SIMPLEPERF

Hamid Hamrah
S362052

17 April 2023

Oslo Metropolitan University

1 Introduction

simpleperf is a simplified version of iperf. Iperf is a network testing tool used to measure the bandwidth and performance of computer networks. It is a command-line utility that runs on multiple operating systems, including Linux, Windows, and macOS.

Iperf works by sending TCP or UDP data streams between two endpoints and measuring the performance of the connection. It can be used to test the network throughput, packet loss, and delay of a connection. Iperf can be used to test both wired and wireless networks, as well as Internet connections.

simpleperf has some of the features that Iperf has but not all of them. With simpleperf the user can measure the performance of the connection and see for example the bandwidth , the total data transferred and some other details.

The task given was to implement the simpleperf with some particular features. The way i planned to implement was to make two socket on each of our end points. Then I connect them together. The server socket has its connection open all the time. When I run the client side, The client sends a stream of data as much as possible in the packet size(1000) within the given time. Once the times is expired the connection from the client side closes and both side print how much data its been sent by the client and how much data its been received by the server.

2 Simpleperf

Simpleperf contains two files, server.py and client.py. I will try to explain how i implement the server side first and then i will go on and explain the client side. Through implementation i faced very few challenges with the server side where as the client side was much more challenging.

2.1 server

The server side of the simpleperf will be invoked by writing the following in the commend line

```
Python3 simpleperf.py -s
```

I have used the argsparser to take in arguments from the user in the commend line.The argsparser handles the parsing of those commands. In the server side I will take few arguments such as the ip adress, port number and the format type which the transferred data will be shown in. All the arguments are optional, which means if the user doesn't provide us any argument, there is a default value for it.

2.1.1 Start_server()

The start_server() functions is where I am making socket for the server side. I bind the socket to the IP address and the port given by the user. The socket is listening and waiting for any connection that is coming inn. I want the server to accept all the connection coming at all time. I could have specified time but here that's not the point. Here I am going to start and infinite loop, which will continue to accept incoming connections.

For each incoming connection I am going to make an object called conn and takes the address for that client. Since I should be able to handle multiple client at the same time and keep track of how much data each of them are sending. I need to make creates new threads for each client and handle them properly.

2.1.2 receive_data()

I wanted my code to be easy to understand and easy to develop that's why i made a separate function for receiving data. The function takes inn two parameters. Conn the object made in the start_server() function and the bufsize where its was specified that it should be 1000.

As I need to keep track of the data I am receiving for each object, I define a variable Total_bytes_received and set it to be zero and i will add to it every time i receive a chunk of the bufsize. The other variable I need is to keep track of the time which will be needed later to find the bandwidth.I set the start_time to the local time and when I start receiving data from the client. I start an infinite loop here as it will be handled later. As I need the the duration for the connection I will again define a variable that take sin the local machine time and set to be end_time. To find the duration for the connection To fin the elapsed time we subtarct the time from time.time(). At the end I will return the elapsed_time and the total_bytes_received.

2.1.3 handle_client(conn, addr, bufszie)

The function will take an object which was made in the main function, the address for the object and the bufsize which specified. The reason for having this function is because of multi-threading.Multi-threading is a programming concept that allows multiple threads of execution to run concurrently within a single process. A thread is a lightweight unit of execution that can perform a specific task or set of tasks independently of other threads in the same program.When I are making threads in the main function I want it target this part of your program for the the connection coming to the socket.

The data transfers through internet in form of bytes. That's why I need a function to convert them to different format. When the user run. it has the freedom of choosing which format the data should presented to them.If the user doesn't mention the format, its default is set to 'MB'.

The simpleperf has the capability to measure the bandwidth. since there are 8 bits in a byte, I need to multiply by 8 in order to convert to bytes per seconds.

Then I divide by the duration of the connection. Since I want to be presented by mbps I need to divide it again by 1,000,000.

2.2 client

My strategy to implement the client side was to divide the whole client side in small functions. The reason for that is effective coding as if i needed multiple time, i shouldn't write it again and again.The other reason is, its easy to troubleshoot if anything goes wrong.And for last, its easy to maintained and understand. So i will be explaining each function in a small section.

2.2.1 make_socket()

As i have divide the client side in small task and there are many cases involved. Its better to make a function which only make sockets. The function makes Takes inn IP address and port number for the server and connect to it. I have tried take of the exception here too. This function will be called whenever it needed to make a socket.

