Portfolio 1 - Simpleperf

Name: Markus Einan
Student ID: S315297

Table of contents

1. Introduction

This is an implementation of a network measurement tool called Simpleperf. The script is coded with Python, and is a simplified version of the network measurement tool called iperf, which is written in C. The implementation of the Simpleperf script went well, but I encountered some problems when I was running tests using Mininet.

I noticed a very high variability in latency when running the RTT tests, which affected the average RTT to a very high degree. It seemed to also affect the tests when I was running tests for throughput with Simpleperf. To make sure the problem was not related to my script, I ran some cross tests with iperf, and encountered the exact same problem. Therefore, I added a file containing the test in the root directory, called cross-test.txt. It was a 100 second test between h1 and h4. I concluded that there was a problem with Mininet, given that the results were inconsistent, even while running the ping tests inside Mininet.

One of my fellow students, with the same Windows PC, encountered similar issues, but he resolved them by running his tests on another student's Mac computer. I spent several hours setting up VBox, Ubuntu VM and Mininet on another Windows computer, but the problem persisted. My last option was then to run the tests with the problems in mind, and try to describe what is happening as precisely as possible. I figured that, for me, it might be a better option to look closer at the minimal latency, opposed to the average, because it gave a more reliable result. The latency was often much greater than expected, but never less than expected. I will therefore include the minimal values of the latency tests, as well as the average values, even though it is not asked for in the task.

There are several ways the large fluctuations in latency can affect the throughput tests when I am running the script. It is not obvious to me what is causing the issue in this specific case, but here are some ideas of what could be happening:

- Bandwidth delay-product: This is the product of the link bandwidth and the round trip time latency. Higher latency increases the BPD, which means that more data needs to be in transit to fully utilize the bandwidth. If the sender’s og receiver’s buffer can not handle the increased amount of in-flight data, it may affect the throughput.
- TCP slow start and congestion control: The use of gradually increasing data sending rate to reduce the amount of congestion. In high latency networks, it takes longer for the sender to receive acknowledgements, which can result in lower throughput.
- Impact of packet loss: TCP will reduce its sending rate, when detecting packet loss, which may affect throughput.
- Flow control mechanisms: TCP adapts its transmission rate, based on other network conditions.

After doing some further testing with the throughput, I have noticed that, at best, the results are as expected, and at worst, they are very far from expected. This means that the results sometimes are close to what can be expected, which makes it more possible for me to draw conclusions based on the results. To help me identify potential issues, I have therefore chosen to run every test with 1 second intervals on the client side. That will make potential drops in performance visible to me, and make it easier for me to analyze the results. I sometimes ran multiple tests, until the rate came through as expected, but with the parallel tests in the later test cases, this was not as viable. That is because the probability of having issues is doubled or tripled, depending on how many connections I am running at the same time. Some of the results I have documented will therefore be off, but if I think they are, I will comment on them in the discussion section for each respective test case.

2. Implementation of Simpleperf

As I mentioned in the introduction, the Simpleperf tool is written in Python. The script includes a server side, and a client side mode, as well as functionality to take input from the user.

Server mode:
To run a server with Simpleperf you can type “python3 <path to file>Simpleperf.py -s” in the terminal window. If running with just the -s option, as shown in the example, it will start with the default options declared in the script.

Otherwise, the options available if running server mode are:

- Bind: Binds the server to the specified option (-b or –bind). The default value for the IP address is: ‘10.0.0.2’.
- Port: Sets the server's port number to the port specified by the user (-p or –port). The default value for the port is: 8088.
- Format: Sets the desired output format for displaying the received data (‘B’, ‘KB’ or ‘MB’), using (-f or –format). The default value for the format is: ‘MB’.

When the script is started, the main function calls on either server mode or client mode, depending on what the user has specified. If the script is started in server mode, the server_mode() function is called. The server_mode() function waits for incoming client connections, and if a client connects, it calls on the handle_client() function. The handle_client() function creates a number of parallel processes, depending on how many clients are connecting at the same time. It receives data from the client, and when the ‘BYE’ message is received from the client, it sends back an ‘ACK: BYE’ message to inform the client that it has closed the connection. After the connection to the client is closed, the function calls on the print functions, to print out the result of the connection.

Client mode:
To connect a client to an active server with Simpleperf, you can type “python3 <path to file>Simpleperf.py -c -I <server-ip> -p <server-port>” in the terminal window. This example runs a client connection in default mode, with the default values.

