Project report

Portfolio 1

Name: Behdad Nikkhah
ID: s362085

OSLOMET

Date of submission: 17.04.2023

Table of contents

1. Introduction 3

2. Implementation of Simpleperf 4

3. Experimental setup 5

4. Performance evaluations 7

4.1 Network tools for performance evaluation 7

4.2 Performance metrics 7

4.3 Test case 1: Measuring bandwidth with iperf in UDP mode 8

4.3.1 Results 8

4.3.2 Discussion 8

4.4 Test case 2: Link latency and throughput 10

4.4.1 Results 10

4.4.2 Discussion 10

4.5 Test case 3: Path Latency and throughput 12

4.5.1 Results 12

4.5.2 Discussion 12

4.6 Test case 4: Effects of multiplexing and latency 13

4.6.1 Results 13

4.6.2 Discussion 13

4.7 Test case 5: Effects of parallel connections 15

4.7.1 Results 15

4.7.2 Discussion 15

5. Conclusion 16

6. Sources 17

1. Introduction

This report contains a detailed walkthrough on topics regarding network performance, and the means of evaluating such concepts. To evaluate the performance of the given virtualized network, I have to implement a program called simpleperf, which is a simple version of iperf. This program will be built in python, using socket module and TCP as protocol to transmit data. Another relevant work mentioned in this report includes Mininet. This tool is used to build a virtualized network following the network topology and allows me to perform and evaluate different tests.

The tests incorporate different methods of problem solving within networking. This involves measuring throughput in UDP and TCP, calculating latency, and validating the fairness of network resources, among other things.

This experiment contains limitations on some areas such as iperf. Iperf is a useful tool for measuring network performance in the form of a protocol called UDP. This is a protocol which I have not implemented in my simpleperf code, due to the task's limitations. Instead, I have used TCP which is more reliable than UDP. Another limitation is the use of a virtual network, which may not be an accurate representation of network conditions in the real world.

This document consists of six main chapters, including the list of references at the end. I will start off by thoroughly describing my simpleperf implementation. I will then explain the network topology given for this task and its devices. Chapter four contains descriptions of the network tools and performance metrics, which give a basic understanding of the terms used in the report. Test case one to five make up for the rest of chapter four and serves as the main body of this report. The end will naturally include a conclusion and a list of sources.

By using the virtualized network to compare my expectations to the actual results, I will in this report thoroughly discuss my findings and include theoretical arguments.

2. Implementation of Simpleperf

My simpleperf code is written and designed in python-programming language, with the main purpose of measuring network performance. The code runs in two different modes: *client mode* and *server mode*. When running in client mode, it transfers data in chunks of 1000 bytes. The server then calculates (1) the transmitted data received from the client, (2) the total time duration, and (3) the throughput – also known as the bandwidth. The performance statistics are then displayed on both the client and the server. The code utilizes socket programming for the communication between the two modes. Simpleperf also uses TCP, which is a reliable protocol that retransmits data and lost packets. However, this reliability comes at the cost of the network performance, more specifically the speed (ExtraHop, n.d.).

The server mode implements two dependent functions: *server* and *handle_client*. The server functions bind the server to a specified ip-address and port number, and listens for any incoming client connections. When a client connection is accepted, it starts to run handle_client in a separate thread.

Firstly, the handle_client receives a message regarding start time from the client, which is then followed up by data of 1000 bytes. Then it continues to receive data until it collects a 'bye' message. When this happens, the handle_client prepares an acknowledgement and sends it back to the client. Next, it closes the connection socket, calculates the transmission rate, and prints the performance statistics in specified format.

The client mode also implements two functions: *client_send_data* and *client*. When we are running the client function, it creates a number of new socket objects and then connects to the server-ip and port number. This number is chosen by the client with *-P* command, which is 1 by default. Then it transfers data in chunks of 1000 bytes in separate threads by running client_send_data.

