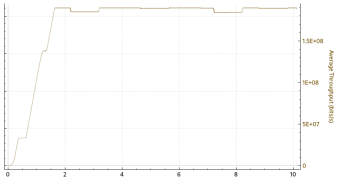


Figure 6.3: Throughput of upload test 1.

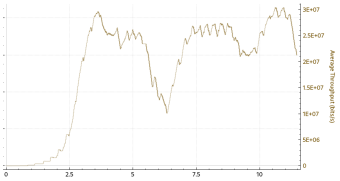


Figure 6.4: Throughput of upload test 2.

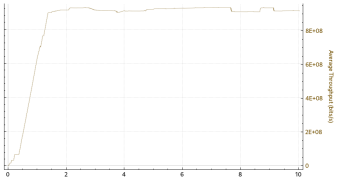


Figure 6.5: Throughput of upload test 3.

In the figures 6.3, 6.4 and 6.5, we can see three throughput graphs that are from upload tests. Even though they are all slightly different, we can see that two of them, 6.3 and 6.5, are similar to each other and are similar to the way a BBR throughput would look. The other one, figure 6.4, is different from the other two and resembles the way that our test upload with Cubic looked. We would interpret this as having found two upload tests that used BBR and one that used Cubic. Moreover, even though very manual and without automatization, we are able to infer information about the CC of upload tests.

We can say that we found good examples that show that our hypothesis, that we can identify the CCA based on the throughput graphs, seems to be true. However, it is also vital to search for counterexamples. One such example is visible in figure 6.6. On the one hand, we can see very rapid decreases in the throughput, something we would relate to a loss-based CCA. On the other hand, after these decreases, we see relatively stable throughputs, and we do not see any cubic function like behaviour in the throughput, which we should see. In general, we were unsure how to interpret this throughput graph, and we would have identified it as Cubic, even though this is

actually a throughput of a BBR using client.

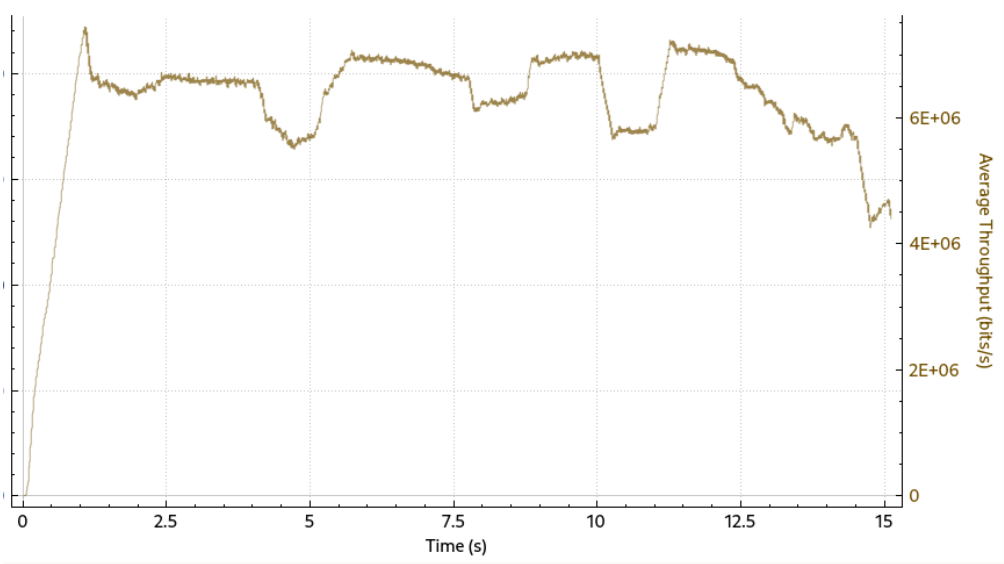


Figure 6.6: Counterexample for throughput based inference on CCAs.

This example shows us how flawed our approach can be. Not only did we find an example where we would have guessed wrong, but also one where it is generally challenging to decide. One reason this could be the case is because many other factors like applimitation, rwndlimitation, full buffers and many other things can also influence the throughput. If we honestly want to use the throughputs, we would need a more complex decision structure to decide based on the throughputs. This could be quite difficult for humans to accomplish, especially considering how many throughputs one must analyze to find such a decision structure. One could train a neural network, feed it with a dataset of these throughput graphs, and let it solve this as a classification problem, but this would be beyond the scope for this thesis. Therefore, we would conclude that inference on the clients used CCA by identifying rapid decreases and other factors in the throughput is not accurate enough to be considered when using humans for the classification but could be possible with more potent classification tools like a well-trained neural network.

6.3 RTT Smoothness

RTTs are a very important metric in transport layer communication that both sides of the communication can compute. When comparing congestion control, RTTs can be a

way to tell them apart. BBR attempts to avoid buffer bloat and should have relatively consistent latencies, so one hint for BBR might be the RTT smoothness. Loss-based CCAs do not care about the RTTs and often even artificially increase them by filling the buffers. This, together with rapid changes in sending rate when packet loss occurs, results in rapid changes in RTTs. So, if we have BBR with relatively smooth RTT distributions and Cubic with very fluctuating RTT distributions, we could use this to infer information about the CCA. If we could quantify the "smoothness" of the RTT distribution, measure it, and find a threshold, we could make decisions based on this.

6.3.1 Finding Good Metrics for Smoothness

Measuring the "smoothness" of a distribution can be very difficult, but in mathematics, the concept of "variance" comes close to it. The variance is the average squared distance of mean [19]. This means that for each speed test, we will look at how far, on average, the data points are away from the mean of the data points; smooth distributions will have very low distances between the mean and the individual data points, while very volatile distributions will have very high distances, so a high variance.

The variance is an absolute measure. It measures the squared distance to the mean, which will be higher for connections with high RTTs and lower for connections with low RTTs. This might, in the best case, weaken our expressiveness of the test and, in the worst case, introduce fatal mistakes. Therefore, the variance alone might not be enough, and it might be that we want to enhance this test by introducing another measurement with which we can mathematically express the "smoothness" of the distribution. We want a measurement that helps us quantify relative changes, and this measurement will be "volatility". In plain words, one can say that volatility is the measurement of change and of how much things change. So, low volatility will mean that we have more smooth graphs, while high volatility values will mean that we have a lot of rapid changes in the data. Mathematically, there is no one thing called volatility, as there are several different versions. We use the so-called "mean percent change volatility", where we use the percentage change from one RTT to the next as the metric based on which we compute our decision-making metrics. This also means that unlike variance, which is an absolute value, our volatility will be a relative value, making it more comparable between speedtests. Whether the volatility will perform better than the variance is difficult to predict and, therefore, part of this section's tests.

Before we dive deeper into the actual tests we will perform, we want to take some time to discuss the specific type of volatility we use, why we use it and how exactly we compute it. volatility is usually based on the standard deviation of the data. As the standard deviation is simply the square root of the variance, using the standard

deviation of the data would not result in us getting an independent metric from the variance of the data, which we want to use already. It would also result in us having an absolute metric again, which we already have with the variance. So instead, we are taking an approach often taken in these scenarios, which is using the percent change of the data and computing the standard deviation for the percent changes instead to get our volatility. Computing the standard deviation of the percent changes would result in us having a single number for the smoothness of our graph. However, it could result in some information loss if there are local spikes or general high Volatiliy in some parts of the graph and not others. To give more weight to these local spikes, we compute the standard deviation of the percent changes over sliding windows. With this, we would again have several standard deviation values, which we want to reduce to one value. In these cases, the mean or median are often used. The median is a robust measurement that does not change with very high or very low values and is therefore robust against outliers, while the mean is not. As we want to have a sensitive measurement of how high the specific parts of the graph are and specifically want to detect outliers and high spikes, we would use the not-robust and very sensitive mean. So, computing the mean of the sliding window averages over the percent changes should give us one volatility number, which should be a relative measurement sensitive to spikes in the data and the graphs. Approaches similar to this can be seen in other study areas as well. In medicine, they use the Average Annual Percent Change (AAPC) where they use annual data and compute averages over the percent changes. In finance, we see approaches like this in volatility analysis of stocks in computing the relative volatility [39, 51].

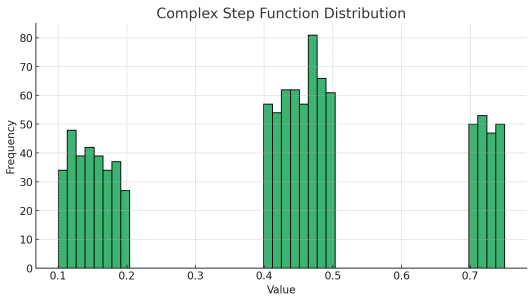


Figure 6.7: Example step function distribution with low variance.