Otherwise, the options available if running is client mode is:

- Time: Specifies the duration of the connection to the client (-t or –time). The default duration is 25 seconds.
- Interval: Specifies a rate at which intervals of the data transfer process will be printed out to the user, in seconds (-i or –interval). The default interval rate is 0 seconds.
- Parallel: Specifies the number of parallel clients connecting to the server (-p or –parallel). It must be between 1 and 5 parallel connections, the default option is 1.
- Num: The total amount of data to send, specified with a unit (‘B’, ‘KB’ or ‘MB’), using (-n or –num). This option is only used if specified, and the default option for num is: None.
- Format: Sets the desired output format for displaying the received data (‘B’, ‘KB’ or ‘MB’), using (-f or –format). The default value for the format is: ‘MB’.

If the script is started in client mode, the main function makes a call to the client_mode function. Which in turn makes a call to the create_parallel_connections() function. The create_parallel_connections() function creates a number of parallel connections, depending on how many are specified by the user. Then it creates connections to the server, and starts to send data to the server, for the duration of the connection, or just the number of data if specified. It also creates a list to store the results of the connections, to be able to print the results in a structured

manner, after the connections are closed. The reason I have used the multiprocessing module instead of threading in this case, is because threading in Python does not result in true parallelization. This is because of the Global Interpreter Lock (GIL), in Python, which ensures that only one thread executes Python code at a time.

For both the client and server mode options, the arguments are not positional, which means that the position of the arguments does not matter. For a more detailed description of what happens in the script, please read the documentation and comments provided.

3. Experimental setup

The setup I will be using for these network measurement experiments, includes a Python file called “portfolio-topology.py”, it has 9 hosts, 4 routers and 2 switches.

The topology setup is as follows:

- The hosts h1, h2 and h3 are connected to router r1, via the switch s1.
- r1 is in turn directly connected to r2, via the L1 link between them.
- r2 is directly connected to r3, via the L2 link between them.
- r3 is connected to h4, h5 and h6 via the s2 switch, and also has a direct connection to h8, as well as a connection r4, via the L3 link.
- r4 is directly connected to host h9.

There has also been specified some limitations on certain subsets which might be worth noting:
Subnet B (link between r1 and r2):

- Bandwidth: 40 Mbps
- Delay: 10 ms
- Maximum queue size: 67
- Hierarchical bucket token: True

Subnet D (link between r2 and r3):

- Bandwidth: 30 Mbps
- Delay: 20 ms
- Maximum queue size: 100
- Hierarchical bucket token: True

Subnet G (link between r3 and r4):

- Bandwidth: 20 Mbps
- Delay: 10 ms
- Maximum queue size: 33
- Hierarchical bucket token: True

4. Performance evaluations

4.1 Network tools for performance evaluation

The tools I will be using to measure the internet performance are:

Mininet:
A network emulation platform, which allows you to create, test and experiment with different network topologies. It supports OpenFlow, a software defined networking protocol, which allows you to implement flexible and programmable network control planes using custom controllers.

iperf:
This is an open-source network testing tool, used to measure bandwidth and the quality of network links. It can also help with troubleshooting network problems, identifying bottlenecks, or evaluating the performance of different network configurations or protocols.

Simpleperf:
This is also a network testing tool, similar to iperf, and is described in greater detail under the “Implementation of Simpleperf” section.

Xterm:
A terminal emulator for the X Windows system. In this case, I am using it to run terminals for each respective node that I want to run tests on.

Open vSwitch:
OVS is a multilayer virtual switch designed to enable massive network automation. In this case, I will be using the lightweight testcontroller, since we are running tests on a smaller network topology.

4.2 Performance metrics

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

4.3.1 Results

Throughput h1-h4:

Interval	Transfer	Bandwidth	Jitter	Lost/Total Datagrams
0.0000-10.0203 sec	20.8 MBytes	17.4 Mbits/sec	0.214 ms	11937/26753 (45%)

Throughput h1-h9:

Interval	Transfer	Bandwidth	Jitter	Lost/Total Datagrams
0.0000-10.0291 sec	10.6 MBytes	8.90 Mbits/sec	0.242 ms	19167/26754 (72%)

Throughput h7-h9:

Interval	Transfer	Bandwidth	Jitter	Lost/Total Datagrams
0.0000-10.2275 sec	8.20 MBytes	6.72 Mbits/sec	4.051 ms	20891/26739 (78%)

4.3.2 Discussion

I chose the rate (X) to be 30 Mbps as the bandwidth option, this is because there is a bottleneck of 30 Mbps in between r2 and r3, which all the routes need to go through. After running the tests, I

saw that the bandwidth I chose may have been too high for the test done in this topology. That is based on the high package loss percentage, and jitter values in the report. We also have to take into consideration the high variability in latency I am experiencing with my Mininet setup, which might also have affected the results in different ways.