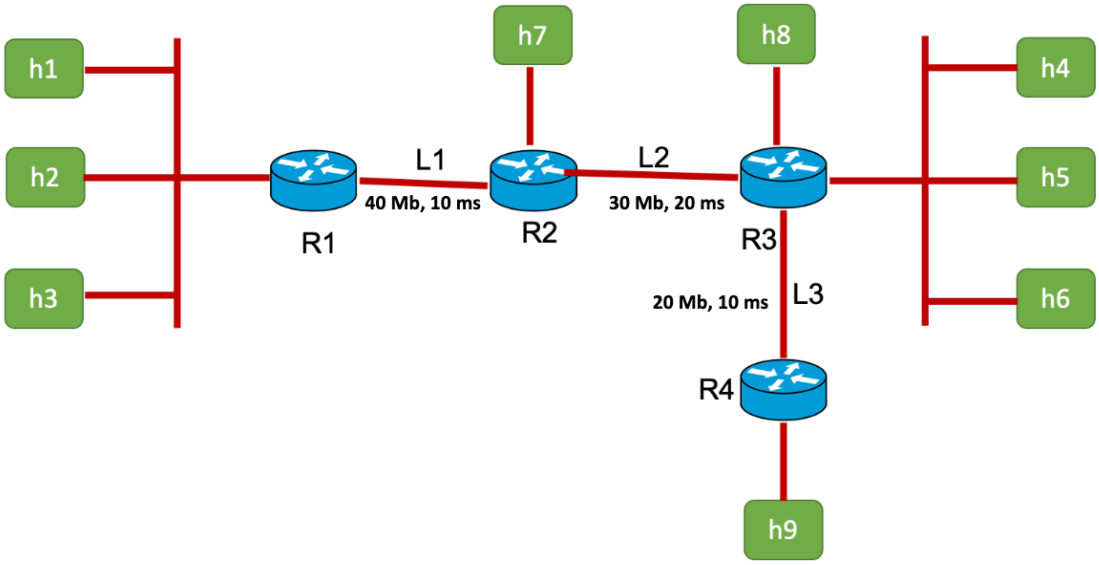
Following this, client_send_data sends a start message and transfers data. The data is transferred until the specified time duration ends, or the specified number of bytes is transferred. When all the data is transmitted, it will send a 'bye' message and wait for an acknowledgment. When it receives the acknowledgment, it prints the performance statistics in the correct format to the client terminal.

In my simpleperf implementation, I also have a function called *print_stats_interval*. The function calculates the elapsed time since the last printed interval, the bandwidth used during the time interval, and the converted bytes in specified format. Then it prints all of these calculations to the terminal. This function is called by client_send_data only if the users specify interval with the -i flag.

To summarize, simpleperf is a python-based code, which calculates network performance as throughput. The current program is built with socket and threading module, which is running in two modes. When running in server mode, the program waits and listens for any possible connections. Meanwhile, in client mode it initiates connections to the server and transmits data.

3. Experimental setup

For this report, Mininet is utilized to make a virtual network using the network topology as a base-structure, and python as the programming language. This is a visual representation of the network topology:



The network topology contains different devices such as two switches, four routers and multiple hosts. Both the hosts and the remaining devices are all connected to the virtual network made with Mininet. All the hosts within this network have a unique ip-address, which makes it possible for them to communicate with one another, including the routers. The routers have three separate links (link1, link2 and link3) which connects them to each other.

This virtual network also contains different subnets, which is a separated piece of a larger network. Subnetting is done to divide a single physical network into smaller segments, thus creating multiple logical sub-networks (Pandey, 2022). In this network topology, all subnets have different network performance represented by link bandwidth and latency. Both are useful information when measuring various delays and throughput in the simulated network.

Delay is defined as the amount of time a packet takes to transfer from the source to a destination. Network delay is divided into three main categories: transmission delay, propagation delay and processing delay. Propagation delay is used when evaluating the latency in this topology, and it measures the time it takes for one bit to move from one side of the link to the other. The routers in this topology also include processing delays, which are included in the total time for network delay. One important thing to notice, is that every link has a specific max queue size, which determines the total maximum of packets that can be transferred through the link. In cases where the link experiences overload due to excessive data, the subsequent data will be placed in a buffer. Here, the data waits to be retransmitted. If the buffer also reaches its maximum capacity, it will start to drop any new coming packets. This indicates packet loss (Islam, Performance Evaluation, Powerpoint, 2023).

The network topology contains eight subnets, ranging from A to I.

A. The subnet **A** contains three different hosts (H1, H2, H3), a switch and one router. The hosts are all connected to switch one (S1). S1 is then connected to router one (R1) by a link with the interface of *r1-eth0* (the correct interface of router one). H1 has a unique ip-address (10.0.0.2/24), H2 (10.0.0.3/24) and H3 (10.0.0.4/24).