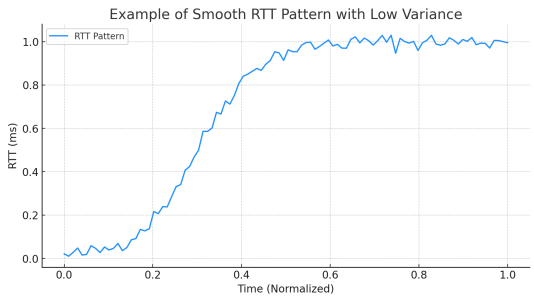


Figure 6.8: Example RTT with low variance.

There are some last considerations before we can use the variance and volatility. Smooth distributions will have a low variance and volatility. However, that does not mean that distributions with low variance and volatility are necessarily smooth. There are other distributions that also have low variance and would still not be classified as smooth by a human. One example of this could be a step function distribution like in

figure 6.7. It is a distribution where most of the values lay on a few different, discrete values, which can lead to low variance even though the distribution is not smooth. Normally, we would need to consider this when analyzing data; however, because we know that we are analyzing only RTT data, we should not have a problem. RTT data distributions usually have a profound form depending on its underlying CCA. One example of this can be seen in figure 6.8. RTT data will not produce distribution types like complex step function distributions, which is why we will assume that for tests, low variance and volatility in our data will be related to smooth distribution functions. The second consideration we have to make is noise. RTTs are usually very noisy as everything from network path changes to unrelated traffic of other network participants filling the buffer or even just processing delays of the receiver can influence it. Concerning the noise, we probably have to live with it. We originally thought smoothing the data could reduce noise and give better results, but this turned out to be wrong.

6.3.2 Conducting a Test for RTT Variance and Volatility

The goal for now is to assess how possible and feasible it is to get information about the CCA from the smoothness of the RTTs. We know that all of the servers use BBR, and we know that this is not the case for the clients. We expect most clients to use Cubic, and only a small part of them use BBR. We also know that the server is the one who sends the data in the download tests, and the client is the one sending in the upload tests, so comparing several thousand upload tests and download tests with each other, we should see differences in the RTT tendencies which will enable us to see whether inferring information about the CCA from RTTs is possible. If our theory is correct, and we can infer CC information from the RTTs variance and volatility because BBR has lower variance and volatility, then we should see that the download tests are, on average, lower in volatility and variance and that the upload test is bimodal, as the variance and volatility values of the cubic using clients should be significantly higher and stick somewhere together while the variance and volatility of the BBR using clients should be significantly lower and stick together, so we should have a bimodal distribution.

The processing pipeline we will use for this is very simple. We use the pcap files from the GCS as the basis for this test. We already preprocessed them into easily readable parquet files in the preprocessing pipeline, where we extract all of the RTT values from the pcaps. So, in general, we have a collection of several thousand parquet files, each containing, among other things, a list of RTTs. We just need to compute the variance and volatility for each of these lists and save them together with the information,

whether it was an upload or download test.

Before we start analyzing the distributions and their modality, we will first have a broader look at the distributions to decide what to use in the analysis. In figure 6.9, we can see the boxplots for the variance distributions. We see that whether the test was an upload or download test makes a difference. Unlike expected, the download tests have significantly higher variance than the upload tests. To put this into perspective, we need to do the same test with the volatilities.

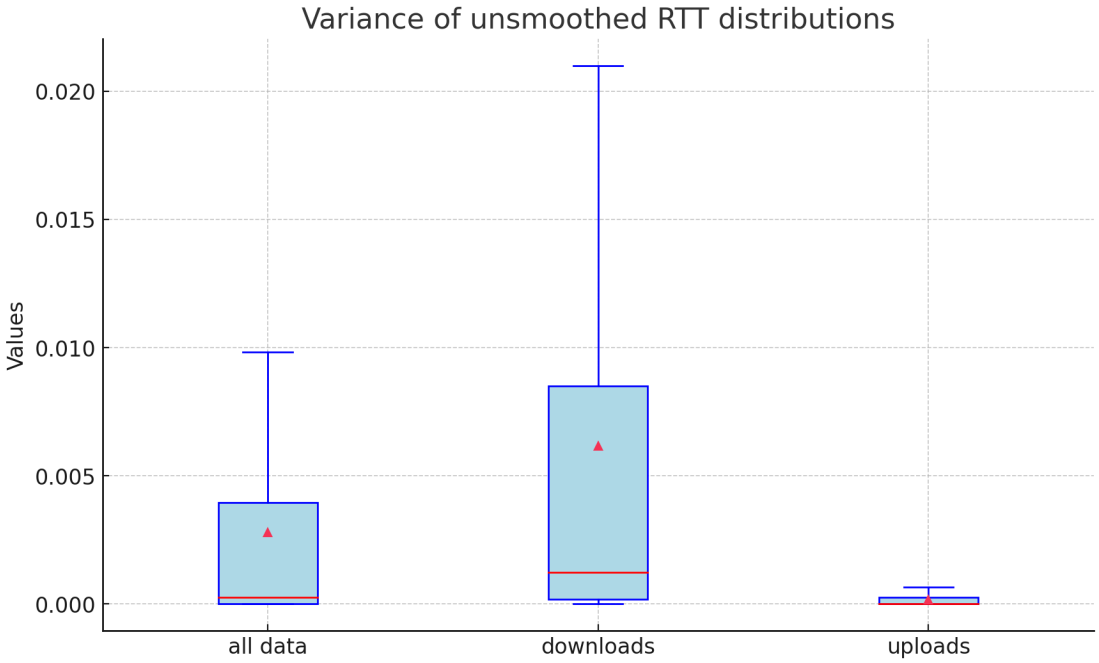


Figure 6.9: Variance distributions of RTTs.

Looking at figure 6.10, we see the volatility distributions for the RTT data. Again, we see a significant difference in the ratio between upload and download tests. Just as we expected at the beginning, we see that the uploads have a significantly higher volatility.

Even though what we see with the volatility graphs is entirely within the expected behaviour, it is nonetheless completely different from what we see in the variance graphs. The results seem to contradict itself. In data analysis, this is often a sign that we have bad data, mistakes in the testing, or both. When comparing the volatility and variance data of RTT between upload and download tests, we made the mistake of not testing for the differences of RTT data between upload and download data beforehand. Upon doing that, we realized that there is a big problem with how the upload test data is computed. We extract the RTTs from the packet captures via Wireshark and its

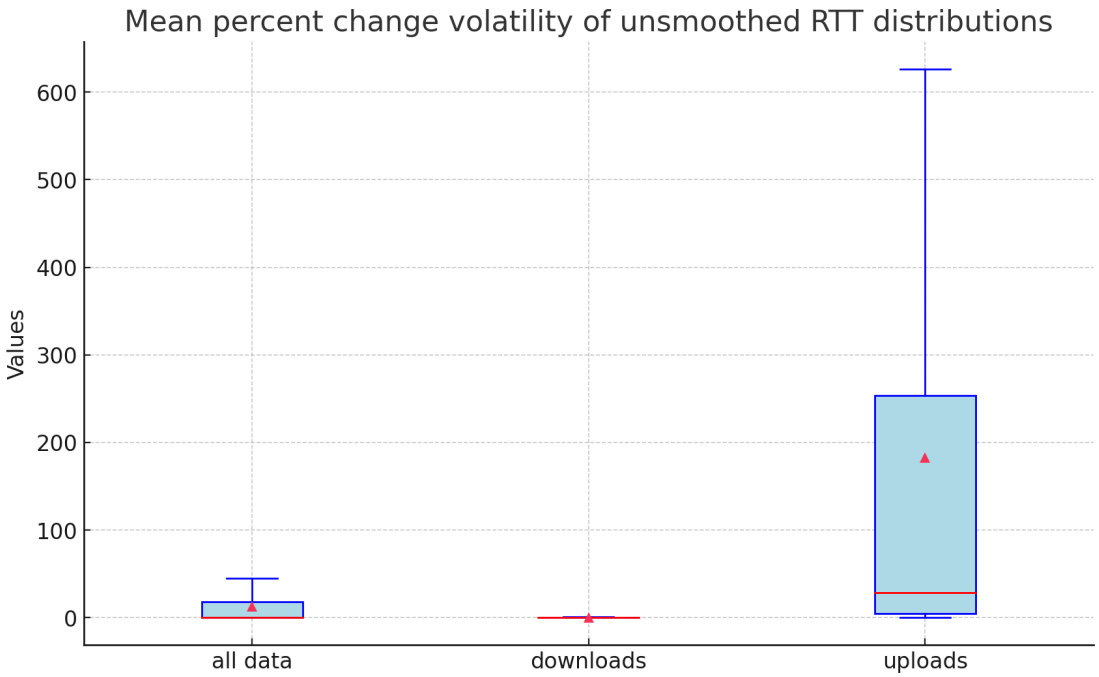


Figure 6.10: Volatility distributions of RTTs.

terminal version, tshark. When we do that with the upload test packet captures, we get significantly lower RTTs than with the download packet captures. This is likely because all packet captures are server-side, and estimating the RTT of client-sent packets from a server-side packet capture leaves us with problems. We conducted a small test to ensure that this was not due to Wireshark or the clients having lower RTTs than the servers. We did an M-Lab speed test with our own machine, captured a packet capture locally on the machine, and extracted the RTT data from it. We then got the server-side packet capture and extracted with the same method the RTT data from the server-side client RTT measurements.

