PORTFOLIO 1

DATA2410

Teresa Pham

s345368 | Oslo Metropolitan University
April 17. 2023

TABLE OF CONTENTS

1 INTRODUCTION

Measuring network throughput has become crucial for ensuring efficient and reliable network performance. The key topic for this report is network testing and performance evaluation, including aspects of TCP connections, Python socket programming, network topology, latency and throughput, multiplexing, parallel connections, and the effects of congestion and packet routing on network performance.

The approach involves building on existing work on network throughput measurement and designing and implementing a simplified version of iPerf using sockets, called simpleperf. The objective is to gain a thorough understanding of the testing tools, the results obtained, and the factors that affect network performance. Through various tests using simpleperf, ping, and iPerf on a virtual network managed by Mininet on a virtual machine, we aim to understand the testing tools, the results, and the factors influencing network performance. However, the approach has certain limitations, such as the simplified nature of our tool and the fact that we are testing the network on a virtual environment. The outcome of this portfolio includes the implementation of simpleperf, the results obtained from testing the virtual network, and hopefully valuable perspectives on the evaluation of network performance and testing.

The remaining sections of the document are structured into four parts. Section 2 discusses the design and implementation of simpleperf, Section 3 describes the experimental setup, Section 4 presents the test results and analysis, and Section 5 concludes the paper while addressing the limitations and shortcomings of our approach.

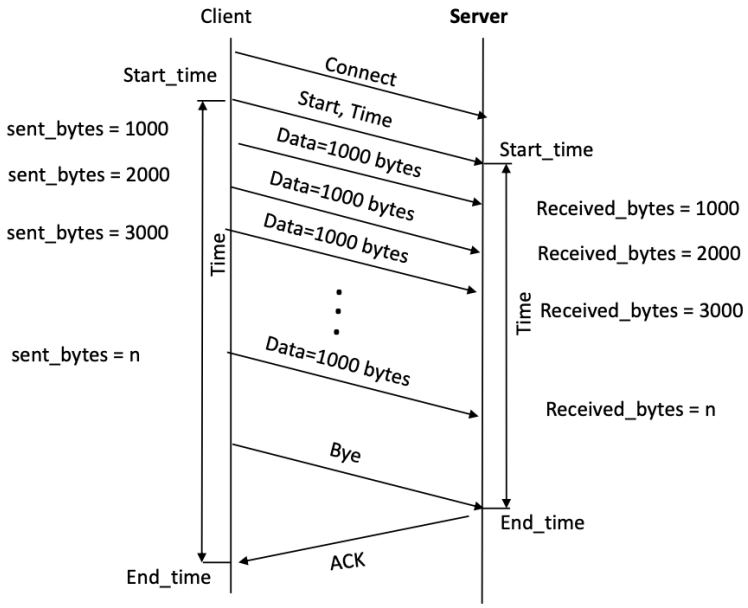
2 IMPLEMENTATION OF SIMPLEPERF

The simple network throughput measurement tool I have implemented, *simpleperf*, is written in Python and runs in two modes: *server* and *client* mode. Simpleperf is implemented with the help of Python's built-in *socket* module; to create a TCP network socket and to communicate between the client and server, *threading* module; to create multiple threads for parallel connections, and *time* module; to measure the time taken for data to be transferred between the client and server. Also, *argparse*, *ipadress* and *re* module to implement optional arguments so the user can invoke the server and client in a specific way. Lastly *sys* module, for exiting the program during exceptions.

The optional arguments for the servers let us select the IP address (`-b`), the port it should listen on (`-p`), and the format of the output data (`-f`). For the sake of simplicity, we assume `1 KB = 1000 Bytes, and 1 MB = 1000 KB`. While for the client let us select the IP address to connect to (`-I`), set the time (`-t`) and chose the format of the output data (`-f`). Additionally, the client can print statistics at specified intervals `(-i)`, set the number of bytes to transfer (`-n`) and chose the number of parallel connections using `(-P)`.