In general, a good way to measure the bandwidth of a topology you don't know anything about would be:

1. Start with a low target bandwidth: Begin by specifying a low target bandwidth (e.g., 1 Mbps). This ensures that the initial tests will not saturate the network and helps in understanding the network's behavior at lower bandwidths.
2. Gradually increase the target bandwidth: Increase the target bandwidth incrementally (e.g., by 1 Mbps or 5 Mbps) in subsequent tests. Observe the server-side reported bandwidth, packet loss, and jitter in each test.
3. Identify the saturation point: Continue increasing the target bandwidth until you notice a significant increase in packet loss, a decrease in the server-side reported bandwidth, or high jitter values. This point indicates that the network is becoming saturated, and it's likely that you have approached the network's maximum available bandwidth.
4. Fine-tune the target bandwidth: You can further refine your estimate by conducting additional tests around the saturation point. This will help you find the optimal balance between the target bandwidth, packet loss, and jitter.

While this approach can give you a reasonable estimate of the network's bandwidth, it has some limitations:

- Time-consuming: Gradually increasing the target bandwidth and running multiple tests can be time-consuming, especially if the network has a high available bandwidth.
- Potential network disruption: Running tests at or near the saturation point may temporarily disrupt the network for other users and applications.
- Network conditions: The results may vary depending on the current network conditions, including congestion, latency, and other ongoing network activities. It's essential to consider these factors when interpreting the results.
- UDP limitations: iPerf's UDP mode doesn't account for congestion control or reliable delivery mechanisms present in TCP, which may result in different performance characteristics compared to real-world applications using TCP.

4.4 Test case 2: link latency and throughput

4.4.1 Results

Latency L1: 20.8 (min), 30.8 (avg)
Latency L2: 40.6 (min), 52,7 (avg)
Latency L3: 20.4 (min), 24.9 (avg)

Throughput L1: 36.9 (simpleperf), 23.5 (iperf)
Throughput L2: 28.3 (simpleperf), 15.3 (iperf)
Throughput L2: 19 (simpleperf), 11.3 (iperf)

4.4.2 Discussion

To demonstrate the throughput measurement issues that I explained in the introduction further, I have included cross tests with iperf in all throughput tests done in test case 2. I will not do cross testing for the later test cases, for the sake of time, and the increasing complexity of the test cases. As you can see in the results, I am sometimes lucky, and are not experiencing latency issues that are affecting the throughput as much.

Expected latency:
L1: Given the 10 ms delay, the RTT would be 10 * 2 = 20 ms
L2: Given the 20 ms delay, the RTT would be 20 * 2 = 40 ms
L3: Given the 10 ms delay, the RTT would be 10 * 2 = 20 ms

Evaluation of latency:
As I said, in this case, the minimum values give me a better indication of the actual latency, since the jumps in variation ruin the average. We know that the delay for L2 is 20 seconds, whereas for L1 and L3 it is 10 seconds. That means that it is to be expected that the latency for L2 is double that of L1 and L3. If we look at the minimum values in the results, this corresponds well to the RTT results I got from running the ping tests.

Evaluation of throughput:
I ran several tests with simpleperf for each connection to try to get as close as possible to a realistic throughput for each connection. The average was sometimes down to 5 Mbps for each connection. If you look at the intervals of the cross tests with the iperf tests (which I only ran one time per connection to demonstrate), you can see that the values sometimes are close to, or at the expected rates for the respective connection.

4.5 Test case 3: path latency and throughput

4.5.1 Results

Latency h1-h4: 61.4 (min), 153.4 (avg)
Latency h1-h9: 85.7 (min), 177.6 (avg)
Latency h7-h9: 60.6 (min), 85 (avg)

Throughput h1-h4: 27.6
Throughput h1-h9: 18.9
Throughput h7-h9: 2.6

4.5.2 Discussion

Expected latency:

- h1-h4: The route for this connection will be h1 > s1 > r1 > r2 > r3 > s2 > h4, and back, which means that it crosses L1 and L2, with respective delays of 10 ms and 20 ms. That means that the total OWL will be approximately 30 ms and the RTT will be 30 * 2 = 60 ms. There will also be a small negligible delay between the hosts and their closest routers, which I will not include in the calculations.

- h1-h9: The route for this connection will be h1 > s1 > r1 > r2 > r3 > r4 > h9, and back. It crosses L1, L2 and L3, with respective latencies of 10, 20 and 10 ms. This adds up to a 40 ms OWL, and gives an expected RTT of 80 ms.

