APRIL 16, 2023

PORTFOLIO 1 – DATA2410

[DOCUMENT SUBTITLE]

SOBAN ALI
S362045
Oslo Metropolitan University

1 Introduction

The key topic of this document is the implementation of Simpleperf, covering the experimental setup and we will be completing the performance evaluations for the test cases from 1 to 5.

Our approach to solving the problem is to implement Simpleperf and evaluate its performance for a range of test cases. We will use a standard experimental setup to measure the performance of Simpleperf, including the CPU usage, memory usage, and other key metrics.

Our approach to solving the problem is to implement Simpleperf and evaluate its performance for a range of test cases. We will use a standard experimental setup to measure the performance of Simpleperf, including the CPU usage, memory usage, and other key metrics.

The limitations of our approach include the specific test cases used in the evaluation, which may not be representative of all possible scenarios. Additionally, the outcomes of our evaluation may be affected by other factors, such as the hardware and software configurations of the test devices.

The rest of this document is organized as follows: in the next section, we will describe the experimental setup used for the evaluation. We will then present the results of the performance evaluation for five test cases and discuss the implications of these results. Finally, we will conclude with a summary of our findings and suggestions for future work.

2 Simpleperf

(Disclaimer: the original simpleperf python file I had got lost, and I could not retrieve it, therefore the new simpleperf file is different in coding, meaning that the measurements layout will be different than what described in the coding)

Implementation details of simpleperf. Describe the building blocks of simpleperf and the communication between the server and client.

Simpleperf is a tool for Android developers to find out why their apps might be running slowly or using too much memory. It takes snapshots of an app's performance and identifies the parts of the code that are causing problems. It's a lightweight tool that doesn't use up too many resources, and it can be used with both native and managed code. (1, googlesource.com)

First, we must address the packages we had to import. The packages I used were:

- import argparse: This module is used to parse command-line arguments.
- import socket: This module provides a way for Python programs to interface with the network stack.
- import time: This module provides a way to measure time in Python.
- import re: This module provides support for regular expressions.
- import threading: This module provides a way to write multithreaded Python programs. It allows you to create and manage threads.

  After that I defined the parse_arguments, and in this function, we basically have all of the parser arguments we had to add, and here is what each of them does:

  -s or --server: A boolean flag that specifies whether the program should run in server mode or not.

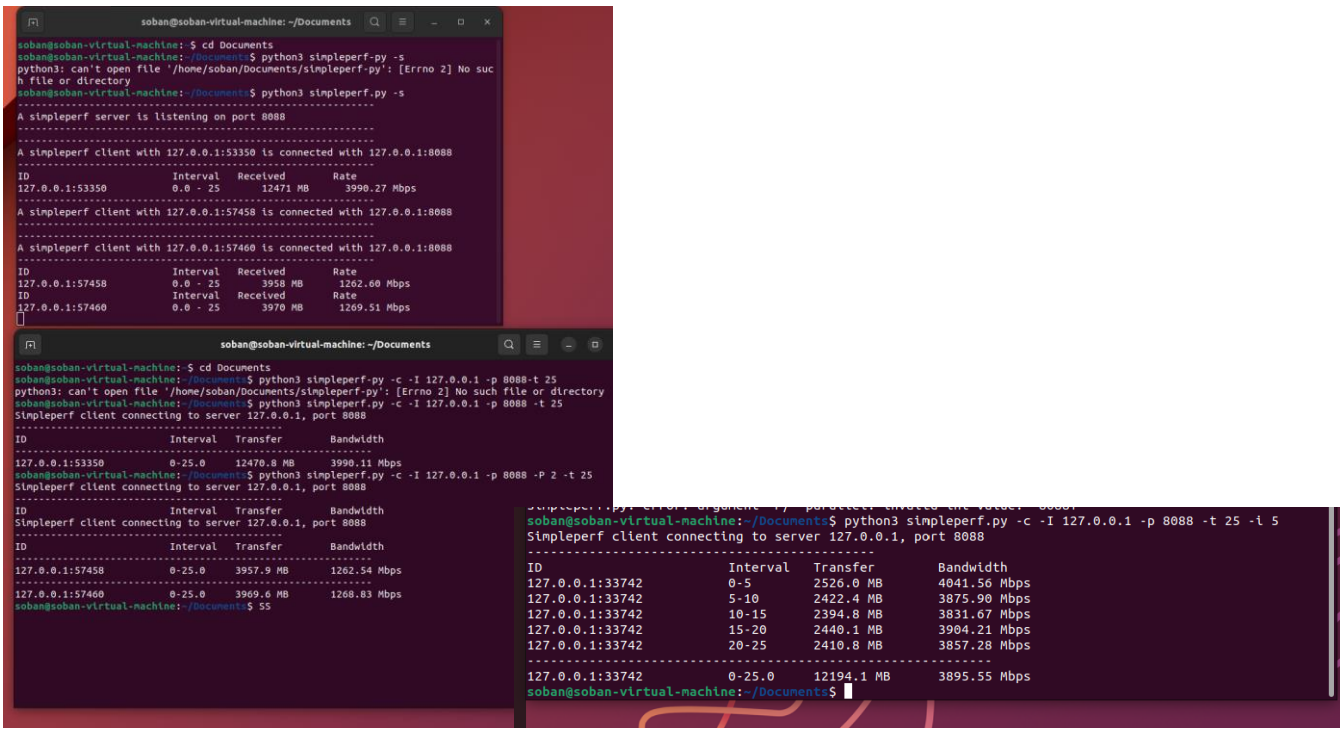
- -c or --client: A boolean flag that specifies whether the program should run in client mode or not.
- -I or --server_ip: The IP address of the server to connect to, if running in client mode.