The server part consists of two functions. The first function, `server(…)`, starts up the server by setting up a TCP socket, bind it to the specified IP address and port number and listen for incoming client connections. When a client connection is accepted it will run the connection in a new thread with the second function. The second function, `handle_client(…)`, receives and reads data in chunks of 1000 bytes from the client, calculates the duration and the rate of the transfer, and sends an acknowledgement (“`ACK: BYE`”) back to the client before closing the connection. The rate is calculated by dividing the total number of received bytes by the duration of the transfer and multiplying by 8 and 1,000,000 to convert to megabits per second (Mbps). The received bytes are also converted to the user-selected format for output (`MB, KB or B`), before printing the statistics for each client connection, consisting of the client IP address and port number, the duration of the transfer, the total data received, and the transfer rate. The first server function, will keep running indefinitely to keep listening for incoming connections, until it is interrupted by a keyboard interrupt signal (CTRL+C), then the function gracefully closes the socket and exits the program. The program is also designed to handle exceptions, which ensures that if an error occurs, it will be handled appropriately.

The client part also consists of two functions. The first function, `client(…)`, starts the client by establishing a TCP connection with the server, and then sends data by running the second function in a new thread. The second function, `send_data(…)`,is the main function and is responsible for sending data to the server. It first checks the data format chosen by the user and sets a factor accordingly to convert the sent bytes to the chosen format for output (`MB, KB or B`). The code then enters a loop that sends data chunks of 1000 bytes until the specified transfer time or transfer size limit is reached. The loop sends the data chunk using the `sendall()` method of the socket module and adds the number of bytes sent to the total. The code also calculates and prints the statistics for the interval seconds if it is specified by the user. Once the transfer time or transfer size limit is reached, the loop breaks, and a BYE message is sent to indicate the end of the transfer. The code then waits for acknowledgement ("ACK: BYE") from the server before it closes the connection. Then the code calculates the duration of the transfer and the rate of the transfer in Mbps. Finally, it prints the statistics, including the server IP address and port number, the duration of the transfer, the total data sent, and the transfer rate. The calculation and printing of statistics are placed in a finally block to ensure that they are displayed even in the event of an error. Exceptions are handled, so if any error occurs, the code catches it and handles it accordingly.



CLIENT AND SERVER (ISLAM, 2023).

The figure visualizes the communication between the server and the client invoked in default mode. See the file `simpleperf.py` for full details.