In figure 6.11, we can see the difference between the local and remote capture. We can see that the RTTs we got from the server remotely measured are significantly different than the locally measured RTTs. While the locally measured RTTs are between 0.45 and 0.11, the remotely measured data are in the graphic at around 0.0 because the actual mean of the remotely measured data is 0.00001. So, the remotely measured RTT data has significantly smaller absolute values. Nonetheless, we should still see smoother distributions for BBR than for Cubic, so when comparing relative values with the volatility, we can see that the volatility is higher for download tests than for

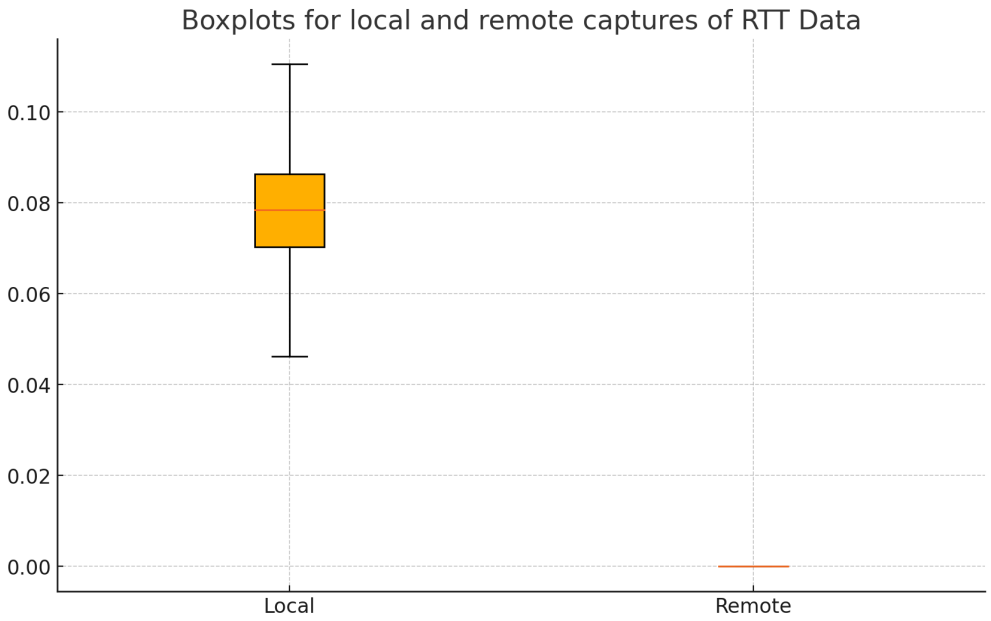


Figure 6.11: Difference between local and remote captured RTT data.

upload tests. This alone does not prove we can distinguish BBR and Cubic by the RTT smoothness, but it gives us good reason to believe that further testing can make sense.

6.3.3 Analysis with Bimodality Focus

Our goal for the entire congestion control chapter was to explore ways to infer information about the client-side congestion control algorithm. In this section, we try to find out whether analyzing the smoothness of RTTs is a good way to do so. For this, we already defined two metrics we can use, the variance and mean percent change volatility and found out that only the latter one makes sense to use with our data. In the first test, we found out that we see differences in the volatility between download tests (all of which use BBR) and upload tests (most of which do not use BBR), which suggests a correlation between the congestion control algorithm and RTT volatility. In this part, we will test further whether this correlation exists.

To do this, we will test for bimodality. The upload test data consists of BBR using clients and Cubic using clients. If BBR-using clients and Cubic-using clients have different volatility, we should see two groups of speed tests in the upload data, each with a different mean. We would see bimodality. To test this, we will visualize the data in a logarithmic histogram and assess the appearance of bimodality in the data.

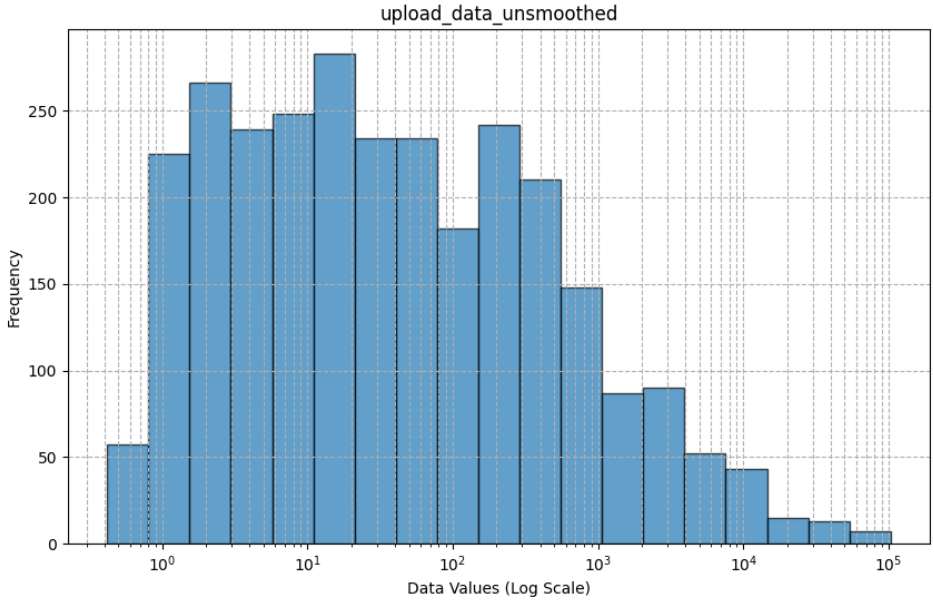


Figure 6.12: Volatility of upload speed test RTTs.

In figure 6.12, we can see a histogram for the volatility of upload tests. What we can see is an unimodal distribution with a large skew. The bimodality we expected to see because of the difference between BBR and Cubic volatility in RTTs is not visible, at least not from our data. We can see something more similar to what we expected if we take the download tests into account again. In 6.13, we see the download tests in blue and the upload tests in orange. We can see a clear distinction between the upload and the download tests. The download tests have a concentrated distribution with low standard deviation and skew. In contrast, the upload tests have a low concentration and high standard deviation and skew, especially considering that the scales that we are using are logarithmic. However, we do not see bimodality. We see nothing that could help us distinguish the congestion control algorithm, as mere shifting of the data or higher and lower skew could also result from the difference in measurements for download and upload tests or other external factors. So, we would need to conclude this section with a very clear conclusion: At least with our data, we were not able to see a reliable correlation between RTT volatility and congestion control algorithm in a way that would help us distinguish them.

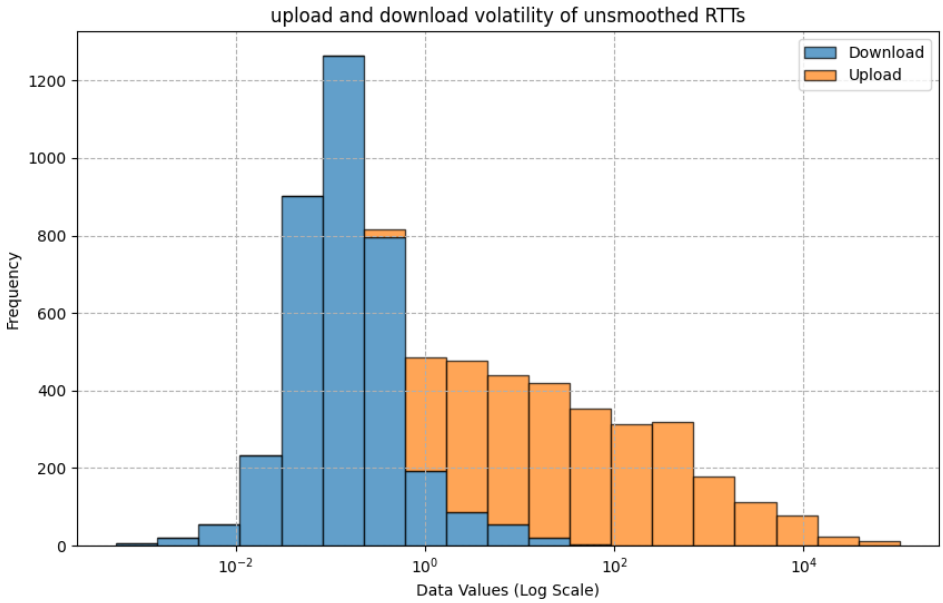


Figure 6.13: Volatility of upload and download RTTs stacked.