- h7-h9: The route for this connection will be h7 > r2 > r3 > r4 > h9, and back. It crosses L2 and L3, with respective latencies of 20 and 10 ms. This adds up to a 30 ms OWL, and multiplied by two, an expected RTT of 60 ms.

Expected throughput:

- h1-h4: The route for this connection will be h1 > s1 > r1 > r2 > r3 > s2 > h4, and back, which means that it crosses L1 and L2, with respective bandwidths of 40 and 30 Mbps. Since we are using the TCP protocol, the congestion control mechanism makes the throughput close to the largest bottleneck in the network. In this case that is the 30 Mbps bottleneck between r2 and r3. Which means that the expected throughput rate should be around 30 Mbps.

- h1-h9: The route for this connection will be h1 > s1 > r1 > r2 > r3 > r4 > h9, and back. It crosses L1, L2 and L3, with respective bandwidths of 40, 30 and 20 Mbps. Given the explanation above, the largest bottleneck here will be the 20 Mbps in L3. Which gives us an expected rate of around 20 Mbps.

- h7-h9: The route for this connection will be h7 > r2 > r3 > r4 > h9, and back. It crosses L2 and L3, with respective bandwidths of 30 and 20 Mbps. That means the expected rate for this connection will be the same as the previously discussed, around 20 Mbps.

Evaluation of latency:
The average values of the latency results got destroyed by the variability in latency. But if we look at the minimal values, we can see that the results are corresponding well to the expected latencies I have calculated. The latency is never less than the expected values, but sometimes a little bit higher, which is to be expected, given the small delays in transmission between host, switches and routers. It is also to be expected that these small delays add up more over longer routes, as is clear in the h1-h9 connection, which is the longest route of the three.

Evaluation of throughput:
The throughput values of the test are also corresponding well to the expected results. The exception is the measurement between h7 and h9, where we would expect a throughput of around 20 Mbps. But given the issues I have been facing with Mininet, this was a problem I was expecting when running tests. I could maybe have run more tests and hoped for a luckier connection period, but in this case I will just comment that I think the results of the h7-h9 connection is way off the expected result. If we look at the other two, we can see that the bandwidth is corresponding quite well to the expected, with a 27.6 for the expected 30 on h1-h4, and 18.9 for the expected 20 og h1-h9.

4.6 Test case 4: effects of multiplexing and latency

4.6.1 Results

Test 1:
Latency h1-h4: 60.5 (min), 108.7 (avg)
Latency h2-h5: 60.7 (min), 80.4 (avg)

Throughput h1-h4: 9.5
Throughput h2-h5: 18

Test 2:
Latency h1-h4: 63.2 (min), 127.1 (avg)
Latency h2-h5: 61.2 (min), 115 (avg)
Latency h3-h6: 62 (min), 119 (avg)

Throughput h1-h4: 12.3
Throughput h2-h5: 11.3
Throughput h3-h6: 4.4

Test 3:
Latency h1-h4: 61.6 (min), 123 (avg)
Latency h7-h9: 61.5 (min), 100 (avg)

Throughput h1-h4: 10.9
Throughput h7-h9: 5.8

Test 4:
Latency h1-h4: 60.8 (min), 85 (avg)
Latency h9-h9: 20.2 (min), 26 (avg)

Throughput h1-h4: 26.5
Throughput h8-h9: 18

4.6.2 Discussion

Evaluation of latency:

- Test 1: The only thing I have to base my assumptions on is the minimal value, which in this case gives me a much harder job to look for interference between the two connections running at the same time. But even if the tests were normal, I would not expect to see a great deal of interference between the two connections. That is because the traffic load of a ping test is very low, so even though the two connections share the same route, I would assume the interference to be low.

- Test 2: Again, in this case, my data to evaluate a potential interference is quite low. But in the minimal results, we can see a slight increase in latency, compared to the parallel test in test 1. That might mean that the traffic load of 3 parallel connections running ping is sufficient to create an increase in latency. But it could also be random.

- Test 3: When running two parallel connections from h1-h4 and h7-9, they are sharing a link in their routes, between r2 and r3. This could lead to an interference, but it is hard to evaluate with my results.

- Test 4: In this case, the routes from h1-h4 and h8-h9 do not have any shared links. But they both have to go through the router (r3). That means that the potential interference between the two connections, depends on the capacity of the router to forward packets. In this case, I would assume the interference to be low, given the low traffic load of ping tests.