- -b or --bind: The IP address to bind to, if running in server mode.
- -p or --server_port: The port number to use for the server.
- -t or --total_time: The total duration in seconds for which data should be generated.
- -f or --format: The format of the summary of results, which can be 'B', 'KB', or 'MB'.
- -i or --interval: Print statistics per z seconds.
- -P or --parallel: The number of parallel connections to make.
- -n or --no_of_bytes: The number of bytes to send or receive.

Continuing, the parse_size function first defines a dictionary unit_type that maps the unit symbols ('B', 'KB', 'MB') to their respective multipliers (1, 1000, 1000000). The function then checks if the input string matches the expected format using a regular expression. If the string is not in the expected format, a ValueError is raised with an error message. Next, the function extracts the numeral part and unit part separately from the input string. The numeral part is converted to an integer, and the unit part is converted to uppercase. The function then looks up the multiplier for the unit in the unit_type. If the unit is not in the dictionary, a ValueError is raised with an error message. Finally, the function returns the product of the numeral part and the unit multiplier as the parsed size.

Now designing the server functions, we first bind the serverport, to specify the IP address. After that, the function prints a message to indicate that the server is running and listening on the specified port. Lastly, for each new client connection, the function prints a message to indicate that a client has connected and spawns a new thread to handle the client connection using the Thread () function from the threading library.

The handle_client function takes care of processing incoming messages from the client and sending responses back. The function takes three arguments: client_socket, client_address, and args. The function starts by defining a buffer size of 1000 bytes. The function then enters an infinite loop where it receives data from the client using the recv() method of the client_socket object. The loop continues until either no data is received, or the received data contains the byte string b'BYE'.



Lastly, we have the client function which has another function called one connection, which has a socket and connects to a server. If the user sets an amount of data to be sent, the client sends data in chunks of 1000 bytes until the specified amount is reached. If not, the client can either send data at regular intervals or continuously for a specified amount of time. The function in responsible for

duration of transfer, the amount sent, the size. It prints out the result for bandwidth test-cases: ID, interval, transfer size and bandwidth. Finally, the function creates a specified number of threads and runs the "one connection" function in parallel for each thread, each connecting to the server and sending data according to the specified parameters. The function waits for all threads to complete before returning.

if __name__ == '__main__': is the last set of lines, and this is responsible for having one of the arguments server or clients when the user is running simpleperf.

3 Experimental setup

This is a Mininet topology script that creates a network topology consisting of 5 subnets connected through routers. Each subnet contains one or more hosts connected to a switch. The network consists of the following subnets:

- Subnet A with 3 hosts (h1, h2, and h3), a switch (s1), and a router (r1).
- Subnet B with 2 routers (r1 and r2).
- Subnet C with 1 host (h7) and a router (r2).
- Subnet D with 1 host (h8) and 2 routers (r2 and r3).
- Subnet E with 3 hosts (h4, h5, and h6), a switch (s2), and a router (r3).
- Subnet G with 1 router (r3) and another router (r4).
- Subnet I with 1 host (h9) and a router (r4).