2.2.2 close_socket()

When ever we make socket, we want to be able to close that socket too.Here i have implement that the function takes inn the socket we mad and look for a good bye message sent from the server. Whenever the good bye message comes through, it will print it and close the function.

2.2.3 convert()

When the simple is run in client mode, the user have the chance to choose in which format they want to see the transfers data inn. That's we need a function to convert to different type of format. As we are sending data in bytes, we take that as the start point. If the format was something else then bytes then we have the formula for that to convert. So here the function takes inn the total bytes sent to the server and convert it to the preferred format bye the user.

2.2.4 default()

The default function is when the user wants to make a normal connection to a server. The user have the freedom to choose the IP address, port Numbers, the duration for the connection and the which format the data transferred to be shown. If none of those above are mention it will run on the default values set. Here we call and the make_socket() at first to make a socket. We want the client to sent as much packets in the chunks of 1000 bytes as long as we are under the time given by the user. We check if we are under the time given the user with a while loop.At when its equal to the time given by the user we will go out of the loop and close the socket with close_socket() function.

2.2.5 interval()

When simpleperf is invoked in client mode with -i flag, The interval function will print the statistic per -i seconds. In this function we need multiple variable in order to keep track of how much data its sent in each interval.

2.3 parallel()

This function will be used when the user wants to make parallel connection form one client to the server.To make the parallel connection possible we need to use multi-threading in order to keep track of of each thread. Here we use for loop which makes as many thread as the number of parallel given by the user. Here are going to reuse the code for the default part. We do the same as the default here but with many connections.

2.3.1 num()

In simpleperf the user can send an specific amount of data to the server. That's where the num() function will be invoked. Since the user can need to specify the format for the data and the amount we need to separate them first . Then we go on a check which format they are inn and then convert them to bytes depending on the format. We go on a while loop and holds the connection as long as the bytes expect to send is equal to the total sent. Once they are equal we go out of the while loop and closet the connection.

2.3.2 run_client()

Running simpleperf with -c will invoke the run_client(). We have four cases in the client side and each case run depends on which flag is invoked in the simpleperf. The function takes the whole args as a parameter.

2.4 validation functions

Through my report i choose to write some functions to validate port, Ip, time given by the user and the number of parallel connection given by the user. The valid_port function will takes in the a port number given in the commend line, the function will will check whether its in the range I have defined or not. If not it will raise an error telling the user that port is not in the defined range.

TO validate the ip address we import the library Ip-address.this function takes in the IP address given in the command line and checks with the library which I import in our code.

To check that the user is not giving any negative time. we write a function for it. The function accepts only values greater then zero.If the user give something below zero then it raise an error. The last validation function is for the number of parallel connection.This function takes the number of parallel connection by the user and validates it either its in defined range or not. The number of parallel connection which can be accepted starts form 1 and until 5. The limitation was mention in the task given. I raise an error if number of parallel exceeds the range given.

3 Experimental setup

The code is tested on topology given in hand. The topology Ire made of 9 hosts and 4 routers.hosts1,2 and 3 Ire connected to router 1. host4,5,6 and 8 Ire connected to router 3, host7 to router 2 and host9 to router4.

4 Performance evaluations

For performance evaluations I am going to use some network tools where i get an output. The output will get us a glimpse of the performance metrics we are looking for.

4.1 Network tools

Network tools I used in the performance metrics were Ping, Iperf and simpleperf. Ping and Iperf were preinstalled and the simpleperf is my implementations.

4.2 performance metrics

In this section I are going to evaluate the performance of our network with Iperf and Simpleperf. The performance matrics I am going to look at are going to the delay time betIen our connection and the throughput for the connections. As its mentioned before the topology for the network Ire given in hand.

4.3 Test case 1

In order to find the bandwidth between the server a client in UDP mode I use the command below

```
server: Iperf -s -u

client: iperf -c 10.0.?.? -u -b ??Mb
```

The correct quantity must be chosen for -b to properly use the bandwidth when I want to measure the bandwidth between a server and a client. If I select a value for -b that is too high, numerous packets will be discarded while the connection is active. Instead of doing this, we select a value that is too low, overestimate the bandwidth of the link, and then underutilize it. Therefore, it's crucial to pick the appropriate value for -b in order to minimize the number of missed packets while maximizing the bandwidth of the connection between the server and the client.