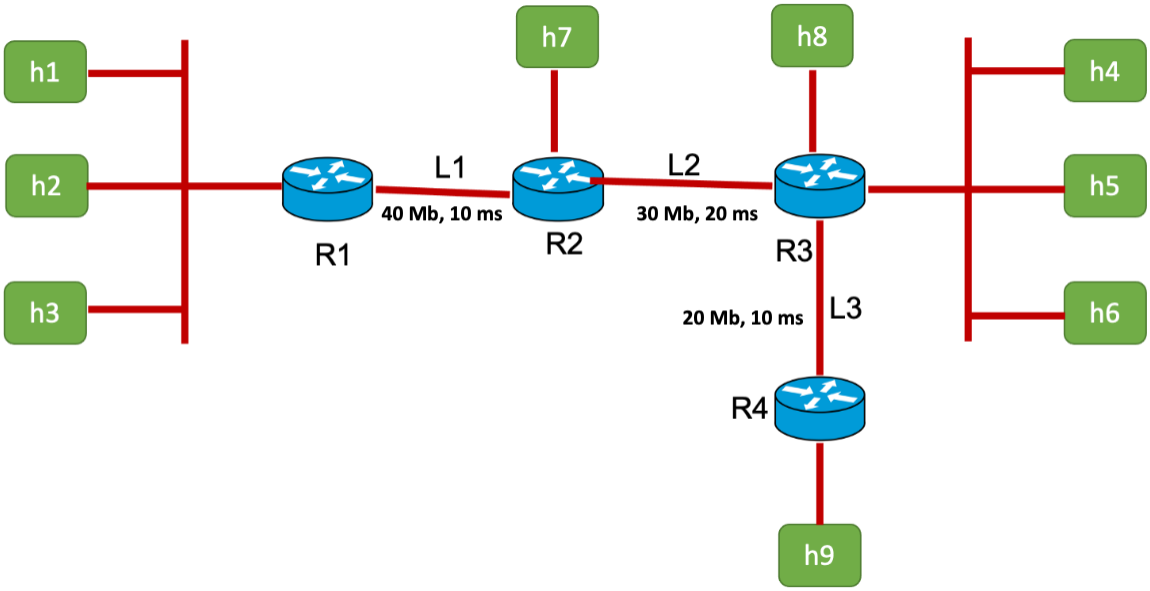
3 EXPERIMENTAL SETUP

Various of tests will be performed on a virtual network topology provided in the `portfolio_topology.py` file. The virtual network was set up using Mininet in a virtual machine, Debian 11 from UTM. Virtual machines are often used for testing because they provide an isolated environment that would not affect the host machine or other virtual machines (VMware, Inc., 2023).

The network topology consists of 9 hosts (h1 to h9), 4 routers (R1 to R4) and 2 switches (s1 to s2), and makes out 8 subnets labeled as A, B, C, D, E, F, G and H. Subnet A consists of three hosts (h1, h2, and h3) connected to a switch (s1), connected to a router (R1). Subnet B consists of two routers (R1 and R2) connected by a link L1, subnet C of a host (h7) connected to a router (R2), subnet D of two routers (R2 and R3) connected by link L2, subnet E has a single host (h8) connected to router (R3), subnet F consists of three hosts (h4, h5, and h6) connected to a switch (s2), connected to a router (R3). Subnet G consist of two routers (R3, R4) connected by link L3, subnet H has a single host (h9) connected to router (R4).

The links between the routers has set bandwidth, delay, and queue size that allows us to simulate different network conditions. The first link L1 (R1–R2), has a bandwidth of 40Mbps, a delay of 10ms, and a maximum queue size of 67 packets. The second link, L2 (R2–R3), has a bandwidth of 30Mbps, a delay of 20ms, and a maximum queue size of 100 packets. Lastly, the third link L3 (R3–R4), has a bandwidth of 20Mbps, a delay of 10ms, and a maximum queue size of 33 packets.

See the `portfolio_topology.py` file for the IP addresses and more details. The network should look like this:



THE NETWORK TOPOLOGY (ISLAM, 2023).

4 Performance evaluations

4.1 Network tools for performance evaluation

For performance evaluation I will mainly use the *simpleperf* tool and the standard latency measurement tool *ping* to measure the bandwidth and latency on the virtual network in Mininet. Ping calculates the *round-trip time (*RTT), which is the time it takes for data to travel from the client host to a target host and back again, by using Internet Control Message Protocol packets (ICMP). This is done by sending an ICMP echo request to the target host and waiting for an ICMP echo reply in response. (Kurose & Ross, 2022, p. 474). In addition, I will use the original tool used for active measurements of the maximum achievable bandwidth on IP networks, the original simpleperf, *iperf* (iPerf, 2023).

4.2 Performance metrics

The performance metrics for simpleperf include interval, transfer, and the bandwidth. *Interval* shows the transfer time, the duration of the entire transfer. *Transfer* shows the total number of data transferred for the interval in chosen format (default MB). The *bandwidth* is the maximum speed at which the data is transferred in Mbps, and is the most important metric here. High throughput, or bandwidth, is desirable and indicates that the network is capable of handling large amounts of data quickly and efficiently. While a low throughput indicates a bottleneck in the network that is limiting the data transfer rate. With this, we can monitor the rate of transfer to identify potential bottlenecks or other performance issues.