6.4 Approximating the Normal Way

Looking at throughput graphs and analyzing RTT smoothness are not conventional ways to infer information about a sender's CCA. It gave us some information but was insufficient for proper inference. Usually, in an ideal scenario, we would use packet loss and the SndCwnd for congestion control analysis. We would look at the times when we had packet loss or retransmissions and then determine whether the SndCwnd was reduced because of that or not. If yes, we would know that the sender is using a loss-based CCA like Cubic, and if no, we would know that the sender does not use a loss-based CCA, a CCA like BBR. The problem is that we do not have access to any of those. We do not know about retransmissions or packet loss as the packet captures are done solely from the server, so we do not know anything about the client that is not sent in the packets. Unfortunately, CC information is not sent in packets. We also do not know anything from the client about the TCP info information, as we only have server-side TCP info snapshots. Nevertheless, we may still find a workaround.

In the last two sections, we looked at things at hand, like throughputs and RTTs, even though they were not very descriptive. As that did not deliver sufficient results and

we cannot access the most descriptive metrics, we will try the opposite now. This time, we will look at metrics like packet loss and SndCwnd, which we know are incredibly meaningful and descriptive but which we do not have at hand. As we do not have the metrics, we would need to find ways to approximate, predict or estimate them and then use these estimates for analysis. This changes the problem at hand. This is no longer a task for inferring information about CC because these metrics are descriptive enough that if we had them, we could determine the CCA, but we do not have them. Instead, this task is to approximate the metrics as well as possible, and if we could approximate them well enough, we could use them for inferring CC information. In the following, we will look at "Packet Loss" and "SndCwnd", decompose how they work and then try finding ways to estimate, predict or approximate them. In the end, we will then evaluate how well this worked and whether or not this helps us infer information about the congestion control algorithm.

6.4.1 Finding Packet Loss

Generally, a sender does not know what happens on the line to the receiver with the packets, so the sender also does not know if and when packet loss occurs. There are three ways a sender can deduct that packet loss occurred: timeouts, duplicate acks and selective acknowledgments (SACK). If we could find a way to get these values, we could reconstruct whether packet loss occurred and approximate the actual packet loss.

Timeouts

The first thing we are going to have a look at are timeouts. Timeouts are very difficult to approximate, to find packet loss. The sender is not broadcasting or sending anything about this via the network, which makes it impossible to detect this as the receiver. Normally, we could use the retransmission timeout (RTO) to simulate how the timeout could have happened, but the receiver also does not know the RTO as the sender continuously and dynamically computes it. This may make it impossible or at least very difficult for the receiver to simulate the timeout retransmissions. While very difficult, conceptionally and theoretically, it could still be possible to approximate this by calculation and estimation.

When estimating packet loss via timeouts, the biggest problem is that we do not know after what time period the sender considers a packet as timed out. Nevertheless, this time, the RTO is computed, and the receiver can do the computations as well, at least approximately. We could compute them with Jacobson's algorithm, but we would need the RTTs, the smoothed RTTs, and the RTTs' variance. This will not be perfectly accurate as the sender can incorporate information about the connection that the receiver does

not have, for example after receiving explicit congestion notification (ECN)s. However, it should be a relatively accurate approximation. With that, we could try reconstructing timeouts. If we get three out-of-order packets, first packet 1, then packet 3, and then packet 2, then we know that packet 2 should have been sent between packet 1 and packet 3. We can look at the times they were received and compute the middle. We can then subtract half of the RTT from that value to estimate when the packet was sent. If this approximated sending time and the actual receiving time have a longer time difference than our estimated RTO, we could expect this packet to be a retransmit. However, this is full of consecutive estimations and approximations and is difficult to do in practice. We were not able to do this in practice here, but future work might be able to verify whether this can work in practice. Until then, we consider this approach for this thesis as not feasible.

Duplicate Acks

The second thing we will look at is the Fast Retransmit method based on Duplicate Acknowledgments. Instead of waiting for a timer to run out, which can potentially take quite some time, the sender will resend its packet if it receives three duplicate acknowledgements. These acknowledgements are sent via the network, meaning they will be in the pcap files, so we should be able to reconstruct this.

We constructed a test where we did a speed test with one of our own machines. On our machine, we can locally capture the data, and we should see relatively accurate information about the congestion control algorithm and the packet loss occurrence. We will then use the pcap captured by the remote server and compare the packet loss information we would get out of it with our locally sourced ones.

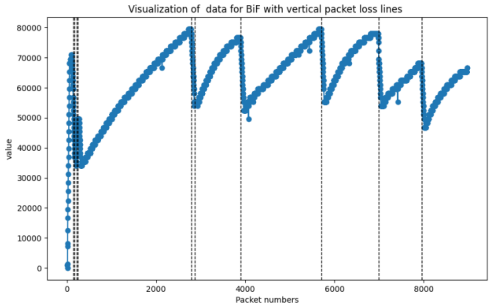


Figure 6.14: Locally sourced packet loss information.

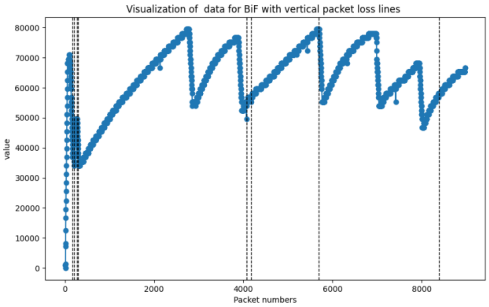


Figure 6.15: Remotely sourced packet loss information

In figure 6.14, we see the locally sourced BiF as a blue line. Whether or not remotely

sourced BiF are a good option for evaluating the congestion behaviour is something we will discuss in the next part, but the locally sourced BiF data will help us put some context to the packet loss data we collected. Whenever we saw three duplicate Acks in the packet capture, we marked this with a vertical black line in the graph. We can see that the BiF follow a shark tooth pattern, as it is common for Cubic, and that the packet loss we detect from our local packet capture aligns very well with the sharp drops in BiF we see. This, however, is not necessarily true for the remotely sourced packet loss information in figure 6.15. Again, we have the same BiF graph underlying to give the packet loss lines some context and make them more comparable to the local packet loss information. We can see that the information on remotely sourced packet loss does not match well. Some lines match, some are missing, and the last one is misaligned. We can see that the end of the slow start phase is well visible in both graphs, but other than that, we would conclude that trying to estimate the packet loss with remote packet capture is very unreliable. The method itself seems to work well, as we can see with the local data, but the remote data does not provide the accuracy that we would need.

SACKs

The last thing we might consider are SACKs. They are unique in the way that in the event of packet loss, they leave out the lost packets from the acknowledgements but still acknowledge the packets that came afterwards to prevent unnecessary retransmitting. However, the way they indicate the sender packet loss is generally the same as the fast retransmit method we just looked at, as they also use three duplicate acks for signalling packet loss. It is, therefore, very unlikely that we would get any additional information out of them other than what we got already from analyzing duplicate acks.

We can, therefore, conclude that estimating the occurrence of packet loss from a receiver's side is mainly possible by analyzing the duplicate acks in the packet captures. However, as this approximation is quite unreliable, it would need to be augmented with other data like BiF or SndCwnd to make it usable. If that is not possible, we do not have a reliable way to estimate the actual occurrence of packet loss of the sender from the receiver side.

6.4.2 Approximating SndCwnd

One of the most significant differences between loss-based and congestion-based CCAs is their behaviour concerning the SndCwnd. Therefore, it makes sense that if we could analyze the SndCwnd, we could gain valuable insights into how the connection behaved

and what CCA might have been used. As the SndCwnd is an internal state that is not being broadcasted or sent, we could only get information about the client-side SndCwnd with client-side data. However, all of our data is server-side and only entails information about the server's internal state. This leaves us with no other option than approximating the SndCwnd.

Fundamentally, the SndCwnd is the amount of data that can be sent without getting acknowledged. So, it is the maximal possible value that the difference between the last sent sequence number and the last acknowledged sequence number can have. Even though this maximum is not always fully reached as other factors like applimitation, rwndlimitation, or other things can hinder the sender from reaching the full SndCwnd, the sender tries typically to use its resources as efficiently as possible and tries to use as much of the SndCwnd as possible. This actual value of how many bytes are being sent but not acknowledged, so the actual difference between the last sent sequence number and the last acknowledged sequence number is called BiF. Moreover, if the sender already tries to get the BiF as close as possible to the SndCwnd, it would make sense to use it to approximate the SndCwnd.

Even if the Bytes in Flight do not approximate the SndCwnd well, it is not be a significant problem, what would be important is that it follows the same trends. If it has significant drops when the SndCwnd has significant drops and generally rises when the SndCwnd rises, we can use the BiF for analyzing CC behaviour.