4.3.1 h1-h4

Test No	x Value	Bandwidth	Lost/Total Data-grams
1	30Mb	29.1 Mbits/sec	7.2%
2	29Mb	29.2 Mbits/sec	3.9%
3	28Mb	29.1 Mbits/sec	0.64%

From the topology we know that the maximum bandwidth between h1 and h4 is set to 30Mb. So we start with by setting -b 30Mb. We see that the 7.2% of the data sent from the server is been dropped.It means that we should lower the the amount for -b and set to 29Mb. We see again that 3.9% of the data are again dropped. So we lower the amount to -b 28Mb. Now we see that 0.64% packets are dropped again.If we go lower then 28Mb, Then the we are not utilizing the maximum capacity for the bandwidth.

4.3.2 h1-h9

Test No	x Value	Bandwidth	Lost/Total Datagrams
1	20Mb	19.3 Mbits/sec	8%
2	19Mb	19.4 Mbits/sec	2.4%
3	18Mb	18.9 Mbits/sec	0%

According to the topology, the maximum bandwidth between h1 and h4 is 30Mb. Therefore, we begin by setting -b 30Mb. We can see that 7.2% of the data that the server sends is lost.This suggests that the quantity for -b should be decreased and set to 29Mb. We can once more see that 3.9% of the data are lost. So, we reduce it to -b 28Mb. We can see that 0.64 percent of packets are dropped once more.If the amount of bandwidth used is less than 28Mb, the bandwidth is not being used to its full potential.

4.3.3 h7-h9

Test No	x Value	Bandwidth	Lost/Total Datagrams
1	20Mb	19.4 Mbits/sec	7.2%
2	19Mb	19.4 Mbits/sec	2.3%
3	18Mb	18.9 Mbits/sec	0%

I perform the same procedure for h7 through h9 and set -b 20MB. Then, 7.2% of the packets are dropped. So I set -b 19Mb and reduce the value here as well. This time, 2.3% of the packets droop. I then reduce it once more and set it to 18Mb. Now, 0% of the packets were dropped.

4.3.4 Conclusion

Now, if you want to find the bandwidth but are unsure about the network structure. There are various methods for doing that. One method is to run numerous tests with various packet sizes and rates, then evaluate the results, like I did here. It's crucial to understand that there are some restrictions to take into account. The outcome may be impacted by things like congestion, packet loss, or other interference, for instance, which could end up in inaccurate results. Another drawback is that real-time applications could need a certain level of performance assurance because it only gives a general idea of the bandwidth that can be attained in specific circumstances.

4.4 Test case 2

The latency between r1 and r2 will be examined for case 2. By using simpleperf as client on one end and the server on the other, I'll also be able to determine the bandwidth between the routers. In order to determine L1's latency. I'll ping from r2 to r1. Given that the delay is specified in the topology to be 10ms, I anticipate the result to be somewhere close to that value..

rtt min/**avg**/max/mdev = 20.431/**22.046**/23.695/0.936 ms

I pinged 25 packets , with an RTT of 22.046 ms on average. Now that's for delay in both directions as I transmit a packet from r1 to r2, start the timer, and when I get an acknowledgement from r2, stop it. When I divide it by 2, the result is 11.03ms, which is quite close to what I anticipated. Verify the bandwidth between r1 and r2 now. On r2, I will first run our simpleperf server. On r2, I will connect it by running simpleperf as a client. The bandwidth ought to be close to 40Mbps, as I anticipate.

10.0.2.1:8088 0.0 - 25.0 119.23MB **38.43** Mbps

My current bandwidth is 38.43 Mbps, which is fairly close to what I had anticipated.That's good for now, and we'll talk about why there was a discrepancy in the conclusion. I now go through the same procedure again to determine the latency and bandwidth for L2 and L3. I ping from r3 to r2 to determine the L2 latency. Since the topology has the delay set to be 20ms, I anticipate that the latency will be near to that value. The identical measurement is made as I did for L1, and the results are analyzed. I ping r3 and r2 first.

rtt min/**avg**/max/mdev = 40.372/**40.668**/41.161/0.185 ms

The average RTT we received was 40.668 ms; we divided this by two to obtain the latency, which came out to 20.334 ms, which is quite close to my expectation. Let's run simpleperf on each of our end points and verify the bandwidth. I anticipate that the bandwidth for l2 will be around 30 Mbps.