The performance metrics for iperf in UDP mode include *bandwidth*, measured in Mbps, *jitter,* variation in delay between packets, and *packet loss,* and percentage of packets that do not arrive at the destination. Equivalent to simpleperf, a high throughput or bandwidth and indicates that the network is running smoothly, while low throughput indicates that there are network issues affecting the transfer.

On the other hand, ping measures network latency or delay in the network. The performance metrics for ping includes the number of packets *transmitted*, *received*, the *percentage of packet loss*, and the *minimum*, *maximum*, and *average* RTT for the packets, measured in milliseconds (ms). The key metric is the average RTT. A low RTT is desirable and indicates that the network is fast and responsive, while a high RTT indicates a slow or congested network.

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

4.3.1 RESULTS

Host pair	*Bandwidth*
h1-h4	28.3 Mbits/sec
h1-h9	17.8 Mbits/sec
h7-h9	17.8 Mbits/sec

4.3.2 DISCUSSION

For this test case, I conducted three separate iperf tests in UDP mode using -b XM between the client-server pairs h1-h4, h1-h9, and h7-h9. Here, X represents the chosen rate for measuring the bandwidth across the different pairs. It is preferable to select a rate that does not exceed the capacity of any of the links in the path between the source and destination when testing network bandwidth. This is because the bandwidth of a network is limited by its slowest link, so if any of the links in the network is congested, it can create a bottleneck, slow down the entire network and drop packets, resulting in lower bandwidth than expected. Therefore, to measure the actual bandwidth accurately, the test rate should be less than or equal to the bottleneck link's capacity. I found the minimum bandwidth limit along the path of the network in each case to ensure efficient network functioning based on the network's topology.

From the network topology, we can see that the pair h1-h4 are in different subnets, and the traffic must traverse through links L1 (R1-R2) and L2 (R2-R3), that has set bandwidths of 40 Mbps and 30 Mbps respectively. To ensure reliable performance, I selected a rate of 27 Mbps, which is slightly below the slowest link speed of 30 Mbps. The client pair h1-h9 are also in different subnets, and the traffic needs to pass through the links L1, L2 and L3 (R3-R4), that has set bandwidth 20 Mbps. In this case, I chose 17 Mbps as the measurement rate, also a little less than the bottleneck bandwidth. Finally, the h7-h9 pair are also in different subnets, the traffic must pass through links L2 and L3, also having the same bottleneck bandwidth of 20 Mbps. For this pair, I also selected a rate of 17 Mbps.

The table above shows the bandwidth achieved during the tests, and we can see they are all slightly higher than than the specified bandwidth. Suggesting that the network links were not heavily congested during the test, which allowed for slightly higher than expected throughput. UDP is a connectionless protocol that lacks reliability, flow control, congestion control, and error recovery mechanisms (Kurose & Ross, 2022, p. 123). This can result in potentially higher data transmission rates compared to TCP traffic with the risk of packet loss. However, the raw measurement reveals no packet loss and a minimal rate of jitter, indicating that the network devices were able to handle the traffic efficiently. Therefore, it can be concluded that the network is performing well and can support the specified data transfer rate.

If I were asked to use iPerf in UDP mode to measure the bandwidth where I don't know anything about the network topology, I would run tests starting with a low rate, like 1 Mbps, and then gradually increase the packet size until packet loss or a decrease in throughput is observed. Based on these results, I would be able to estimate the maximum achievable bandwidth for that network topology. However, this approach may not be practical or reliable in all cases, especially if the network is complex or dynamic. Furthermore, UDP may not provide a complete picture of the network's behavior since it only measures the maximum achievable throughput under ideal conditions. UDP provides an unreliable data transfer service, and may affect the accuracy of bandwidth measurements due to packet loss and jitter (Kurose & Ross, 2022, p. 122). Moreover, the bandwidth can be influenced by various network conditions that may not be apparent without prior knowledge of the network topology. Therefore, it may be necessary to use other additional tools and methods to gain a better understanding of the network's characteristics, such as Wireshark.