Evaluation of throughput:

- Test 1: Given that the TCP protocol uses various congestion control algorithms to share network resources fairly between multiple connections, I would expect the results in this case to be more even. My assumption based on the results I have gotten is this: It looks like the transmission rate of h1-h4 was lower than expected, which might have freed up some space for the h2-h5 connection, given that it was allocated almost double the bandwidth. Since they are both simultaneously going through the same 30 Mbps bottleneck in L2, I would have expected around 14 Mbps per connection, which would have been 14 * 2 = 28 Mbps in total. But even though my throughputs were uneven, they add up to 18 + 9.5 = 27.5 Mbps. This means that I was probably lucky with my connection timing, and ended up utilizing close to the bandwidth bottleneck of 30 Mbps.

- Test 2: The case for test 2 looks kind of similar to the prognose for test 1. It looks like the h3-h6 connection's transmission rate was lower than it should be, which freed up space for the other two connections. Normally I would have expected a fair distribution between the 3 connections, something like 3 * 9.5 = 28.5. In my case the throughputs add up to 12.3 + 11.3 + 4.4 = 28, which again means that I have been lucky with my connection timing.

- Test 3: For the third time, the connections are sharing a bottleneck of 30 Mbps in link 2. I would have expected an approximate 14.5/14.5 throughput between the connections. My throughputs add up to 10.9 + 5.8 = 16.7, which is quite far away from the desired 28-29 Mbps available bandwidth in link 2.

- Test 4: In this case, the connection’s do not have any shared links, and therefore, no shared bottleneck. But since they go through the same router (r3), the potential interference depends on the router's capacity to forward packets at a high enough rate. It looks to me like I was lucky with the connection timing again this time. Because both results were close to the bottleneck of their routes. 26.5/30 Mbps for h1-4, and 18/20 Mbps for h8-h9.

4.7 Test case 5: effects of parallel connections

4.7.1 Results

Throughput h1-h4: 5.2 (1), 6.4 (2)
Throughput h2-h5: 7.2
Throughput h3-h6: 7.9

4.7.2 Discussion

Looking at the results, it seems I have been lucky with my connection timing again. In my opinion the results look quite realistic. This is because all the connections go through the same bottleneck in L2, which is 30 Mbps. Which means that the transmission rate at any given time, should not exceed 30 Mbps. If we calculate the sum of all the throughput rates from my connections we get: 5.2 + 6.4 + 7.2 + 7.9 = 26,7. This is pretty close to the 30 Mbps bottleneck.

5. Conclusions

Test case 1:
There could have been several factors affecting my results when measuring bandwidth in UDP mode. For one, I could not rely on Mininet to give consistent reliable latency, and on the other hand, 30 Mbps might have been a little high transmission rate. That is based on the high percentage package loss in the report. What I learned from the tests, is that the UDP mode is a much less reliable option than TCP mode, because of its lack of congestion control or reliable delivery mechanisms.

Test case 2:
I managed to get results that corresponded well to the expected results that I calculated. And in this particular case, the minimal value for RTT testing was a decent indicator, as opposed to the average. I was also quite lucky to be able to run tests that were close to the expected bandwidth in Mbps.

Test case 3:
I also managed to get decent results on the tests in this case, except from the throughput test between h7 and h9. For the latency, my tests corresponded well in comparison to the rates I calculated. The throughput rates also corresponded well to the expected rates, based on the bottlenecks.

Test case 4:
The latency did not seem affected to a high degree by the parallel connections, the exception might have been in test 2, running 3 parallel connections, but it is hard to conclude with just the minimal values.
I was able to demonstrate the relationship between the bandwidth of a given link, and parallel connections going through it. On test 1, 2 and 4, the results were close to the expected bottleneck values. The exception was test 3 where the throughputs were probably off. Even though my results may be off, there is not always the case that it is a fair distribution between connections. This could

be because of various factors like, congestion control algorithms, RTT, jitter, packet loss or TCP slow start.

Test case 5:
Since I ran two parallel connections from one terminal, and two other parallel connections from two other terminals, this added up to 2 parallel connections, and multiplexing between 3 connections. They all went through the same bottleneck, and the correlation between the expected throughput and the sum of my connections was high. We could even see that the two multiplexed connections without parallelization were allocated less resources than the one running parallel connections, given that the rates were reliable.

In conclusion, I experienced some difficulties because of my struggle with Mininet. But I got a lot of reasonable test results, which meant that I was able to see the relationship between bandwidth and throughput, and delays and latency. I learned how to estimate approximate expectations for connections on a given route, and use the simpleperf tool to verify my assumptions. I also learned how multiplexing and parallel connections affect each other's throughput rates, if going through the same route.

6. References

Mininet:
http://Mininet.org/vm-setup-notes/
iperf:
https://iperf.fr/iperf-doc.php