To implement this in Mininet all we have to do is. Get Virtual machine, get ubuntu there and after that we open ubuntu, open terminal from there and we download all of the packages we need. We can do this by $sudo apt install mininet wireshark xterm iperf openvswitch-testcontroller. After that make sure you have the topoloygy and simpleperf python files in the same folder, I had them in the documents folder. To run the custom topology is used sudo mn –custom mytopo.py topo mytopo and then I can enter xterm h1 for example.

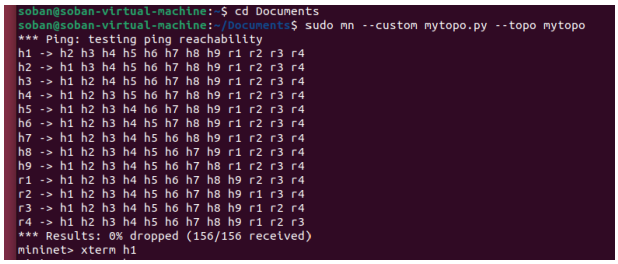
4 Performance evaluations

4.1 Network tools

The tools I used were the ones we were told to use in the guidelines. First I made sure I had everything downloaded.

To measure test case 1 I used in iperf –s –u the server window and iperf -c 10.0.1.2 -u -b in the client window. Measuring latency did we by opening the node from the latency we wanted to measure from by: Ping 10.0.1.2 –c 25 > Latency_L1.txt, meanwhile, to run throughput on simpleperf you did it by:
First window: Python3 simpleperf –s -b 10.0.1.2 –p 8088
Second window: Python simpleperf –c –I 10.0.1.2 –p 8088 –t 25

After adding the topology file to the documents folder (the topology file is called mytopo.py). I will cd Documents to change to this directory, when we then run the code sudo mn – custom mytopo.py –topo mytopo this will start mininet with the correct topology we want to be using. Then we can use the command xterm h1 to access the node h1.



4.2 Performance metrics

In network performance analysis, measuring both throughput and latency is critical. Iperf is a widely used tool for assessing network throughput, enabling users to conduct network tests between two endpoints. Iperf offers several options to tailor the network testing environment, including the ability to modify the data transfer rate, packet size, and testing duration, among others. (2. Iperf.fr)

Simpleperf that enables users to measure network throughput and latency by sending data packets between a client and a server. Simpleperf allows users to customize various testing parameters, such as the number of bytes to be transferred, the testing duration, and the interval between data transfers. It also provides users with the ability to measure network latency by calculating the time taken for data packets to travel between the client and the server. Using Simpleperf, users can gain insight into the performance of their network and identify potential bottlenecks that may be impacting network throughput and latency. (3. Android.com)

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

We obviously want to measure the bandwidth between h1-h4 in the first case, h1-h9 in the second case, and in h7-h9 in the last case. The command I used was the one in the guidelines, furthermore, comparing the measured results with what we expected, we see that the measured bandwidth is lower than what we would expect for a gigabit network. This is because the network is experiencing a high amount of packet loss and delay, which is causing the available bandwidth to be lower than the network capacity. The high percentage of lost packets and out-of-order packets indicates that there is congestion in the network, and packets are being dropped or delayed due to insufficient network capacity.

If we are asked to use iPerf in UDP mode to measure the bandwidth, where we do not know anything about the network topology, we would take an iterative approach to estimate the bandwidth. We would start by setting the desired rate to a low value, and gradually increase it until we observe packet loss or delay in the test results. By doing so, we can determine the maximum achievable bandwidth without causing congestion in the network.

However, this approach may not be practical in some cases, as it requires multiple iterations to estimate the bandwidth accurately. In addition, it may not be feasible to conduct such tests in a production network as it could cause disruption to ongoing network traffic.

In the given results, we can see that the measured bandwidth for all instances is around 10.5 Mbps, which is lower than what we would expect for a gigabit network. The high percentage of lost packets and out-of-order packets indicate that the network is experiencing congestion, which is limiting the available bandwidth.