4.4 TEST CASE 2: LINK LATENCY AND THROUGHPUT

4.4.1 RESULTS

Link	*Average RTT*	*Bandwidth*
L1	23.118	38.12 Mbps
L2	45.634	28.47 Mbps
L3	23.205	19.08 Mbps

4.4.2 Discussion

In this case I measured the RTT and bandwidth of each of the three individual links between the routers (L1, L2 and L3), running ping with 25 packets and simpleperf for 25 seconds. As mentioned, RTT is the time taken for a packet to travel from the client to the target and back again, and delay is one of the factors that affect the RTT of a network. The delay parameter in a network refers to the time it takes for a packet to traverse the link between two devices, including the processing time of the devices. As a result, the expected RTT for the links is at least twice the delay time set. While, the expected bandwidth should be close to the set bandwidth limit for the links. This is the theoretical maximum, the actual throughput may be lower due to network congestion, packet loss, and other factors.

The actual results of the links are very close to the expectations based on their set properties in the topology. Link L1 achieved an average RTT of 23.118 ms, which is close to the expected value of 20 ms, given its 10 ms delay. Additionally, the measured bandwidth of 38.12 Mbps is also very close to the set bandwidth of 40 Mbps. For L2, the measured average RTT of 45.634 ms is slightly higher than the expected value of 40 ms, given its 20 ms delay. However, the measured bandwidth of 28.47 Mbps is still close to the given bandwidth of 30 Mbps. Lastly, for L3, the measured average RTT of 23.205 ms is again close to the expected value of 20 ms, given its delay of 10 ms. While, the measured bandwidth of 19.08 Mbps is almost the same the given bandwidth of 20 Mbps.

In practical scenarios, it is typical to witness some differences between the expected and actual results due to several factors like network congestion, processing time of the devices, and other external factors. Nevertheless, the actual results indicate that the links are performing close to the expected RTT and bandwidth values, with only slight deviations. The deviation in the results is relatively small and is within an acceptable range. Therefore, we can conclude that the network is performing well.

4.5 Test case 3: Path latency and throughput

4.5.1 Results

Host pair	*Average RTT*	*Bandwidth*
h1-h4	64.995	28.21 Mbps
h7-h9	67.117	18.97 Mbps
h1-h9	87.996	18.84 Mbps

4.5.2 Discussion

Here, I measured the RTT and bandwidth of each of the three paths between the hosts h1-h4, h7-h9 and h1-h9, running ping with 25 packets and simpleperf for 25 seconds. In the previous test case, we only looked at the links, here we need to consider the entire path between the hosts. In this context, it is important to note that the expected RTT for the path is twice the sum of the delays for the path. Furthermore, the bandwidth results should be similar to the lowest set bandwidth limit, which is the bottleneck link. To calculate the expected latency and throughput for each pair of hosts, we need the path, which we already have from test case 1 and the network topology.

We know that for the pair h1-h4, the traffic must traverse through links L1 and L2. The expected latency between h1-h4 can be calculated by adding the delay of each links. The delay for L1 is 10ms and L2 is 20 ms, so the total delay is 30 ms, resulting in an expected RTT of at least 60 ms. The actual result of 64.995 ms is quite close to the expected results. L1 has bandwidth 40 Mbps and L2 has bandwidth 30 Mbps, which is the bottleneck link and the expected throughput between h1-h4. The actual result of 28.21 Mbps is also very close to the expected result.

In the same way, we can find the expected values between h7-h9. The traffic needs to pass through L2 and L3, with delays of 20 ms and 10 ms respectively, so the total delay is also 30 ms, resulting in an expected RTT also at least 60 ms. The actual result of 67.117 ms is slightly higher to the expected value. R3 has a

bandwidth of 20 Mbps, which is the bottleneck link, and the expected throughput. The actual result of 18.97 Mbps is very close to the expected results.