B. The subnet **B** also contains R1 in addition to a second router, router two (R2). These routers are connected through a link, link one (L1). The links bandwidth is 40mbps and has a latency of 10ms, while the maximum queue size is 67 packets.
C. The subnet **C** contains router two (R2) and the host (H7). The ip-address of the host is 10.0.2.2/24, which is connected to R2 through a link with interface *r2-eth1*.
D. The subnet **D** contains router R2 and router three (R3). The routers are also connected through another link, link two (L2). The maximum transmission rate of the link is 30mbps and has a latency of 20ms. The maximum queue size is 100 packets.
   The **unnamed** subnet contains the router three (R3) and the host (H8). The host has the ip-address of 10.0.4.2/24 and is connected through the R3 interface *r3-eth1*.
E. The subnet **E** consists of one router (R3), and three different hosts (H4, H5 and H6), in addition to a switch. H4 has the ip-address 10.0.5.2/24, H5 10.0.5.3/24 and H6 10.0.5.4/24. The hosts are all connected to switch (S2), and then the switch is connected to R3 through a link by the interface *r3-eth2*.
G. The subnet **G** contains two different routers, R3 and R4. The routers are connected through link three (L3) which has a bandwidth of 20mbps, and a latency of 10ms. The maximum number of packets supported in this link is 33.
I. The subnet **I** contains R4 and the host H9, which has the ip-address of 10.0.7.2/24. The following host is connected to the router through a link with the interface *r4-eth1*.

To summarize, this is a virtualized network topology which is using Mininet and python as a building programming language. This gives us the opportunity to experiment repeatedly in controlled and safe network characteristics, performances, and optimizations.

4. Performance evaluations

4.1 Network tools for performance evaluation

Iperf is a bandwidth monitor tool which, in addition to other things, measures network performance by generating UDP and TCP traffic-protocols between two hosts. Between a server and a client, this monitor tool can be used between the hosts´ to measure the maximum network bandwidth (throughput). While bandwidth and throughput both give an indication of the network performance, it is important to notice a slight difference between the two. The bandwidth indicates the capacity of the path, while throughput refers to the transmission rate. In most cases, the throughput will be lower than the bandwidth (Roberts, 2022). Another thing to notice, is that iperf is in this report only used in UDP mode to measure the hosts´ bandwidth and to calculate any potential packet loss.

Ping is a network tool, which is used to measure the round-trip time of a message sent from one host to a specific destination. The ping sends an *echo request* message to a destination host and measures the round-trip time once received back. More specifically, ping operates by Internet Control Message Protocol (ICMP) which sends an ICMP echo request and waits for an ICMP echo reply (Islam , mininet-lab-intro.pdf, n.d.). Ping also reports the total percentage of the packets lost and received, and the total time of the packets being transmitted. In this report, ping is used in almost all of the experiments.

Simpleperf is a simple network tool build in python – which measures network performance. This is done by transferring data in chunks of 1000 bytes between two host, a client, and a server. The code measures the throughput of the data transmitted, by using the total amount of data transferred in a specific time. This will eventually calculate the final throughput between the client and the server. This tool has been used in all experiments except the first one.

Mininet is a tool used to create a virtual network through software. Its virtual hosts, links, switches, and controllers are realistic and have a similar behavior to the elements of hardware (Mininet, n.d.). This is used to create a virtual network following the setup of the network topology, which provides the basis needed to complete and evaluate the different experiments in this report.

4.2 Performance metrics

To simulate different experiments with simpleperf, the output will print different network performance statistics such as the bandwidth/throughput, transfer, interval, and the duration.

The **bandwidth** measures the total amount of data transmitted over a specified duration. Simpleperf calculates the bandwidth in megabits per second (Mbps). The higher the bandwidth is the faster is the transfer of data between the client and the server.

Transfer is the total amount of data transmitted between the client and the server, during a single transfer request. The default setting is printing the amount in MB – megabytes, but the client has the opportunity to choose the format that benefits them. This metric can share information regarding the transfers success.

The simpleperf code can print the statistics in different **intervals** chosen by the client. The interval calculates the transmission rate (bandwidth), and the total amount of bytes sent in each interval. Additionally, it evaluates the transferred data by its speed and size, estimating its success.

Duration is the total amount of the time where the data has been successfully transmitted from the client to the server. The duration is calculated by the end time minus the start time. It is a metric that can help us evaluate the efficiency of the transfer procedure.

4.3 Test case 1: Measuring bandwidth with iperf in UDP mode

In this task, I am going to measure the bandwidth with iperf in UDP-mode. Firstly, I must open two separate terminals, which both contain two different hosts. This task also depends on which hosts we are sending and receiving packets from. Following, I need to choose which one of the two is the server and the client. Usually, the one who is receiving is the server part. To gain more understanding of this concept, I will use the first test (between H1-H4) as an example of how I will approach this experiment. In this case, H1 is the client and H4 is the server.