10.0.3.2:8088 0.0 - 25.0 91.21MB **29.19** Mbps

There is nothing weird about this because I understand that the bandwidth is 29.19 Mbps, which is fairly close. I conduct L3 using the same procedure. I will ping from r4 to r3 now. I'm anticipating a result that is close to 10 ms.

rtt min/**avg**/max/mdev = 20.297/**20.611**/21.443/0.234 ms

I received an average RTT of 20.661 ms, so we divide it by two to get latency10.335 ms, which is really close to my expectation. Let's examine L3's bandwidth. On each of our endpoints, in this case the server on r4 and the client on r3, we launch simpleperf. I anticipate that the outcome will be 20Mbps.

10.0.6.2:8088 0.0 - 25.0 59.98MB **19.19** Mbps

The bandwidth I got is 19.19Mbps which is very close to what I expect.

4.4.1 Conclusion

I obtained results from all of my testing that were really close to what I had anticipated for the latency. We are aware that there are four factors that can affect latency, including transmission and propagation delays, which are linked-dependent, and processing and queuing delays, which are traffic- and router-dependent. Therefore, if each of the four components has a 0.1 ms delay added, there will always be a discrepancy from the delay specified in the topology.But given that we are using Linux instead over Windows, I could have been confused due to hardware limitations. We come extremely near to the topology's specified bandwidth for this parameter. Now, it's critical to understand that the link's maximum bandwidth is determined by the topology. Therefore, it won't always be used to its full potential. As a result, there is a discrepancy between what we anticipate and what we receive.

4.5 Test case 3

In test case 3, we will perform the same test as in test case 2, except we will ping and run simpleperf on the host rather than the router. We want to analyze the path latency and bandwidth between two hosts in a network.

4.5.1 h1-h4

To determine the time delay between h1 and h4. I'm going to start pinging h1 from h4. Considering what I know about the topology diagram, a packet from h1 must pass through L1 and L2 to get to h4. Now, the amount of delay I should be anticipating here should equal the total of the L1 and L2 latency, which is 30ms. Thus, the delay should be around that much when I ping h4 from h1.

rtt min/**avg**/max/mdev = 60.673/**61.335**/64.769/0.864 ms

We can get the latency by dividing the average RTT by 2, which gives us a result of 30.665 ms, which is about what I anticipated. Next, let's examine the bandwidth between h1 and h4. In order to determine the bandwidth, I first launch SimplePerf in Client Mode on h1 and then in Server Mode on h4. Currently, L1 has a bandwidth of 40Mb, whereas L2 has a capacity of 30Mb. When a packet needs to pass across two links with different bandwidths in order to reach a server, the bandwidth of the connection with the lower bandwidth will be important. As a result, I am assuming that the bandwidth should be close to 30Mbps.

10.0.5.2:8088 0.0-25 88.56 MB **28.34** Mbps

Now, my outcome is rather close to the value I was hoping for. I conduct the identical test twice more and examine the results to be sure this is not a coincidence.

4.5.2 h7-h9

For h7 and h9, I perform the same test again. I begin by pinging h9 from h7. Given that the packet should pass through L2 and L3, I anticipate the latency to be close to 30 ms in this case. Therefore, the L2 and L3 delay is 30 ms.

rtt min/**avg**/max/mdev = 60.642/**61.068**/61.450/0.240 ms

So, the RTT 61.068 divided by 2 resulted in a time of 30.354 ms, which is quite close to my expectation. Using the bandwidth between h9 and h7, let's check. Since the bandwidth for L3 is the lowest at this location, I am anticipating 20Mb.

10.0.7.2:8088 0.0-25 53.60 MB **17.15** Mbps

Based on what I expected, I got a similar result. My suspicion is that the difference between what I expected and what I got might be the number of routers between our end points and a hardware shortage.

4.5.3 h1-h9

For h1 and h9 I run the same tests as above. For the latency between I ping h9 from h1. Here I am expecting the latency to be close to 40ms.

rtt min/**avg**/max/mdev = 80.848/**81.793**/84.071/0.566 ms

The latency result I received was 40.742ms, which is pretty close to what I had anticipated and indicates that there was no traffic on our network at the time of the ping. I use simpleperf on our end points to measure the bandwidth between h1 and h9. I'm anticipating a bandwidth of 20Mb.

10.0.7.2:8088 0.0-25 56.94 MB **18.22** Mbps

The result I got is close and the difference might be again the number of routers, hardware shortage and other factors.