Finally, for h1-h9, the traffic must go through the links L1, L2 and L3, with delays of 10 ms, 20 ms and 10ms, summed up to the total delay 40 ms, resulting in an expected RTT of at least 80 ms. The actual result of 87.996 ms is slightly higher to the expected value. The bottleneck link here is also L3 with a bandwidth of 20 Mbps, which is the expected throughput. Here, again the bottleneck link is L3 with bandwidth 20 Mbps, which is the expected throughput. Again, the actual throughput result of 18.84 Mbps is very close to the expected results.

In summary, the measured results for both RTT and throughput closely matched the expected values for all three pairs, indicating proper network functionality. Although the RTT for h7-h9 was marginally higher than h1-h4, despite having the same expected latency, it still falls within an acceptable range, as network performance can be affected by various factors. Hence, we can conclude that the network is performing well and meeting its expected parameters.

4.6 Test case 4: Effects of multiplexing and latency

4.6.1 Results

Host pair	*Average RTT*	*Bandwidth*
h1-h4	101.944	14.37 Mbps
h2-h5	100.987	14.49 Mbps
h1-h4	101.872	9.85 Mbps
h2-h5	98.627	9.37 Mbps
h3-h6	103.045	9.94 Mbps
h1-h4	93.582	15.94 Mbps
h7-h9	91.829	12.68 Mbps
h1-h4	92.354	28.40 Mbps
h8-h9	40.277	19.10 Mbps

4.6.2 Discussion

In this experiment, I will look into the impact of multiplexing on network performance. Different pairs of hosts will simultaneously communicate using simpleperf and ping, while measuring latency and throughput using the same parameters as in previous tests (25 packets and 25 seconds). Since multiple hosts will be communicating at once, they will have to compete for network resources, potentially leading to congestion and increased latency.

Generally, the latency is expected to be higher and the measured throughput is likely to be lower compared to when only one pair is communicating. When both hosts start transmitting at the same time, their packets will compete for available bandwidth on each link, causing queuing delays and increasing the overall RTT. TCP has built-in congestion control mechanisms that prevent network congestion and ensure fair sharing of bandwidth among competing simultaneous connections. Thus, we can anticipate the maximum bandwidth for each host to be the bottleneck bandwidth divided by the number of pairs. Furthermore, the sum of the throughputs should be about equal the bottleneck bandwidth (Kurose & Ross, 2022, p. 306). While the expected RTT for a link is usually at least twice its delay time, in this scenario, we can anticipate considerably higher average RTT due to congestion and queuing delays. We can use *Jain's fairness index* (JFI) to evaluate the fairness of the bandwidth allocation between the host pairs. It is a value between 0 and 1, where 0 indicates complete unfairness and 1 indicates complete fairness, and is a desirable property in a network to ensure that all hosts are able to share the network resources fairly (Wikipedia, 2022). The Jain's fairness index can be calculated using the formula:

$$JFI = \frac{sum\ of\ throughputs^2}{number\ of\ pairs\ \cdot\ sum\ of\ squared\ throughputs}$$

We know that the first pair of hosts pairs, h1-h4 and h2-h5, the traffic goes through the same path L1, L2 and L3, so they have the same bottleneck bandwidth and total delay, which is 30 Mbps and 30 ms. When both the pairs start transmitting at the same time, we can expect them to transmit at a rate of 15 Mbps, half of the bottleneck bandwidth. The measured bandwidths of 14.37 Mbps and 14.49 Mbps for h1-h4 and h2-h5, respectively, confirm our expectations. Calculation of JFI also indicates that the bandwidth allocation between the two host pairs is quite fair, with a fairness index of 0.999983, which is close to 1. This means that both host pairs are receiving similar shares of the available bandwidth. The average RTT results of 101.944 ms and 100.987 ms are relatively high, indicating significant latency variation for each packet sent between the hosts.

