An overview on Measurement Lab

Ben Julian Riegel, Simon Bauer, Johannes Zirngibl*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ben.riegel@tum.de, bauersi@net.in.tum.de, zirngibl@net.in.tum.de

***Abstract*—**

In times when more and more technologies become connected over the Internet, proper monitoring and maintenance of this network is more important than ever. Measurement Lab collects such data on Internet-scale and provides a public database for this data set.

This paper will show what Measurement Lab is, how it works and especially how to work with it. Our analysis shows that Transport Layer Security (TLS) can influence speed test results of weak clients with low computational power. The paper will also report that during the COVID-19 pandemic, the data set was spammed by single contributors, which highly impacted the results.

***Index Terms*—measurement technology, internet speed tests, public data set, ndt, measurement-lab**

1. Introduction

Understanding and learning about the products you built, by monitoring the product and collection data about its behavior, is common practice. Just like this, network architects, researchers and consumers need accurate information on their Internet connection to properly maintain and monitor it.

The idea of Measurement Lab, short M-Lab, is to create a platform that satisfies consumers as well as researchers needs. Consumers receive information about how their specific connection performs, as well as researchers are provided with a publicly accessible database filled with information about worldwide broadband Internet performances collected by various open source services. Open source server software makes it possible for everyone to write own client sided software and offer it to the public to fulfill its own goals and simultaneously participate in the overall network of M-Lab. Also, this makes every service of M-Lab transparent and comprehensible [1].

M-Lab provides an infrastructure for the mentioned open source services to run. Therefore, they have servers spread all over the world. These servers collect all accruing data, which is generated by the services and put them together at one database. M-Lab also prepares data sets in different and useful views.

Since the platform was founded in 2009, it evolved and technologies became obsolete, the team behind M-Lab decided to completely rebuild the platform to so called M-Lab 2.0. With this update at the beginning of 2020, M-Lab retired services or completely rewrote them. All M-Lab 2.0 services now run on stock Linux kernel services, are dockerized (Docker: Software to isolate applications into containers) and are deployed on Googles container application management system Google Kubernetes Engine [2].

The remainder of this paper is to bring researchers closer to M-Lab. Therefore, Chapter 2 will present the technological background. Chapter 3 is based on this knowledge and presents an example of how to work with M-Lab's software, as it examines if TLS can influence the outcome of a speed test result. Chapter 4 will be dedicated to a wider view on the platform and the data. The paper will therefore show how to work with the raw data, as well as show how to use Big Query (a service by Google to manage and analyze large amounts of data) tables, M-Lab gives access to.

2. Background on M-Lab's Technology

Starting with a client who wants to know more details about his connection, M-Lab currently provides three services to choose from:

- Ndt (Network Diagnostic Tool):
  A test to find the maximum download and upload rates achievable.
- Neubot DASH:
  A test that emulates a video stream, to see how a connection performs under this circumstance.
- WeHe:
  A test how ones Internet connection handles traffic, collected from real world applications.

Since M-Lab is still changing a lot, this selection might differ over time. Moreover, the platform supports three core services. Core services are passively carried out all the time, when someone uses any service.

- Packet Header Service:
  Collects incoming packet headers for every incoming TCP connection.
- TCP Info:
  A service that collects data and creates statistics on all incoming TCP connections.
- Traceroute:
  A commonly known service that collects information about the network topology between server and client.

During the execution of one of the tests above, the M-Lab server saves all data which will be created throughout the execution to Google Cloud Storage.

doi: 10.2313/NET-2020-11-1_17

The raw data then will be passed through a pipeline, where the data is summarized and additional information is added. In specific, every opened TCP connection throughout a test receives an Universally Unique Identifier (UUID), generated by concatenating the server host name, server boot time and the TCP socket cookie. This UUID can be used later to match query results with the actual raw data or join main service results with a core service. Geolocation information for the client's IP address is added as well. by the M-Lab annotation service[1]. The data then is accessible through Big Query. Important to mention is, that not yet every data set is supported by Big Query. To receive access to the Big Query project, you need to join the M-Lab discussion group on Google Groups.