One important thing I must do is to specify what protocol we are sending packets from, since Iperf is by default sending TCP. However, since the task specifically asks for UDP as a protocol, we must state this with *-u* command in the terminal for both server and client. Next, I must specify in the client mode which transmission rate I want to transfer packets to the server. This is done by using the *-b* command and then choosing *XM* (X stands for chosen rate).

4.3.1 Results

Test	Host's	Bandwidth (throughput)
Test 1:	H1 – H4	26.2 Mbits/sec
Test 2:	H1 – H9	16.8 Mbits/sec
Test 3:	H7 – H9	16.8 Mbits/sec

4.3.2 Discussion

Test 1

Since we are sending packets from H1 to H4, we must go through link 1 (L1) and link 2 (L2). L1 contains 40Mbps and 10ms latency, while L2 has 30Mbps and 20ms latency. L2 is thus the bottleneck in the path between H1 and H4, which sets a limitation of the network performance. A bottleneck is a particular area which, compared to the rest of the network, is limited due to a lower bandwidth or capacity (Rouse, 2016). Sending a transmission rate greater than 30Mbps will therefore result in queuing delays, even if L1 is capable of handling more.

To make sure that the packets are transmitted steadily, I thought that the estimated rate I should use is 30Mbps, but in doing so I lost 7% of the transmitted packets. When running multiple tests', I found out that the ideal rate I should choose is 25Mbps. With 25Mbps as rate, none of the packets transmitted between the hosts would be lost. During the tests, I noticed that using 20Mbps, or even 27Mbps, would not make me lose any transmitted packets. A low rate such as 20Mpbs would end up transferring 25MBytes – which is not wrong per se, but it would not be as efficient as 25Mbps. On the other hand, 27Mbps would deliver around 33.8MBytes. The last solution would be the most efficient, because it would transfer the most data and not lose any packets. However, I have to keep in mind that network trafficking and processing delay *can* result in losing packets at 27Mbps. Considering all this, I thought that both the safest and most efficient transmission rate to measure the bandwidth is 25Mbps. Even when using this rate, I noticed that the bandwidth increased with one (1) in almost all cases. In instances where I used 27Mbps as rate, the result in the bandwidth would, for example, be 28.3mbps.

Test 2

In this test we are sending packets from H1, the client, to H9, the server. The transmitted packets are in this case traveling between L1, L2, and L3. The bandwidth and the latency of the first two links are as mentioned above, meanwhile L3 contains 20Mbps and 10ms latency. This makes L3 the bottleneck in this particular test, which means that the maximum transmission rate is 20Mbps. By using the same calculations as above, I began with using this as the maximum transmission rate, and in doing so I lost 7.6% of the packets transmitted. While calculating the ideal rate for me, I chose the same approach as the first test. I

calculated that 10Mpbs would be non-efficient, meanwhile 18Mbps would be too risky. I ended up choosing 15Mbps as the most efficient and safest option.

Test 3

This task specifies sending packets between H7 and H9, where the client is H7 and the server is H9. The path between these hosts is connected by L2 and L3. The bottleneck is once again L3 – making the highest transmission rate possible 20mbps. Having learned from the two first tests, I knew for a fact that sending 20Mbps would not work. Similar to test 2, I used 15Mbps as my chosen rate – it is efficient, not too low or too high, and will not cost me any packet-loss.

Measuring the throughput without knowing the network topology ...

If I were to be asked to use iPerf in UDP-mode to measure the bandwidth without knowing any information of the network topology, I would use the *try and fail* method. By doing this, I will try different but repeated attempts at problem solving until success is achieved or I stop trying. In my case, this means running multiple tests while using different transmission rates to see how my network responds and reports.

To begin with, I need to run multiple tests. One option is to start off with a low rate, and then slowly increase the transmission rate to see how my network responds and calculates the final throughput. The other option would be to start off at a high rate to see how much data will be lost at a certain point. Considering these two methods, I would prefer to start at a high rate. This method is in my opinion the most efficient because it gives me an overview of the highest achievable data transmitted, without having to lose too many packets in the process.