Lastly about the results, the h1-h4. During the 20-second interval, a total of 15.1.0 MB of data were transferred over these connections, resulting in a bandwidth 6.13 Mbps, respectively.

For the H1-H9 connection, During the 20-second interval, 14.3 MB of data was transferred, resulting in a bandwidth speed of 5.84 Mbps.

For the H7-H9 connection, During the 20-second interval, 16.1 MB of data was transferred, resulting in a bandwidth of 6.72 Mbps. The highest bandwidth speed was on h7-h9.

In case you were wondering what commands I used to measure all of this all you really needed was iperf -s -u on xterm h1, and open another window with xterm r1 and here you write iperf -c the host address and then -u –b (value of the M, which you can try with different values)M.

Considering that we are measuring RTT and Bandwidth, I am expecting to get H1 to H9 giving the best results considering this will be using a wide range of the devices, whilst after that I am pretty convinced that H7 to H9 will be second, because all of H7, H8 and H9 are connected to their own router, meaning that will give good results, this basically confirms that I think H1 to H4 will be last.

4.4 Test case 2: link latency and throughput

To complete the latency portion of the test case, I basically followed the portfolio guidelines, the command I used in the instance of L1, was opening R1 (router), with xterm R1, and then I used Ping 10.0.1.2 -c 25 > Latency_L1.txt

"ttl" stands for "time to live," which is a value in the IP header that is decremented by one for each hop (i.e. router) that the packet takes to reach its destination. This is used to prevent packets from circulating indefinitely in the network. In this case, the TTL is 63, which means that the device at 10.0.1.2 is on the same local network as the device sending the ping requests.

The final output shows the ping statistics for the 25 packets that were sent. The statistics show that all 25 packets were received without any loss, and the minimum, average, maximum, and standard deviation of the round-trip times are reported. The average round-trip time is 30.795 ms, with a standard deviation of 3.037 ms.

For L2 (between R2 and R3) the average RTT was 83.384 and the standard deviation of 11.931 ms, meanwhile for L3 (between R3 and R4) the average was 123.985 which is pretty high, and the standard deviation was approximately 5.577 ms. L1 obsviously had the best results and L3 the worst, meanwhile L2 had the one in the middle, but the latency changed a lot as you can see with the high standard deviation.

To complete this test-case I followed the portfolio guidelines, and to complete L1 (between R1 and R2) I used the following code: xterm r2 then python3 simpleperf.py –s -b 10.0.1.2 –p 8088 and then xterm r1 then python3 simpleperf.py –c –I 10.0.1.2 –p 8088 –t 25 > throughput_L1.txt. I am using the L1 testing between R1 and R2, as an example here:

The test tells you that "Transfer" is the total amount of data that was transferred during the test. In this case, it was 77.2 megabytes (MB). "Interval" indicates the duration of the test, which in this case was 25 seconds. "Bandwidth" is the rate at which data was transferred during the test. It is reported in megabits per second (Mbps), which is a measure of the number of bits of data that can be transmitted per second. In this case, the bandwidth was 24.29 Mbps.

The results for L2 where transfer was 65.3 megabytes, with the same interval, meanwhile the bandwidth speed was 20.36. The results for L3 where transfer was 56.0 megabytes, with the same interval, meanwhile the bandwidth speed was 17.74, giving L1 the highest bandwidth speed and L3 the lowest one. I had a hunch that L1 was going to give the fastest bandwidth speed when I saw the topology, but I had no idea which one of L2 and L3 were going to be faster than the other.

4.5 Test case 3: path Latency and throughput

The test case 3 report is going to be really like test case 2 considering we are doing the same with some tweaks, like instead of measuring between r1 and r2 for example, we are now going to be measuring between h1-h4, h1-h9 and h7-h9. Therefore, the only changes we really are going to make is to change the IP to the ones allocated to the hosts.

To complete the latency portion of the test case, I basically followed the portfolio guidelines from test case 2 and changed up to the host address, the command I used in the instance of h1-h4, was xterm h1, and then I used Ping 10.0.5.2 -c 25 > Latency_h1-h4.txt

The RTT for h1-h4 was 92.563 ms, and the standard deviation of 5.605 ms. For h1-h9 the results were 119.967 ms and the standard deviation of 9.934 ms. Lastly the results for h7-h9 were 63.377 and standard deviation of 0.790 ms, giving h7-h9 the best and more accurate latency, and h1-h9 the worst.