Our goal here is not to infer the CCA of the client but to find out if it is possible and feasible. And for that, we want to estimate the client's SndCwnd. For the client, we only have the BiF we need for the estimation, but for the Server, we have the BiF for the estimation and the SndCwnd to see how accurate our estimation is. As we do not have data about the client's SndCwnd to check the accuracy of our analysis, we will instead test with the download tests and the servers BiF because we can check the accuracy of our estimates for it with the actual SndCwnd data. We will compare the BiF data we get from the pcap and the actual SndCwnd we get from the `ndt7` BigQuery data and look for similarities. If we can estimate the SndCwnd with the BiF well for the download tests and the server data, we should be able to use it to estimate the SndCwnd. The end of this section will discuss how we can conclude from this to the upload tests. For now, we start with the download test data analysis.

Two of these analyses we can see in figure 6.16 and figure 6.17. We can see in blue the BiF extracted from the pcap file and in orange the SndCwnd that we get from the `ndt7` data. As we know from previous parts, the `ndt7` data has the tendency to be shorter than the pcap, which is why we made sure to only look at tests where pcap and `ndt7` data are close in length. As the SndCwnd and the BiF are measured in different units of measurement, SndCwnd in maximum segment size (MSS) and BiF in Bytes, we multiplied the SndCwnd with the MSS to make sure everything is measured in Bytes

and is comparable.

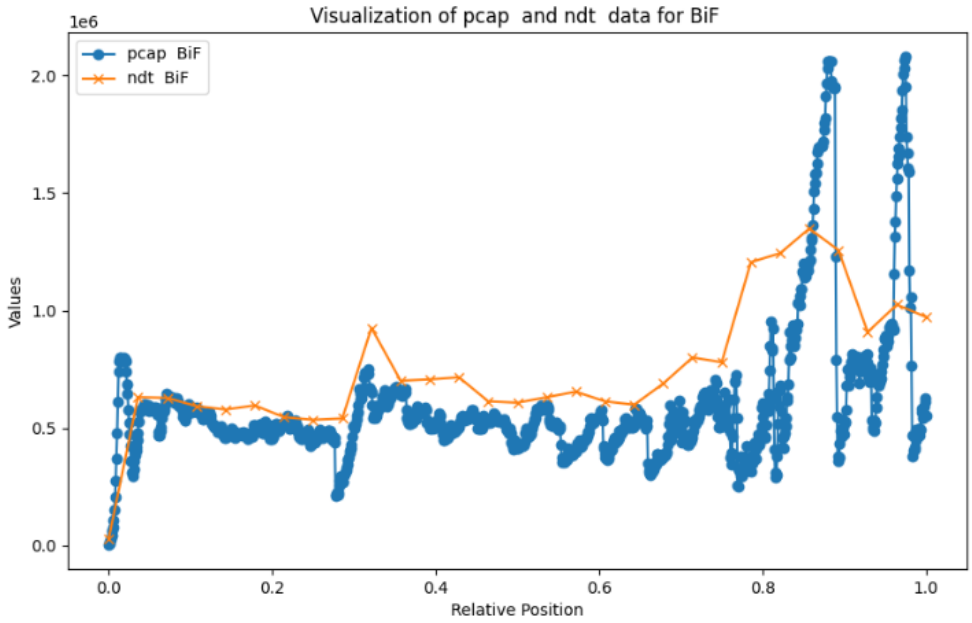


Figure 6.16: Comparision of BiF and SndCwnd for download test data.

Both figures show that the BiF and the SndCwnd have similar behaviour. In 6.16, they both follow a sideward trend and increase at the end; in figure 6.17, they both increase at the beginning significantly and follow a downward trend from there on. We can see in both figures that the graphs generally follow a similar pattern and that peaks and valleys occur together. We even see that especially in figure 6.16 between 0.5 and 0.8, the BiF show a shark tooth-like pattern with rapid decreases and increases that are not in the data for the SndCwnd as they have a width that is roughly the size of the sampling distance of the SndCwnd data. This can generally be a good sign as we see more or less the patterns we expected and even more detailed than what we can see in the SndCwnd Data. These graphs are only two examples. However, other examples paint a different picture.

Upon looking at 6.18, we see some problems. The graphs still follow the same trend; in this case, they both follow an upward trend, but the BiF graph is very volatile. It has very rapid de- and increases that would make it very difficult to use it for approximating the SndCwnd and for inferring CC information as we would be looking for dips that indicate decreases in the SndCwnd, but with such a volatile graph we can barely do that. This significantly decreases the accuracy and expressiveness of our analysis if the

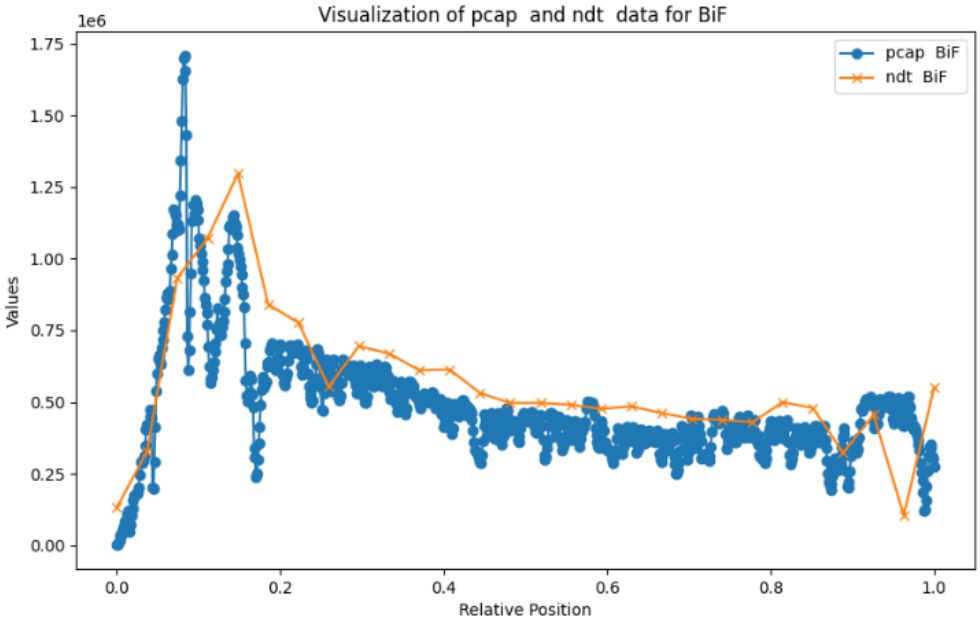


Figure 6.17: Comparision of BiF and SndCwnd for download test data.

data is not augmented by other data like the RTT data or information about packet loss and could potentially invalidate the ability to use this way of estimating the SndCwnd by the BiF at all.

Additionally, we have another major problem: the main assumption we used for this test. We wanted to infer information about client-side congestion control. For that, we tried to estimate the SndCwnd using the BiF. Because we have BiF and SndCwnd for the server but only BiF for the client, we tested the assumption on the server to get an easy-to-test testing pipeline and then tested with small sample size and local tests whether we saw with the clients the same results as with the servers. This main assumption that we should see the same results for client and server BiF does not hold upon testing. The reason for that lies in the way we get the BiF. The server captures the packets server-side, but the BiF are only accurate for the side that captures them. So we see accurate BiF values for the server-side, but the client-side BiF do not have a good quality. To verify this, we did an M-Lab speed test on our own machine and captured the packets locally. We compared the BiF we got from the local capture and from the remote, server-side capture in figure 6.19. We did the test with a client using Cubic, something we can clearly see in the locally sourced BiF values. They follow a very clear structure that is typical for Cubic. However, the BiF we extracted from the remotely

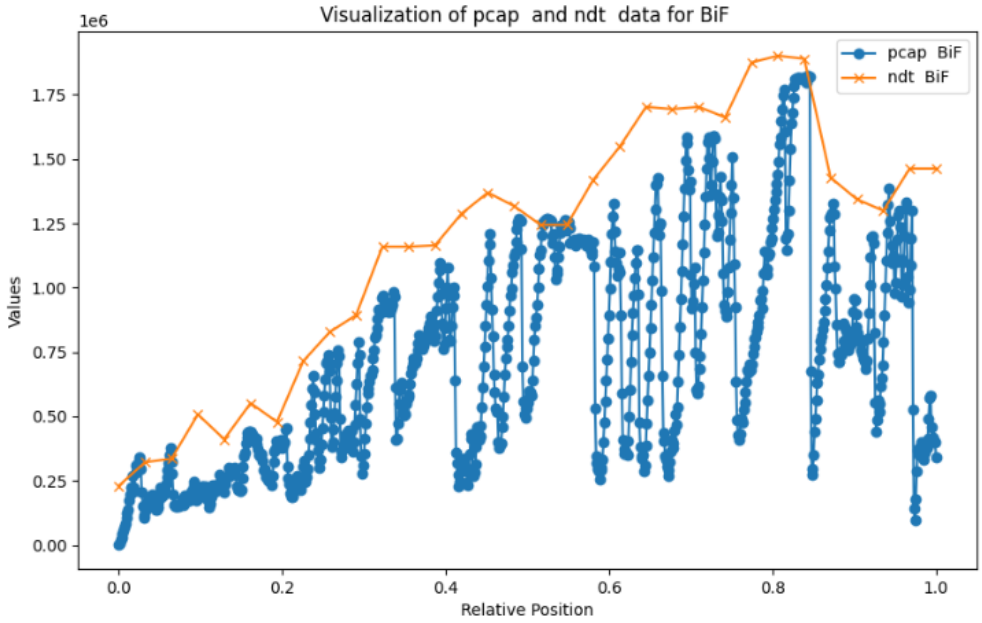


Figure 6.18: Comparision of BiF and SndCwnd for download test data.

done, server-side capture are very poor. They do not match either the local BiF or any way the BiF of a Cubic-using client should be. This makes it very clear that using the BiF from the server to estimate anything for the client is probably impossible.