When I reach a point where I start losing packets, I need to trace the pattern back and reach a point where the maximum transmission rate is efficient and safe. The same goes for the case in which I do not lose any packets – I would increase the transmission rate slowly until I reached an ideal point of efficiency and safety. In the end I would be able to calculate the average transmission rate I can send, and what the maximum would be before losing packets.

However, this method will not be a sufficient nor practical way to estimate the bandwidth in larger and more complex network. A lot of time is also needed to calculate and test out the best transmission rate until I reach the wanted result. The accuracy would also depend on the amount of the links and paths between the source and the destination. In larger networks it would be better to use more advanced network tools and a different method to measure the network bandwidth. It is important to keep in mind that UDP is an unreliable network protocol, and I do not know for sure that all packets transmitted will be received. Therefore, it will be more practical to make use of the TCP network protocol.

The reason why I thought the *try and fail* method is reasonable to use, is because I assumed the unknown network topology is similar to the one we have worked with in this report. However, the task states that I have no information about the network topology, so my approach is not practical in more complex networks.

4.4 Test case 2: Link latency and throughput

In this test, I am going to measure the link latency (round-trip-time) and the throughput from the three induvial links (L1-L3) between routers (R1-R4). The RTT is the time it takes for a packet to be transferred from one end of the network to the destination, and back again. An important aid is the network tool *´ping´*, which will aid me in finding the RTT-time for the three links. The command *´ifconfig´* will give me the unique ip-address of each router. The command *´net´* in Mininet will help determine the right interface which the routers are connected to. For example, the R1 router is connected to R2 through interface *r2-eth0*, which is the right ip-address of R2. An important thing to note, is that we have to transfer 25 packets to measure the correct RTT. This is done by typing the *´-c´* command.

To measure the throughput, I have to run my simpleperf code in 25 seconds. The simpleperf is running in two modes, client and server. For example, R1 is the client transferring data while R2, the server, receives. To invoke the client, we have to specify the command *´-c´* and to invoke the server *´-s´*. It is important to bind the right ip-address using the same method mentioned above, because it is by default *'127.0.0.1'*. The port will continue to run on default, which is *´8088´*.

4.4.1 Results

Link	Routers	Latency (RTT)	Throughput (bandwidth)
L1	Between R1 – R2	21.855ms	38.02Mbps
L2	Between R2 – R3	44.678ms	28.43Mbps
L3	Between R3 – R4	22.815ms	19.05Mbps

4.4.2 Discussion

Latency and throughput for link 1

L1 has 10ms of delay, so when sending 25 packets from R1 to R2 I would estimate a delay of around 20ms or even slightly higher. In the final output I got from my performance evolution, the average RTT was 21.885ms, which was slightly higher than 20ms.

In this test I am going to transfer data from R1 to R2 through L1, which has 40Mbps. The expected throughput should be slightly lower than 40Mbps. The throughput ended up being 38.02Mbps.

Latency and throughput for link 2

By using the same method, I must now use L2 as a traveling path to transfer packets from R2 to R3. Link 2 contains 20ms of delay, and will thus be expected to have a delay slightly higher than 40ms by looking at the topology. While evaluating the performance, the average RTT was 44.678ms.

The maximum transmission rate of link 2 is 30Mpbs, and is therefore expected to have a throughput slightly lower than 30Mbps. As the test reports, it has an output of 28.43Mbps in the throughput.

Latency and throughput for link 3

The conduction of this test will follow the same principles as in test 1 and test 2. In this case, link 3 will be the traveling path which has 10ms of delay. Considering this, the expected delay would be around 20ms. The result of this test had an average RTT of 22.815ms in latency, as what I expected.

The throughput for this test should be slightly lower than 20Mbps, due to the maximum transmission rate of link 3 which is 20Mbps. The result of the throughput was 19.05Mbps.

Compared to L1 and L3, L2 has the highest expected link delay, which is seen by locating the average RTT output. When looking at the previous tests, I noticed that the throughput declined with 2Mpbs compared to the maximum transmission rate. L1 and L3 both have 10ms of delay, but the bandwidth of L1 is higher. I thought that this would be the link with the fastest throughput, and it did not match my expectations. The reason may be that L1 – which is between R1 and R2 – contains a total of four hosts. Meanwhile L3 – between R3 and R4 – contains only two hosts. This will eventually result in a faster processing delay in L3, near the maximum achievable throughput.

There are other reasons to not expect the maximum throughput, such as potential packet loss and network traffic. When transferring data by using my code, I am using the TCP-protocol which has built in network-congestion control (ExtraHop, n.d.). The control makes sure that the network is not congested and divides the network performance amongst all users. In this case, there is no need to put much importance in the network congestion since we are only testing the links´ independent responses. A throughput close to the maximum transmission rate can in these experiments also indicate a low packet loss and network trafficking.