Comparing the amount of entries in M-Labs data base for the Ndt service with other services, Ndt, by far, is the most used service on the platform. This is why I will focus on Ndt and its analysis in the following.

2.1. Ndt

Since Ndt7 is the latest version of the Network Diagnostic Tool protocol, provided by M-Lab, this paper will focus on this technology as older versions might not be supported anymore in the near future.

The main goal of this protocol is to flood a single TCP connection between a well-provisioned server and a client, to measure the maximum possible application layer throughput rates for up- and download. Ndt7 is based on HTTP WebSockets (chapter 2.2). If the congestion control algorithm TCP BBR (chapter 2.3) is available, it takes advantage of it. Otherwise, it will just use the systems default. [3]

2.2. WebSockets

A problem with the HTTP protocol is the large amount of overhead a real time connection generates because HTTP is not originally made for continuous communication. In general, the client must always generate a new request and receives the response afterwards. Multiple requests will be independent from each other. Because the execution of a Ndt7 test will generate a real time traffic between server and client, a lot of overhead in packet headers throughout the execution will be generated as well. Because Ndt7 is made to approximate the application level performance and HTTP operates on the application layer, using pure HTTP may highly impact the results. This is why websockets are used here. [4]

WebSockets work as an upgrade of a HTTP connection. To upgrade a connection, the client must specify the `Connection` and the `Upgrade` fields in the HTTP header during a request. An example request could look like this:

GET /index.html HTTP/1.1 \r\n
Host: www.hostname.com \r\n
Connection: upgrade \r\n
Upgrade: websocket \r\n
... \r\n

1. https://github.com/m-lab/annotation-service

The server will ignore the upgrade request with a standard `200 OK` response if it is not capable of upgrading. `101 Switching Protocols` will be the Status code for a confirmation. [4]

From now on the communication will adhere to the WebSocket specifications. Client and Server can now use a bidirectional communication channel, which maintains on one TCP session. In contrary to a HTTP header, a WebSocket header now only requires 8 Bytes. Therefore, less overhead will be generated throughout a real time connection. [4]

WebSockets differentiate between three main types of frames. Binary, textual and control frames. The payload of binary frames will be interpreted as pure bytes. Textual frames contain payload in UTF-8 encoding. Control frames are e.g. used to close a WebSocket channel. [4]

The WebSocket protocol supports two Uniform Resource Identifier Schemes. `WS://` indicates a standard communication channel, whereas `WSS://` indicates a TLS encrypted communication. [4]

2.3. TCP BBR

The TCP congestion control algorithm Bottleneck Bandwidth and Round-Trip Time (BBR) promises to maximize the TCP-level throughput and median RTT. It was developed by Google in 2016 and implemented in Linux v4.9. Taking advantage of TCP BBR as a service provider is easy, as there is no need for any action on the client side. It must only be deployed on the server side. [5]

A path from server to client consists of multiple hops which all store the incoming packages in a buffered queue. They process them (e.g. routing) and forward them. Thus, the RTT on this path starts increasing when the buffer of the slowest hop fills up faster than it drains. If the buffer is filled up completely, packet loss will appear. [6]

In difference to loss-based algorithms, BBR periodically monitors the RTT and the delivery rates. From this data it creates a model which includes the recent maximum available bandwidth, and the minimum recent RTT. This model then will be used to calculate how fast BBR will send the remaining data. This way, the throughput can be adjusted before packet loss appears, and the bandwidth will be better utilized while optimizing the median RTT and also generating less overhead in retransmission. [6]

Consequently, the bandwidth bottleneck of a BBR connection is the bit rate of the slowest hop in the path, which is exactly what we want to measure with Ndt7 [3]. Also, the kernel level data generated by BBR will be stored and attached to the Ndt7 raw data on Google Cloud storage.