In general, we would say that approximating the SndCwnd with the BiF is possible and logical under certain conditions. It might pose some problems with the accuracy of the approximation and might, therefore, not be suited for decision-making solely based on this. However, as long as we use only the server-side BiF or source the client information locally, we can use this method. Nevertheless, our goal was to estimate the client-side SndCwnd with the server-side captured BiF data, which did not work due to the lack of proper data quality. Estimating the SndCwnd with the BiF does not have high accuracy to begin with, even with good data, but with bad data, as we see it in the remotely captured BiF, it is impossible. This is not necessarily the fault of the M-Lab infrastructure but a general problem with how the BiF are calculated and that it is very difficult to calculate them from a remotely captured packet capture. This method could be usable if M-Lab would, at some point, change the logging of the upload tests and do it on the client instead, but as long as this does not happen, this method of estimating client-side SndCwnd will not work.

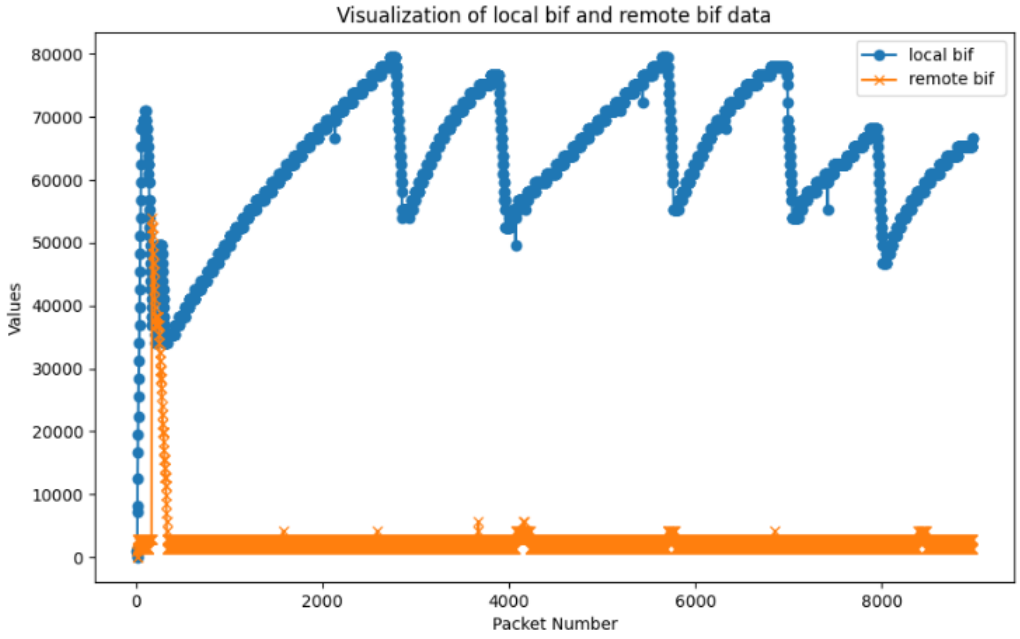


Figure 6.19: Comparision of BiF from local and remote capture.

6.4.3 Conclusion for Inferring Information about CC with Approximations

In conclusion, using approximations to infer congestion control information can be very challenging. In the last section, we discussed options for approximating the SndCwnd and estimating the occurrence of packet loss. We discussed and described different ways to do this for the packet loss and implemented one of them. We saw similarities between the packet loss we would estimate and the actual packet loss. The accuracy, however, was not high enough for proper and reliable identification. Estimating the SndCwnd did not give us the results we wanted at all. We saw that the BiF followed similar enough patterns that they could be used for estimation. However, the accuracy of this approximation might be too bad for it to be used as the sole indication of the behaviour of the SndCwnd. Upon looking at the specific data we got, the server-side captured client BiF, we saw that the data does not have the quality we need to make the approximation work. All in all, we can conclude that approximating congestion control information gave us some insights into how we can approximate packet loss and the SndCwnd from a packet capture, but it does not yet enable us to reliably identify congestion control algorithms from them.

7 Conclusion and Discussion

In this thesis, we explored crowd-sourced Internet speed test data with a focus on their transport layer characteristics. In the first half, we evaluated the quality of the data saving and sampling process before we explored the connection behaviour and different approaches for CC inference in the later half. In this part of the thesis, we want to evaluate our work and discuss our analysis's accuracy and relevance. After that, we will discuss the limitations we faced and future work that might be possible or necessary after this thesis.

The first research question was answered in chapter 4. We were able to identify that we have sampled data when it comes to the TCP info data, which can be found in the `tcpinfo` view as well as in the `ndt7` view of the data but differently sampled. We programmed scripts for fast and automatic data download and preprocessing and found out in statistical tests that the upload tests have problems with duplicates, while the download tests had rare cases where the tests were too short. We conducted additional tests to test the length of the speed tests based on the pcaps and were able to validate that a significant part of the data is in the `ndt7` data shorter than in the pcaps.

In the second part, we looked at the behaviour of connection. Here, we were able to identify how the connections were limited throughout the speed test. We found out that most of the rwnd- and applimitation were either very small and of no concern or relevant for the functionality of the test (e.g. server-side applimitation for upload test). What did concern us was the applimitation of the download tests, which was around 40% of the tests present and, therefore, significantly more common than expected.

Last but not least, we explored possibilities for inference on the client's CCA in chapter 7. We tried out three options: inference via manual throughput graph classification, inference through statistical RTT smoothness analysis and approximation of packet loss and SndCwnd for inference on the CCA. In general, neither of those three gave us the results we hoped. The inference through throughput graph analysis was not scalable but also not accurate enough. The RTT smoothness did not work as expected, as we could not see differences in the data. The approximations of the packet loss and SndCwnd also turned out to be not accurate enough for inferece based on them. In

general, the biggest problem with our results are their accuracy. It could potentially be improved in future work if the approximations were to be augmented with other data or approximations, but until then, we would say that all of our explored ways did not deliver accurate enough results to be considered a success.

In total, we can conclude that while looking at transport layer characteristics of this crowd-sourced data did not deliver us reliable information about the client's CCA, it did enable us to find information about the duplicates and other problems in the sampling process as well as information about the connection behaviour.

7.1 Limitations

In this section, we want to discuss the limitations of our analysis, the limitations we faced during our analysis, and how they affected the results and where bias and variance might have been introduced.

One of the most significant limitations of the analysis in this thesis is the limited timeframe of the data we used. We have a timing bias there as we only took data from 01 December 2023. The most significant limiting factor for a more comprehensive analysis is the computational power to process raw pcap data at scale. While analyzing the BigQuery data was not a big problem and was done in a very timely manner for each analysis, analyzing the raw pcap file data posed a bottleneck that we want to describe here. For the BigQuery data, we were able to use an entire day's worth of global data, download, process and analyze it within one day. One day of worldwide pcap data has a size of roughly three terabytes. Even with a Gigabit connection, the mere download of the data took nearly an entire day. This data must still be preprocessed into a well-readable format for later analysis. We brought the necessary time for the preprocessing of one pcap from roughly five to ten minutes down to 0.5 seconds through parallelization, efficient memory usage, good caching behaviour and a preloaded lookup table for sorting. Nevertheless, even with a processing time of 0.5 seconds, preprocessing the seven million pcaps of one day's worth of data takes 3.5 million seconds or roughly 40 days of processing time. With this, it became clear that using more data than that would be unreasonable. Even this data was too much. Instead, we focused on sampling data from all M-Lab locations within that day to make sure we didn't have a location bias and accepted the bias in time. A time bias can be challenging to live with in our analysis as no one knows what impact it has. It could be that 01 December 2023, the day we used for our data, was normal and representative, but it could also be that the Internet had some problems that day. We do not know that,

but using more data was also not an option, so until our analysis is redone with other data, our analysis will contain this timing bias.