To measure bandwidth, I followed the portfolio guidelines, and to complete h1-h4 for example I used the following code: xterm h4 then python3 simpleperf.py –s -b 10.0.5.2 –p 8088 and then xterm h1 then python3 simpleperf.py –c –I 10.0.5.2 –p 8088 –t 25 > throughput_h1-h4.txt.

All of the results were measured in 25 seconds intervals, and they were h1-h4 had a transfer of 63.9 MB and a bandwidth speed of 19.99 mbps, meanwhile h1-h9 had a transfer value of 50.1 MB and a bandwidth speed of 15.81 mbps. Lastly the transfer of h7-h9 was 49.5 MB and the bandwidth speed 15.59 mbps. This means that h1-h4 had the fastest speed, and h7-h9 the slowest closely followed by h1-h9.

Personally, I thought h7-h9 was going to be the fastest since it was the fastest in the first test case, but I was surprised to find out that this throughput ended up being the slowest.

4.6 Test case 4: effects of multiplexing and latency

In this test case, I followed the guidelines, so I basically ran the previous commands again, the only difference this time is that we were supposed to run them at the same time. For me the results for the latency bit were that the latency average value was pretty similar to the previous test case, but the standard deviation was pretty high at approximately 10ms, meaning that multiplexing gives the latency very varying results for each second.

The bandwidth results were also very polarizing, and since we were running two or three throughput tests at the same time, it makes sense that results for the bandwidth were different as you can see in the measurements folder. Some of the transfer, and bandwidth speed was much slower, because of multiplexing.

Even though I know multiplexing influences the results, I did not expect it to influence the results in the manner it did.

4.7 Test case 5: effects of parallel connections

To measure this test case, I basically followed the guidelines and used the exact commands which were for example for: xterm h1 -> python3 simpleperf.py -c -I 10.0.5.2 -p 8088 -t 25.

In this example, using the results, it appears that multiple parallel connections were established between different pairs of hosts, and the bandwidth results were recorded for each connection.

For the H1-H4 connection, two connections were established with the destination IP address of 10.0.0.2 and different port numbers of 45342 and 45356. During the 25-second interval, a total of 22.0 MB and 13.5 MB of data were transferred over these connections, resulting in a bandwidth of 6.76 Mbps and 4.21 Mbps, respectively.

For the H2-H5 connection, a single connection was established with the destination IP address of 10.0.0.3 and port number 59022. During the 25-second interval, 20.2 MB of data was transferred, resulting in a bandwidth of 6.31 Mbps.

For the H3-H6 connection, a single connection was established with the destination IP address of 10.0.0.4 and port number 33306. During the 25-second interval, 18.4 MB of data was transferred, resulting in a bandwidth of 5.81 Mbps.

Based on these results, it seems that the parallel connections in general helped to increase the bandwidth compared to a single connection. The h1-h4 connection, which had two parallel connections, had the highest total amount of data transferred and the highest overall bandwidth. Meanwhile, the other two connections, which had a single parallel connection each, still achieved higher bandwidth than a single connection would likely have achieved.

The influence of parallel connection surprised me, I did not understand it having an impact like it did.

5 Conclusions

In conclusion, this document focuses on the implementation and evaluation of Simpleperf for measuring the performance. The limitations of the approach are acknowledged, including the specific test cases used and the potential impact of hardware and software configurations. The document presents the results of the performance evaluation for five test cases and discusses the implications of these results. I had some struggles implementing simpleperf, especially the second time around when I had to do the client part of the code again. The fourth test case was also hard, because my computer could barely handle all of the processes that were going on at the same time.

6 References (Optional)

1. https://android.googlesource.com/platform/system/extras/+/master/simpleperf/doc/README.md
2. Iperf.fr
3. https://developer.android.com/ndk/guides/simpleperf

NOTE:

The report cannot exceed 20 pages, including the list of references. The page format must be A4 with

2 cm margins, single spacing and Arial, Calibri, Times New Roman or similar 11-point font.

(I was not so sure what 2 cm margins meant. I even asked a TA and he did not even know. So I have

understood is as 1 cm on the right and left which is 2cm in total.)