In this test, we add in an extra pair, so three pairs of hosts, h1-h4, h2-h5, and h3-h6, communicate simultaneously through L1, L2, and L3, with the same bottleneck bandwidth of 30 Mbps and total delay of 30 ms. We can expect each host to transmit at a maximum rate of 10 Mbps, the bottleneck bandwidth divided by three, when transmitting simultaneously. The bandwidth results of 9.85 Mbps, 9.37 Mbps and 9.94 Mbps for h1-h4, h2-h5, and h3-h6 support our expectations. The JFI of 0.999338 also indicates that the bandwidth allocation between the three host pairs is relatively fair. The average RTT results of 101.872 ms, 98.627 and 103.045 for h1-h4, h2-h5, and h3-h6 respectively, is also relatively high. However, it is worth noting that even with an extra host pair, the RTT does not increase significantly. The reason for this could be that the link had enough bandwidth to handle the increased traffic, so the queuing delays may not have been significant enough to cause a substantial increase in the RTT.

The next host pairs to communicate simultaneously, h1-h4 and h7-h9, only shares one common link. The host pair h1-h4 goes through L1 and L2, while h7-h9 goes through L2 and L3, so they will have to compete for the available bandwidth on L2, which has bandwidth 30 Mbps. When both the pairs start transmitting at the same time, we can expect them to transmit at a rate of 15 Mbps, half of the bandwidth. The measured bandwidth results were in line with our expectations, with 15.94 Mbps and 12.68 Mbps for h1-h4 and h7-h9, respectively. The slight offset could be due to the delay of me pressing ENTER to run the programs. With a JFI of 0.987192 it is indicated that the bandwidth allocation between the two host pairs is moderately fair. Both pairs have coincidentally the same total of delay of 30 ms, as L2 has a delay of 20ms and both L1 and L3 have the same delay of 10 ms. As a result, the expected RTT would be around the same for both pairs. The average RTT results of 93.582 ms and 91.829 ms for h1-h4 and h7-h9, respectively, are slightly lower than the two previous tests, which had an average of around 100 ms. This could be due to the fact that this pair is only competing for available bandwidth on one link, whereas the previous tests had to compete for the whole path consisting of two links.

For the last two host pairs, h1-h4 and h8-h9, we can observe that although they both go through router R3, they do not share a single common link. Specifically, h1-h4 goes through links L1 and L2, while h8-h9 only uses L3, and thus, they do not have a link to compete for available bandwidth. Therefore, I do not expect the bandwidth to be affected by the simultaneous connection. The bottleneck bandwidth for h1-h4 is 30 Mbps, while for h8-h9 it is 20 Mbps, and the actual results of 28.40 Mbps and 19.19 Mbps, respectively, support our expectation. Calculation of JFI gives a fairness index of 0.963082, which is marginally lower than the previous results, but still suggests a relatively fair bandwidth allocation. Although the throughput was not significantly affected, there is traffic load on the links in the network when running ping and simpleperf simultaneously, which can affect the RTT. The total delay for the host pairs are different, hence we can expect different average RTT values. For h1-h4 the total delay is 30 ms, which we typically expect the RTT to be at least 60 ms, and for h8-h9, it is 10 ms, where we typically expect at least 20 ms. Looking at the results for h1-h4 of 92.354 ms and 40.277 ms for h8-h9, we can conclude that the latency is still relatively high due congestion of high levels of traffic.

In conclusion, the results confirm that when multiple pairs of hosts communicating simultaneously, the network congestion can significantly impact the latency and throughput, potentially leading to higher RTT and lower throughput as expected. The deviation between the expected and measured RTT is relatively

high, indicating potential variability in the network conditions or measurement errors. However, the deviation is still within a reasonable range, suggesting that the network is functioning properly. The bandwidth results were consistent with our expectations, and JFI shows that the bandwidth is distributed fairly between the host pairs, indicating good performance in terms of fairness. Ultimately, these results suggest that the network is operating as expected under these congested conditions.