Another limitation of the analysis is the use of many statistical tests. While they are generally a great tool, especially when considering the sheer amount of data we had, each statistical test has its strengths and weaknesses and is usually biased in some way or another. We made sure to use and consider usually one or more alternatives for each test, e.g. testing with the KS test and the MWU test instead of just using one. However, it might still be that by using these tests, we brought some bias into the analysis. This is especially true considering the size of our datasets. For some cases, the KS test switched from exact to asymptotic computation of the testing values. This alone is nothing bad and should still give us roughly the same results, but it slightly lowers the accuracy of our analysis and, therefore, might impose some bias. As this is based on the size of the dataset, to mitigate this, we would need to use completely different, stronger technologies like a well-trained neural network, which in turn would come again with its own limitations in understandability and reproducibility.

7.2 Future work

The M-Lab dataset is not only one of the most extensive Internet measurement datasets and is still constantly expanding but also freely available for the scientific community [27]. This means that future work in analyzing this dataset is not only necessary and vastly meaningful but also very valuable for broadening the knowledge about the structure of the Internet. Our analysis of the sampling methods used in this dataset revealed some rather unsightly problems like duplicates and different test lengths. As these can threaten all analyses based on this data, future work in this aspect, questioning our results and either confirming or disproving our results, is essential.

In this work, we showed that most of the applimitation and rwndlimitation that the client-server connection faces should not be a problem for the speed test. However, at least the download test's applimitation might be a problem. Whether this is the case or not and how significant this impact might be is also something where future work makes sense.

Another part where future work might be needed is the inference for the client's CCA. Inferring a senders CCA from receiver-side captured data is something we expected to be rather difficult but worth trying. As stated above, the approaches we tried could not infer the client's CCA. However, they pose high value for future work. Combining

approaches we experimented on with each other or with other approaches might lead us closer to proper inference on the client's CCA. However, these are things we want to leave for future work.

7.3 Reproducibility

We made sure that all of our results are reproducible. For this, we have prepared a git repository with all of the scripts and some testing data available at this link: https://gitlab.lrz.de/cm/2024-leonardo-kubilay-oezuluca-bachelor-thesis-material

The data used for this thesis is not included in the repository. However, the SQL queries and Python scripts with which we got the data are all included with information on how to use them. If not differently stated, the data used for this thesis is from 01 December 2023. The BigQuery data used for the analyses is the entire global `ndt7` and `tcpinfo` data from that day. The pcaps, however, were significantly more difficult to download, extract and process, so a sample was used. The sample has around 20000 pcaps and was sourced from all M-Lab locations to minimize location bias.

Reproducibility of Section 5.1 In section 5.1 we tested the behavior of the websockets in the browser when conducting the M-Lab speed test. As this did not involve any programming, there is nothing about it in the repository. We used a Firefox browser with version 130.0.1 (64-bit). To reproduce the results, go to the M-Lab website [26] or search via google "how fast is my Internet" and press F12. This will open the browser tools. Navigate to the "Network" tab and choose the "WS" view to see all of the Websockets that will get opened.

Reproducibility of Section 6.2 In section 6.2 we also didn't use any scripts. We analyzed Wireshark troughput graphs.To reproduce them, simply open in Wireshark a packet capture.Via "statistics >TCP Stream Graphs > Throughput" one can generate the Throughput Graphs we used in section 6.2

Abbreviations

M-Lab Measurement Lab

NDT Network Diagnostic Tool

KS Kolmogorov-Smirnov

MWU Mann-Whiteney U

RTT Round-Trip-Time

SndCwnd Sender-Congestion-Window

rwndlimited receive-window-limited

applimited application-limited

BiF Bytes in Flight

GCS Google Cloud Storage

CC congestion control

CCA congestion control algorithm

PDF probability distribution function

CDF cumulative distribution function

RTO retransmission timeout

ECN explicit congestion notification

SACK selective acknowledgments

AAPC Average Annual Percent Change

MSAK Measurement Swiss Army Knive

MSS maximum segment size

List of Figures

List of Tables

Bibliography

[1] D. Abed, M. Ismail, and K. Jumari. "A Survey on Performance of Congestion Control Mechanisms for Standard TCP Versions." In: *Australian Journal of Basic and Applied Sciences* 5 (Dec. 2011), pp. 1345–1352.

[2] S. Bjorklund and L. Ljung. "A review of time-delay estimation techniques." In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. Vol. 3. 2003, 2502–2507 Vol.3. DOI: 10.1109/CDC.2003.1272997.

[3] H. Büning and G. Trenkler. *Nichtparametrische statistische Methoden*. Berlin, Boston: De Gruyter, 1994. ISBN: 9783110902990. DOI: doi:10.1515/9783110902990. URL: https://doi.org/10.1515/9783110902990.

[4] Bunny.net Academy. *What Are Congestion Controls and How Do They Work in Linux TCP?* Accessed: 2024-10-16. 2024. URL: https://bunny.net/academy/network/what-are-congestion-controls-and-how-do-they-work-in-linux-tcp/.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. "BBR: congestion-based congestion control." In: *Commun. ACM* 60.2 (Jan. 2017), pp. 58–66. ISSN: 0001-0782. DOI: 10.1145/3009824. URL: https://doi.org/10.1145/3009824.

[6] P. Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Berlin / Heidelberg: Springer, 2012. ISBN: 978-3-642-31163-5. DOI: 10.1007/978-3-642-31164-2.

[7] I. contributors. *Cross-Correlation*. https://www.investopedia.com/terms/c/crosscorrelation.asp. Accessed: 2024-10-03. 2024.

[8] W. contributors. *Cross-correlation — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Cross-correlation. Accessed: 2024-10-03. 2024.

[9] DATAtab. *Binomial Test - Einfach erklärt*. https://datatab.net/tutorial/binomial-test. Accessed: 2024-10-03. 2024.

[10] DATAtab. *Mann-Whitney-U-Test - Einfach erklärt*. https://datatab.de/tutorial/mann-whitney-u-test. Accessed: 2024-10-03. 2024.

[11] S. Developers. *scipy.stats.kstest — SciPy v1.11.3 Manual*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstest.html. Accessed: 2024-10-03. 2024.

[12] C. Dovrolis, K. Gummadi, A. Kuzmanovic, and S. D. Meinrath. "Measurement lab: overview and an invitation to the research community." In: *SIGCOMM Comput. Commun. Rev.* 40.3 (June 2010), pp. 53–56. ISSN: 0146-4833. DOI: 10.1145/1823844.1823853. URL: https://doi.org/10.1145/1823844.1823853.

[13] Editverse. *Understanding Statistical Correction: Bonferroni and More*. https://editverse.com/understanding-statistical-correction-bonferroni-more/. Accessed: 2024-10-18. 2024.

[14] P. Gill, C. Diot, L. Y. Ohlsen, M. Mathis, and S. Soltesz. "M-Lab: user initiated internet data for the research community." In: *SIGCOMM Comput. Commun. Rev.* 52.1 (Mar. 2022), pp. 34–37. ISSN: 0146-4833. DOI: 10.1145/3523230.3523236. URL: https://doi.org/10.1145/3523230.3523236.

[15] R. Hamming. "CHAPTER 4 - Classical Error-Correcting Codes: Machines should work. People should think." In: *Classical and Quantum Information*. Ed. by D. C. Marinescu and G. M. Marinescu. Boston: Academic Press, 2012, pp. 345–454. ISBN: 978-0-12-383874-2. DOI: https://doi.org/10.1016/B978-0-12-383874-2.00004-7. URL: https://www.sciencedirect.com/science/article/pii/B9780123838742000047.

[16] A. Hayes. *What Is the Bonferroni Test (Correction) and How Is It Used?* https://www.investopedia.com/terms/b/bonferroni-test.asp. Accessed: 2024-10-18. 2024.

[17] J. Hedderich and L. Sachs. *Angewandte Statistik: Methodensammlung mit R*. 17., überarbeitete und ergänzte Auflage. Section 7.4.8: Vergleich zweier unabhängiger Stichproben nach Kolmogoroff/Smirnoff, pp. 550-552. Berlin / Heidelberg: Springer Spektrum, 2018. ISBN: 978-3-662-62293-3.

[18] B. In. *How To Perform a Shapiro-Wilk Test*. https://builtin.com/data-science/shapiro-wilk-test. Accessed: 2024-10-03. 2024.

[19] Investopedia. *Variance*. Accessed: 2024-10-28. 2024. URL: https://www.investopedia.com/terms/v/variance.asp.