2.4. Ndt7 Protocol Specification

Ndt7 differentiates between the upload and download tests as two completely independent tests. Therefore, the following HTTP paths are specified for GET requests: `/ndt/v7/download` and `/ndt/v7/upload` [3]

The client starts by requesting the desired test with the corresponding path, while requesting a WebSocket upgrade. If no error occurs, the server will reply with the `101 Switching Protocols` status code. [3]

doi: 10.2313/NET-2020-11-1_17

When the WebSocket channel has been created successfully and the requested test was a download test, the server starts flooding the channel with binary frames, which must contain between 2^{10} and 2^{24} random bytes. If an upload test was requested, the client has to flood the WebSocket channel in this way. This frame size might be dynamically adjusted throughout the execution to better adapt to environmental requirements. [3]

A running time of up to 10 seconds for every execution is expected. During this interval, server or client are always permitted to provide textual frames, containing measured application level data in JSON format from its own side to support the counterpart with reliable speed measurements. E.g. these might be interesting if one would like to find out, how many bytes the counterparts application layer received of the acknowledged bytes on transportation layer.

These textual frames can be ignored completely from both sides as they only provide additional application-level information and are not mandatory to the protocol. If available, M-Lab servers attach these measurements under `Client Measurements` to the raw data. [3]

3. The influence of TLS on Ndt7

The specification itself claims that "Ndt7 should consume few resources" [3]. This gave the motivation why we wondered, whether TLS on weak clients with low CPU power, influences measurements, as it is the default choice for Ndt7 connections.

To test this, a stress test was set up, which would bring the protocol to its limits. A Raspberry Pi 2 Model B, running Raspbian Buster Lite release 2020-02-13, worked as a good representative of a weak client, according to the hardware specification[2]. It was directly plugged into a computer, running Ubuntu 20.04 LTS, using a cat 6 Ethernet cable. This computer provides way higher resources in terms of CPU power (i7-6700k) and network interface speed (1Gbit/s). The Raspberry Pi only offers a low-end CPU (ARM Cortex-A7) and a network interface with up to 100Mbit/s. The fast computer was running the official M-Lab server software[3] and BBR was enabled on the system. The Raspberry Pi was running a Ndt7 client software by Simone Basso[4].

The goal of this setup was to create a small network, where the results of a speed test would be given by the hardware of the Raspberry Pi, as it has the slowest hardware in every relevant point. If the computational power does not take any influences, we would expect the results to be capped by the network interface of the Raspberry Pi at around 100Mbit/s, no matter if the communication is encrypted or not.

The Client software was executed 10 times using the WS (no TLS) scheme and 10 times using the WSS (TLS) scheme. The client first executes a download test, followed by an upload test.

Figure 1 shows a boxplot of the measured TCP level throughput, using the data which was collected by the server throughout the tests. The median value

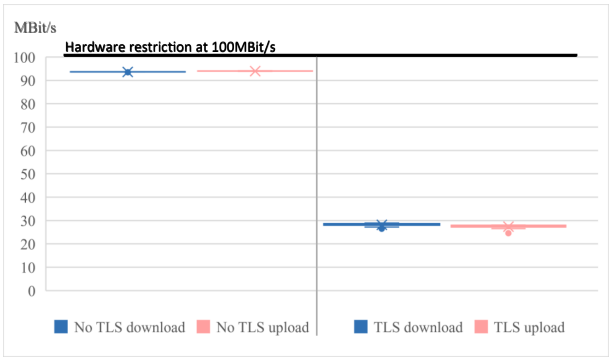

Figure 1: Box plot speed test results TLS vs. no TLS

(line between box borders), the mean value (cross), the 1st quartile (bottom border of each box) and the 3rd quartile (top border of each box) are represented in the Figure. Each plot includes 10 test executions to exclude measurement artifacts. We used the following method to calculate the results for Figure 1 from the raw data:

$$\text{Download} := \frac{AcknowledgedBytes_{max} * 8}{ElapsedTime_{max}} Mbit/s$$

$$\text{Upload} := \frac{ReceivedBytes_{max} * 8}{ElapsedTime_{max}} Mbit/s$$

As Figure 1 shows, for all results, median and mean value nearly line up. The boxes are represented as lines because the results only varied in the range of less than 1Mbit/s.