4.5 Test case 3: Path Latency and throughput

In this test I am going to measure the latency and the throughput of the specified paths – the hosts. This test would be similar to test case 2, except that I am now measuring the path instead of the link. The RTT in this particular case is between two specified hosts. The data will be transferred from one of the hosts to another, and back again. This process will determine the latency of the path. As mentioned above in test case 2, I will be executing the same commands and parameters, but with the purpose of evaluating the hosts.

4.5.1 Results

Which test	Path	Latency RTT	Throughput (bandwidth)
Test 1	H1 – H4	69.606ms	27.57Mbps
Test 2	H7 – H9	69.404ms	18.03Mbps
Test 3	H1 – H9	92.323ms	17.29Mbps

4.5.2 Discussion

Latency and throughput for test 1

The expected latency should, according to the network topology, be the path between L1 and L2. From H1 to H4 the latency is 30ms, and when traveling back it supplies an additional 30ms, making it a total of 60ms of latency. The actual output ended up at 69.606ms of latency.

The excepted throughput is slightly lower than 30Mbps. L2 is the bottleneck and the maximum achievable throughput for the path is therefore 30Mbps. The result of the output was 27.57Mbps.

Latency and throughput for test 2

For this test, the path between the hosts is connected by L2 and L3. Similar to test 1, the total latency is expected to be around 60ms. The report came back with a latency of 69.404ms.

The expected throughput should be slightly lower than 20Mbps. The bottleneck in this case is L3, containing 20Mbps and making it the highest achievable bandwidth. The calculated throughput was 18.03Mbps.

Latency and throughput for test 3

In this test, the latency between H1 to H9 should be around 80ms. All the three links in the network topology will be used in this instance. The added latency between these three links is around 80ms. The final output for the latency was 92.323ms.

The expected throughput for this last test with L3 as the bottleneck, should be around 20Mbps. The final output is 17.29Mbps, which corresponds to my expectation.

Most of my expectations matched my result in this test case. In the small instances where I experienced irregularities, it could be due to factors such as network congestion, processing delay and the network being virtualized. Due to us transferring data between the hosts, it is normal to expect a higher processing delay on the routers. Since the routers can contain multiple hosts, they use extra time to process information on who is transferring and who is receiving data. Another factor contributing to an affected latency, could be because of the network being virtualized. The performance of a virtualized network can be limited due to a number of reasons. One of these is because of the virtualization layer, which adds on the latency compared to a non-virtual network. It also limits the potential resources that the hosts can benefit from. Similar to test case 2, the throughput keeps near the links´ bandwidth. This could be because the potential network congestion and trafficking has a low effect.

4.6 Test case 4: Effects of multiplexing and latency

In this experiment am I going to look for the effect of multiplexing and the latency, where many hosts want to communicate simultaneously. The same commands I have used on the previous cases to measure the throughput and the latency, are also going to be used in this case. The only difference is that we are operating different servers and different hosts, in which both have two operating terminals. Taking test one as an example, H1 is operating two terminals which are simultaneously pinging and transferring data to one H4 terminal, using simpleperf. At the same time, H2 is also operating the same way to H5.

In this test case, I am also implementing JFI. Known as a metric, the Jain´s fairness index measures the performance of a user or application in distributing resources among traffic flows. To determine a fair share of the system resources, the metric returns a value between zero and one – where zero is a low rate of fairness and one is very fair (Lee & Chuah, 2015). To evaluate the fairness of the two connections, I have to implement the JFI metric.

4.6.1 Results

Test	Paths	Latency (RTT)	Throughput (Bandwidth)
Test 1	H1 - H4	96.784ms	13.73Mbps
	H2 - H5	97.071ms	15.00Mbps
Test 2	H1 – H4	101.251ms	10.23Mbps
	H2 – H5	102.070ms	7.91Mbps
	H3 - H6	104.231ms	11.65Mbps
Test 3	H1 – H4	98.571ms	14.78Mbps
	H7 – H9	99.721ms	13.90Mbps
Test 4	H1 – H4	85.571ms	28.39Mbps
	H8 – H9	38.819ms	19.09Mbps