4.7 TEST CASE 5: EFFECTS OF PARALLEL CONNECTIONS

4.7.1 RESULTS

Host pair	*Bandwidth*
h1-h4	8.76 Mbps
h1-h4	7.94 Mbps
h2-h5	6.51 Mbps
h3-h6	6.51 Mbps

4.7.2 DISCUSSION

In this experiment, I will measure the throughput when three pairs of hosts are communicating simultaneously using simpleperf, running for 25 seconds. The hosts h1, h2 and h3 are connected to R1 and will simultaneously talk to hosts (h4, h5 and h6) connected to R3. However, h1 will open two parallel connections to communicate with h4, while h2 and h3 will each open one regular connection.

Like the previous test, the hosts are communicating simultaneously and will all have to contend for network resources, potentially leading to congestion. The measured throughput is expected to be lower than the the second test in the previous case, where the pairs h1-h4, h2-h5 and h3-h6 communicating simultaneously. This is because there is an extra parallel connection, which can lead to higher contention for network resources. All the traffic will go through L1, L2, and L3, which have a bottleneck bandwidth of 30 Mbps and a total delay of 30 ms. Therefore, the expected bandwidth of each host can transmit is around a rate of 7.5 Mbps maximum, the bottleneck bandwidth divided by four. The actual performance is quite accurate, with the results of 8.76 Mbps, 7.94 Mbps, 6.51 Mbps and 6.51 Mbps for h1-h4, h1-h4, h2-h5 and h3-h6, respectively. The slight offset between the results is possibly be due to the delay of trying pressing ENTER to run the simpleperf at the same time. Furthermore, the calculation of JFI indicates that the bandwidth allocation between the two host pairs is quite fair, with a fairness index of 0.983425, meaning that all host pairs are receiving similar shares of the available bandwidth. On the basis of this, we can see that the parallel connection has a similar effect to having an additional host pair communicating. Therefore, when multiple parallel connections are used, it is important to consider the impact on the available bandwidth and use appropriate measures to allocate resources fairly among all connections to achieve optimal performance.

5 CONCLUSIONS

Overall, the implementation and usage of the simpleperf tool has provided plenty of valuable insights into network testing and performance evaluation. I have gained a deeper understanding of various aspects such as TCP connections, Python socket programming, and network testing tools like ping and iPerf. Through the various tests performed with simpleperf, ping, and iPerf, I was able to gain significant perspective into network performance, including link and path latency, throughput, and the effects of multiplexing and parallel connections. During the execution of test case 4 on Ubuntu from the UTM gallery, I faced some problems with RTTs. However, I was able to obtain the expected values when I switched to Debian. This experience highlighted the importance of comprehending the network topology's specifications and their impact on latency and throughput. Without understanding this, I could have reached to incorrect conclusions. Although the tests were limited to a relatively simple network topology, it has provided a solid basis for further exploration into network performance evaluation. The knowledge gained from this will be indispensable for future network testing and troubleshooting.

6 REFERENCES

iPerf. (2023, April). *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. Retrieved from iPerf: https://iperf.fr/

Islam, S. (2023). *Portfolio guidelines.* Retrieved from Canvas: https://oslomet.instructure.com/courses/25246/files/3153005?module_item_id=533441

Kurose, J. F., & Ross, K. W. (2022). *Computer networking: A top-down approach* (Vol. 8th). United Kingdom: Pearson Education.

Mininet Project Contributors. (2022). *Mininet Overview*. Retrieved from Mininet: http://mininet.org/overview/

Oslo Metropolitan University. (2023, April). *Lab: Introduction to Mininet.* Retrieved from Canvas: https://oslomet.instructure.com/courses/25246/files/3124890?module_item_id=525586

VMware, Inc. (2023). *Virtual machine*. Retrieved from vmware: https://www.vmware.com/topics/glossary/content/virtual-machine.html

Wikipedia. (2022, December 8). *Fairness measure*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Fairness_measure