The median values of the non-encrypted tests are as expected around 100Mbit/s. The graph shows them at 93.6Mbit/s for the download and 93.9Mbit/s for the upload.

Interestingly, the encrypted tests revealed way lower results. The median is drawn at 28.5Mbit/s for the download and at 27.8Mbit/s for the upload.

As the encryption is the only thing that changed and encryption is a highly CPU consuming process, this highly indicates the low computational power of the Raspberry Pi to be the limiting factor in this connection. This would mean that speed test results can be influenced by how fast ones CPU is and does not necessarily show the maximum up- / download rate.

My suggestion would be to give weak clients the option to disable any kind of textual frames. So, no personal data would be leaked and they could safely execute non-encrypted tests, as this might be the only chance to receive results, which are not influenced by their CPU. Otherwise they might receive influenced results or reveal sensitive data.

4. Working with M-Lab

Switzerland was one of the first European countries, which considered taking down some of the most popular streaming websites during the COVID-19 pandemic because the internet congestion raised drastically. [7] To show how to access M-Labs data, this section will look at the up- and download speeds during the pandemic in Switzerland, to see if the mentioned increased internet congestion took a noticeable effect on peoples speed test results.

2. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
3. https://github.com/m-lab/ndt-server
4. https://github.com/bassosimone/ndt7-client-go-minimal

doi: 10.2313/NET-2020-11-1_17

This is done in two approaches. The first one is about raw data from M-Labs Cloud Storage and processing it. The second one takes advantage of the prepared views on the data by using M-Labs Big Query tables. The time span from 01/01/2020 until 05/24/2020 was selected for both approaches, as this was the most recent entry in the data set at that time.

4.1. Approach 1: Using raw data

This approach works with unprocessed raw data, which was directly downloaded from the public Google Cloud Storage. Ndt7 is not yet supported by a Big Query table.

Ndt7 raw data is saved in JSON format. Parsing the data to a MySQL Database and adding geolocation information about the client's IP address by using the GeoLite2 Databases by Maxmind, revealed that around 77% of the tests in this data set came from the USA. Only 5826 of the approximately 7 million total samples could be traced back to Switzerland in the total data set. Even though all European M-Lab servers do provide Ndt7 support already, seemingly not many clients have implemented the newest version yet.

Because an average of only 40 Samples per day can be influenced easily by single contributors, this approach could not be further pursued, as the result will be unrepresentative. Still, this section could come up with useful information, which will be helpful for future work with M-Lab.

4.2. Approach 2: Using Big Query

To receive a more representative result than in chapter 4.1, more samples would be required. Therefore, we could also use legacy data from older Ndt versions, which are supported by Big Query.

The Big Query project by M-Lab provides two tables called `unified_downloads` and `unified_uploads`, which will be used in this section. They combine data from legacy Ndt versions. Geolocation information has already been added.

A SQL request asking for the median download and upload rates per day in Switzerland, showed a result, where the median values drastically dropped on 03/10/2020. To find out more, the SQL request was expanded by the amount of total test requests per day.

This revealed that the unexpected behavior of the median rates seem to correlate with the total amount of daily test requests (#reqests). #requests instantly raised from an average of 5000 to over 25000 on 03/10/2020, while up- and download rates decreased at the same time. This unfiltered data can be found in the data directory.

Further examinations showed that single IP addresses contributed multiple speed tests per day to the data set, which must not be surprising, as people can tests their internet speed multiple times per day. But besides some clients, which were already contributing hundreds of results, there were 3 conspicuous IP addresses, which had not contributed any data before 03/10/2020. They started spamming thousands of low-end results per day into the data set, beginning exactly on that date.