4.6.2 Discussion

Test 1

In this first test, I am communicating simultaneously using the simpleperf on the hosts H1 – H4 and H2 – H5. I am going to both ping 25 packets and transfer data in 25 seconds to measure the latency and throughput. All hosts are using the same pipe by going through L1 and L2, making the expected latency 60ms, and the expected maximum transmission rate to 30 Mbps. However, because of the simultaneous connections, I should expect a higher latency.
The result of the measured latency of H1 – H4 is 96.784ms, and for H2 – H5 it is 97.071ms. The throughput may be affected by network congestion, since we are using simultaneous communication. The hosts have to share the network resources and its performance. The result for the measured throughput of H1 – H4 is 13.73Mbps and H2 – H5 is 15.00Mbps. This test has a JFI value of 1, which indicates an ideal fairness-allocation of the network resources.

Test 2

In this second test, I am experimenting simultaneous communication between the host H1 – H4, H2 – H5, and H3 – H6. Since all the hosts are going through the same pipe as the previous test, the expected latency should be slightly higher than the previous test. The throughput should however decrease.
The actual result of the experiment is a higher average RTT and a decreased throughput. The throughput and the latency of H1 – H4 was respectively 10.23Mbps and 101.251ms, for H2 – H5 it was 7.91Mbps and 102.070ms, and lastly H3 – H6 with 11.65Mbps and 104.231ms. This specific experiment had a JFI value of

0,98. This indicates that an almost perfect shar of fairness of the network resources, but not as fair as the previous experiment.

Test 3

In the third test, I am experimenting simultaneous connection between the following host: H1 – H4 and H7 – H9. All links in the network topology will be used. However, L2 will be the most occupied link in the topology, since it is the one occupied by most data. This includes ping data and transfer data from simpleperf sent from H1 and H7. The expected latency of both hosts remains the same as in the previous tests, meanwhile the maximum achievable transmission rate is different for both pairs.
The latency has slightly decreased compared to test 2, but still remains high. The throughput has also decreased more than expected – which is the effect of multiplexing. Since both hosts are operating in the same way through L2, it is processing more data and has a higher link delay compared to the other links. The JFI value of this test is 1, indicating that the hosts are sharing a fair amount of the network's performance.

Test 4

In the fourth test, we are simultaneously experimenting on the connection between H1 – H4 and H8 – H9. The expected latency between the first hosts should be 60ms with a maximum throughput of 30Mbps, and 20ms and maximum throughput of 20Mbps between the second hosts. The latency result of H1 – H4 ended up at 85.571ms, with a throughput of 28.39Mbps. The latency of H8 – H9 was 38.819ms and the throughput was 19.09Mbps. The increase of latency on both connections is due to the processing delay on R3. The router is both handling two different ping messages, and one data transfer from H1 – H4. For the router to be able to process both of the separate ping messages, the average RTT for both hosts are increased, since the router has to find out which of the hosts are pinging and which are transferring data. This test had a JFI value of 0.99.

Compared to my expectations, the findings of this test case vary from increasing and decreasing results. The throughput is in most cases affected by multiplexing, which gathers and combines data chunks at the source from different applications and delivers it to the network layer (Islam, Transport layer, 2023). The RTT is then affected by increased network trafficking which leads to congestion of the network and delays in packet delivery. When we are not multiplexing, the average RTT remains low since only the ping message is sent. However, in this case, the experiments are sending pings *and* transferring data using simpleperf. Since we are using multiplexing, we are doing multiple things at once which can lead to packet loss and queuing delay. When a link is overloaded, it has to transport the remaining data to buffers. These packets have to be re-transmitted, which is also contributing to the decrease in the final latency and some of the throughput. As seen in test 2, the result of decreased throughput is mainly due to network congestion because multiple hosts are using the same link to transfer data. They have to share the network performance resources – transmission rate. In the case of the final test, we can see that the two following hosts´ throughput is not affected by congestion and can thus deliver close to the maximum by using different paths. This is not the case for the three tests previous to number four.

4.7 Test case 5: Effects of parallel connections

In this test case, I am going to use simpleperf to measure the throughput of the following host H1, H2, and H3, which are simultaneously connected to the receivers H4, H5, and H6. The former will run in client mode, meanwhile the latter will run in server mode. One important thing to notice, is that H1 is simultaneously communicating with H4 through two parallel connections. That means we now have four connections, where the three individual hosts are transferring data.