[20] V. Jacobson. "Congestion avoidance and control." In: *SIGCOMM Comput. Commun. Rev.* 18.4 (Aug. 1988), pp. 314–329. ISSN: 0146-4833. DOI: 10.1145/52325.52356. URL: https://doi.org/10.1145/52325.52356.

[21] M. Kerrisk. *Linux Programmer's Manual - tcp(7)*. https://man7.org/linux/man-pages/man7/tcp.7.html. Accessed: 2024-10-14. 2024.

[22] M. Lab. *ndt-server project*. Version v0.22.0. Accessed: 2024-09-03. 2024. URL: https://github.com/m-lab/ndt-server.

[23] M-Lab. *tcpinfo.go*. https://github.com/m-lab/etl/blob/7a4a5b65b51538240a220de218e13a1bee61parser/tcpinfo.go#L100. Accessed: 2024-10-29. 2024.

[24] K. MacMillan, T. Mangla, J. Saxon, N. P. Marwell, and N. Feamster. "A Comparative Analysis of Ookla Speedtest and Measurement Labs Network Diagnostic Test (NDT7)." In: *Proc. ACM Meas. Anal. Comput. Syst.* 7.1 (Mar. 2023). DOI: 10.1145/3579448. URL: https://doi.org/10.1145/3579448.

[25] Measurement Lab. *About Measurement Lab*. Accessed: 2024-08-06. 2024. URL: https://www.measurementlab.net/about/.

[26] Measurement Lab. *M-Lab Speed Test*. Accessed: 2024-10-08. 2024. URL: https://speed.measurementlab.net/%5C#/.

[27] Measurement Lab. *Measurement Lab*. Accessed: 2024-08-06. 2024. URL: https://www.measurementlab.net/.

[28] Measurement Lab. *Measurement Lab Frequently Asked Questions*. Accessed: 2024-08-06. 2024. URL: https://www.measurementlab.net/frequently-asked-questions/.

[29] Measurement Lab. *Measurement Lab Status*. Accessed: 2024-08-06. 2024. URL: https://www.measurementlab.net/status/.

[30] Measurement Lab. *TCP Info Test*. https://www.measurementlab.net/tests/tcp-info/. Accessed: 2024-10-14. 2024.

[31] Measurement Lab. *The M-Lab Google Cloud Storage NDT Data*. Google cloud storage gs://archive-measurement-lab/ndt. (2023-12-01 – 2023-12-02).

[32] Measurement Lab. *The M-Lab NDT Data Set*. https://measurementlab.net/tests/ndt. Bigquery table measurement-lab.ndt.ndt7. (2023-12-01 – 2023-12-02).

[33] Measurement Lab. *The M-Lab TCP INFO Data Set*. https://measurementlab.net/tests/tcp-info. Bigquery table measurement-lab.ndt.tcpinfo. (2023-12-01 – 2023-12-02).

[34] Measurement Lab. *Who We Are - Measurement Lab*. https://www.measurementlab.net/who/. Accessed: 2024-10-18. 2024.

[35] L. Metcalf and W. Casey. "Chapter 2 - Metrics, similarity, and sets." In: *Cybersecurity and Applied Mathematics*. Ed. by L. Metcalf and W. Casey. Section 2.9.2, Hamming Distance. Boston: Syngress, 2016, pp. 3–22. ISBN: 978-0-12-804452-0. DOI: https://doi.org/10.1016/B978-0-12-804452-0.00002-6. URL: https://www.sciencedirect.com/science/article/pii/B9780128044520000026.

[36] U. Z. Methodenberatung. *Mann-Whitney-U-Test - Methodenberatung in SPSS*. https://www.methodenberatung.uzh.ch/de/datenanalyse_spss/unterschiede/zentral/mann.html. Accessed: 2024-10-03. 2024.

[37] Microsoft Documentation. *TCP_INFO_V1 Structure (mstcpip.h) - Win32 apps*. Accessed: 2024-10-28. 2024. URL: https://learn.microsoft.com/en-us/windows/win32/api/mstcpip/ns-mstcpip-tcp_info_v1.

[38] A. Mishra, L. Rastogi, R. Joshi, and B. Leong. "Keeping an Eye on Congestion Control in the Wild with Nebby." In: ACM SIGCOMM '24. Sydney, NSW, Australia: Association for Computing Machinery, 2024, pp. 136–150. ISBN: 9798400706141. DOI: 10.1145/3651890.3672223. URL: https://doi.org/10.1145/3651890.3672223.

[39] National Cancer Institute. *Average Annual Percent Change (AAPC) Definition*. https://surveillance.cancer.gov/help/joinpoint/tech-help/frequently-asked-questions/aapc-definition. Accessed: 2024-09-25. 2024.

[40] T. Networks. *The Binomial Test*. https://www.technologynetworks.com/informatics/articles/the-binomial-test-366022. Accessed: 2024-10-03. 2024.

[41] T. Pangarkar. *Internet Usage Statistics 2024 by Network, Technology, Connectivity*. https://scoop.market.us/internet-usage-statistics/. Accessed: 2024-10-18. 2024.

[42] U. Paul, J. Liu, D. Farias-llerenas, V. Adarsh, A. Gupta, and E. Belding. "Characterizing Internet Access and Quality Inequities in California M-Lab Measurements." In: *Proceedings of the 5th ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies*. COMPASS '22. Seattle, WA, USA: Association for Computing Machinery, 2022, pp. 257–265. ISBN: 9781450393478. DOI: 10.1145/3530190.3534813. URL: https://doi.org/10.1145/3530190.3534813.

[43] U. Paul, J. Liu, M. Gu, A. Gupta, and E. Belding. "The importance of contextualization of crowdsourced active speed test measurements." In: *Proceedings of the 22nd ACM Internet Measurement Conference*. IMC '22. Nice, France: Association for Computing Machinery, 2022, pp. 274–289. ISBN: 9781450392594. DOI: 10.1145/3517745.3561441. URL: https://doi.org/10.1145/3517745.3561441.

[44] J. Pfeiffer. *What really happens on closing a TCP/IP connection?* https://linuxgazette.net/136/pfeiffer.html. Accessed: 2024-10-14. 2007.

[45] H. Rinne. *Taschenbuch der Statistik*. 3rd ed. Section 3.2.4 Binomialtests, page 530-531. Frankfurt am Main: Harri Deutsch, 2003.

[46] N. I. of Standards and T. (NIST). *NIST/SEMATECH e-Handbook of Statistical Methods: 3.5.6.3. Kolmogorov-Smirnov Goodness-of-Fit Test*. https://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm. Accessed: 2024-10-03. 2024.

[47] Statista. *Number of Internet Users Worldwide 2023*. https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/. Accessed: 2024-10-18. 2023.

[48] S. Sundaresan, M. Allman, A. Dhamdhere, and K. Claffy. "TCP congestion signatures." In: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17. London, United Kingdom: Association for Computing Machinery, 2017, pp. 64–77. ISBN: 9781450351188. DOI: 10.1145/3131365.3131381. URL: https://doi.org/10.1145/3131365.3131381.

[49] S. Sundaresan, X. Deng, Y. Feng, D. Lee, and A. Dhamdhere. "Challenges in inferring internet congestion using throughput measurements." In: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17. London, United Kingdom: Association for Computing Machinery, 2017, pp. 43–56. ISBN: 9781450351188. DOI: 10.1145/3131365.3131382. URL: https://doi.org/10.1145/3131365.3131382.

[50] Survey Point Team. *7 Reasons Why Statistical Testing is Essential for Your Research*. https://surveypoint.ai/knowledge-center/why-statistical-testing/. Accessed: 2024-10-18. 2023.

[51] Trading Technologies. *Relative Volatility Index*. https://library.tradingtechnologies.com/trade/chrt-ti-relative-volatility.html. Accessed: 2024-09-25. 2024.

[52] V. M. Vergara, C. Mayer, P. D. K. K. Kiehl, and V. D. Calhoun. "An average sliding window correlation method for dynamic functional connectivity." In: *Human Brain Mapping* 40.7 (2019), pp. 2089–2103. DOI: 10.1002/hbm.24509.

[53] B. Walther. *Shapiro-Wilk-Test in R rechnen*. https://bjoernwalther.com/shapiro-wilk-test-in-r-rechnen/. Accessed: 2024-10-03. 2024.

[54] R. Ware, A. A. Philip, N. Hungria, Y. Kothari, J. Sherry, and S. Seshan. "CC-Analyzer: An Efficient and Nearly-Passive Congestion Control Classifier." In: *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM SIGCOMM '24. Sydney, NSW, Australia: Association for Computing Machinery, 2024, pp. 181–196. ISBN: 9798400706141. DOI: 10.1145/3651890.3672255. URL: https://doi.org/10.1145/3651890.3672255.

[55] A. Wong and P. Vlaar. "Modelling time-varying correlations of financial markets." In: (Oct. 2003).