To filter them out, the next SQL request only took the maximum contribution per IP per day in consideration. This request is plotted in Figure 2. Mind that the y-axis on the right side correlates with the amount of total requests and the left y-axis with the up- and download median values in Mbit/s.

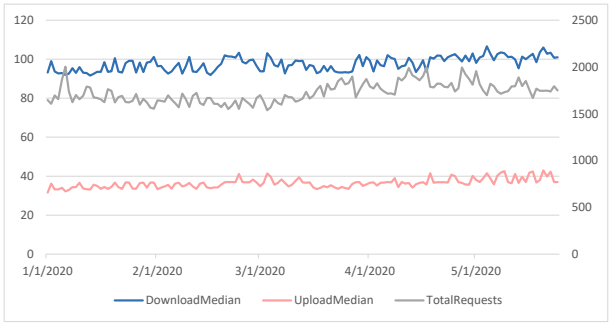

Figure 2: Throughput median in relation with #requests

According to this filtered data set, we can now see, that COVID-19 could not take very obvious influence on the median up- or download speed in Switzerland. The upload rates slightly fluctuate at around 97.6Mbit/s, while the upload values do so at around 36.7Mbit/s. No larger or unexpected outbursts are noticeable. In average, the download speed increased by 2.75Mbit/s from first half (01/01/2020 - 03/15/2020) to second half (03/16/2020 - 05/24/2020) of the time span. The upload did so by 1.69Mbit/s. More significant was the increase of #requests. This value has increased by an average of 145 for the second half.

In summary, we can not say that these decreased values at the beginning of the year necessarily have to do with the increased internet congestion during that time. The deviation is very small. The increased volume of #requests in the second half could be due to media coverage of the topic, so people wanted to test their internet speed, even though, it did not really decreased its median speed.

5. Conclusion and future work

After presenting the technological backround, the paper showed how to work with the open-souce software of the platform. It therefore revealed that a client can receive different speedtest results in the same enviroment, just by enabling the TLS encryption for the connection.

By presenting two possible approaches, the paper gave an idea on how to work with the M-Lab data set. The first one found out that for now, the platform lacks in sufficient Ndt7 sample data from europe. The second revealed a phenomena, where the median speedtest results in switzerland correlated with the amount of total requests made. This could be filtered out, but shows how the data can be influenced by single contributors.

In the future, it will be interesting to look at services like Neubot or Wehe and combine them with results from core tests, like Tranceroute.

Furthermore, it will be worth finding out more about the mentioned IP adresses, which were spamming the results from Chapter 4.2. Where are they located? This incidence might turn out to be related to COVID-19 in some way.

doi: 10.2313/NET-2020-11-1_17

References

[1] "Open Internet Measurement," https://www.measurementlab.net/about/, [Online; accessed 2020-05-28].

[2] S. Soltesz, "The 2.0 Platform Has Landed – Thank you!" https://www.measurementlab.net/blog/the-platform-has-landed/, 2020, [Online; accessed 2020-06-03].

[3] M-Lab, "ndt7 protocol specification," https://github.com/m-lab/ndt-server/blob/master/spec/ndt7-protocol.md, [Online; accessed 2020-06-03].

[4] *The WebSocket Protocol*, https://tools.ietf.org/html/rfc6455, 2011.

[5] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a Deeper Understanding of TCP BBR Congestion Control," 2018.

[6] G. Huston, "Open Internet Measurement," https://blog.apnic.net/2017/05/09/bbr-new-kid-tcp-block/, 2017, [Online; accessed 2020-06-03].

[7] "Netze wegen Corona-Krise überlastet: Müssen die Schweizer bald Netflix & Co. abschalten?" https://www.rtl.de/cms/schweizer-netze-in-corona-krise-ueberlastet-netflix-abschaltung-droht-wie-ist-es-in-deutschland-4506430.html, [Online; accessed 2020-12-08].

doi: 10.2313/NET-2020-11-1_17