4.7.1 Results

Path	Throughput (bandwidth)
H1 – H4, parallel connection one	6.04Mbps
H1 – H4 parallel connection two	7.88Mbps
H2 – H5	7.48Mbps
H3 – H6	7.82Mbps

4.7.2 Discussion

The expected throughput should in this case be lower than the maximum achievable transmission rate, which is 30Mbps. This is due to all hosts using the same path, which is through L1 and L2. L2 is the bottleneck which contains the maximum achievable rate as mentioned in the last sentence. In theory we have three connections through the paths, but since H1 is using a parallel connection, it increases to four different connections. The result of the test was for each of the three hosts around 7Mbps, meanwhile the first parallel connection has 6Mbps. All connections are evenly matched, and we can evaluate the shared fairness by using the JFI metric. The JFI value for this test is 0.99, indicating that the hosts are getting a fair share of the network performance.

To summarize, the three connections between the hosts are sharing the available bandwidth. Adding the parallel connection between H1 and H4, results in an even more decreased throughput. The congestion in the network is more affected by the added parallel connection, versus if all hosts were transferring data independently. Potential packet loss is also expected, since we are using the simpleperf code which transfers data using TCP protocol. This result in lost packets automatically being retransmitted, which also is a cause in decreased throughput.

5. Conclusion

In this report, I have used various tools to evaluate the virtual networks´ performance. This is done by calculating the average RTT and a source´s throughput. In this network topology, the sources can be described as routers, hosts and links. Tests of these segments can provide an approximate assessment of the network´s performance. After performing these tests, I noticed that this network usually responded in correspondence with the expectations (in most cases). It also managed to share a fair amount of resources when there was increased traffic in the network.

Unfortunately, I experienced difficulties regarding the multiplexing experiments. My linux virtual machine, which was the core of being able to execute my experiments, reported back unrealistic numbers of the RTT. The solution was to install another virtual machine, called Debian. This machine buffered the remaining data transmitted, giving us a more realistically calculated response. It also made the connections able to share the same transmission performance.

Regarding the simpleperf, it is only operating using the TCP-protocol and is not implementing latency calculation or potential packets loss. However, the advantage with simpleperf is that it has built in network congestion control, and is retransmitting any lost packets or buffered packets.

One of the most important findings in this report, include the difference between expected bandwidth and link delay, versus the actual results. The expectations regarding this virtual network's topology are based on the best possible estimates, and do not include realistic challenges such as congestion and different sets of delay. The tools and theories presented in this report have been achievable using the given network topology. However, I still encountered some difficulties along the way, in form of iperf and Mininet bugging. The bugging resulted in me having to run specific tests several times, which was very time consuming. If I were to imagine this kind of implementation in a more extensive network, the process will be much more demanding and resourceful. Therefore, I recommend using more advanced tools when evaluating the network performance in a more complex network.

6. Sources

Literature:

ExtraHop. (n.d.). *Transmission Control Protocol (TCP)*. Retrieved April 2023, from ExtraHop: extrahop.com/resources/protocols/tcp/

Islam , S. (n.d.). *mininet-lab-intro.pdf*. Retrieved April 2023, from Canvas: OsloMet: https://oslomet.instructure.com/courses/25246/files/3124890?module_item_id=525586

Islam, S. (2023). *Performance Evaluation, Powerpoint*. Retrieved April 2023, from Canvas: OsloMet: https://oslomet.instructure.com/courses/25246/files/3053422?module_item_id=511765

Islam, S. (2023). *Transport layer*. Retrieved April 2023, from Canvas: OsloMet: https://oslomet.instructure.com/courses/25246/files/3140129?module_item_id=529682

Lee, Y. L., & Chuah, T. C. (2015). *Fairness Index*. Retrieved April 2023, from ScienceDirect: https://www.sciencedirect.com/topics/computer-science/fairness-index

Mininet. (n.d.). *Mininet Overview*. Retrieved April 2023, from Mininet: http://mininet.org/overview/

Pandey, P. (2022, 09 08). *Subnetting in Computer Networks*. Retrieved from Scaler: https://www.scaler.com/topics/computer-network/subnetting/

Roberts, M. (2022, 06 14). *iPerf: How to Test Network Bandwidth*. Retrieved April 2023, from Networkverge: https://networkverge.com/iperf-test-network-bandwidth/

Rouse, M. (2016, 11 11). *Network Bottleneck*. Retrieved April 2023, from Techopdeia: https://www.techopedia.com/definition/24819/network-bottleneck

Pictures:

OsloMet logo:
https://kommunikasjon.ntb.no/presserom/oslomet/mi?publisherId=15678779&item=logo-16984812

Network topology