4.5.4 Conclusion

I gained this information, to describe the tests. Firstly, there is a correlation between the number of connections and their latency and the delay between two end end points. I thus sum the amount of delay times for each link to determine the latency between two end points in order to roughly determine the latency between hosts. Second, I realized that numerous links with various bandwidths are available while I'm trying to discover bandwidth. It is important to consider the bandwidth of the link with the lowest bandwidth. The bandwidth for L3 is important since it is lowest between 1 and h9.

4.6 Test case 4

I'll examine the impact of multiplexing on latency and bandwidth in test case 4.I have four short examinations through test case 4, which I will examine and then draw conclusions on. It was a limited margin of deference because there are several hosts that should connect at once, but we can see how it functions.

4.6.1 h1-h4 & h2-h5

I am aware that the latency between h1 and h4 should be 30 ms and that between h2 and h5 it should also be around the same amount based on the topology and test case 3.Now, though, I'm curious as to what happens to the latency when two clients are connected to separate servers using the same network. My first step is to launch our simpleperf on the server in our endpoints. I then ping h4 from h1 and h5 from h2 after that. Given the volume of traffic on the link, I anticipate that the delay I experience will be greater than the one stated in the topology. I now just have the RTT times for each, but if I split them by two to

Pair	rtt avg	Expected	Bandwidth
h1-h4	73.578ms	15Mbps	13.45 Mbps
h2-h5	71.927ms	15Mbps	15.54Mbps

account for latency, I end up with a result that is around 35 ms. Currently, both results are greater than I anticipated but are still quite close to one another.The huge volume of traffic to the link is likely responsible for it.It happens because traffic causes the link's latency to increase as it takes longer for packets to reach their destinations. Let's now examine what happens to the link's bandwidth when two hosts are connected to it at the same time.In order to link them for 25 seconds, I start our simpleperf on each of our endpoints. Now, about the bandwidth, I am aware from topology and test case 3 that it should be 30 Mbps as that is the lowest between these two connections. However, given the presence of two connections, I anticipate that the bandwidth will be cut in half.

It's fascinating since I received two different bandwidths from them even though they were both running via the same network at the same time. I can easily see that one is less than 1/2 the links bandwidth and the other is little over halve.

That implies that the link's bandwidth has been divided among the connections, but it is obvious that the distribution wasn't equitable because one connection received more bandwidth than the other.

4.6.2 h1-h4 & h2-h5 & h3-h6

Here, I go through the procedure again, but this time, h3 and h6 are connected one more time.Let's now examine the additional delay and bandwidth it will contribute. I first ping them to check the latency. I have bigger expectations than I did for the prior test.

Pair	**rtt avg**	**Expected**	**Bandwidth**
h1-h4	73.212ms	10 Mbps	9.61 Mbps
h2-h5	74.197ms	10 Mbps	11.4 Mbps
h3-h6	72.428ms	10Mbps	8.63Mbps

Now, there isn't much of a difference between the three outcomes I obtained; they are extremely similar to the one when I had two connections. Let's examine the impact of three connections using the same link on the bandwidth. Since L2 has a 30Mb bandwidth, that is the lowest bandwidth between these connections. I can see that in order to be equitable for all connections, it should have been split by three and then dispersed among them. But this place didn't do it. In comparison to the other two connections, h1-h5 received higher traffic.

4.6.3 h1-h4 & h7-h9

Now, in this test, I'm going to proceed as before, but the distinction is made when our endpoints have different delays from one another. I begin executing simpleperf on our end points. I anticipate that the delay period between h1 and h4 should be close to 30 ms. I anticipate that the delay period should be similar to 30 ms between h7 and h9.However, because they both share the L2, I can check to see if the latency will change as a result.

Pair	**rtt avg**	**Expected**	**Bandwidth**
h1-h4	73.722ms	15 Mbps	18.15 Mbps
h7-h9	72.939 ms	15 Mbps	10.9 Mbps

For the delay, which is RTT/2, I essentially obtained the same outcome as above. I suspect that while several connections are active, the latency may rise.But I don't come to a conclude until I administer the last test in this chapter. I can test what happens to the bandwidth first, though, before that. The h1-h4 and h7-h9 share the L2 for the connections, as I previously said. Let's now execute our simpleperf on our endpoints and examine the outcomes. It's interesting to note that for h1-h4, I receive 2/3 of the bandwidth for L2, while for h7-h9, I received 1/3. due to the fact that the minimum bandwidth between h1 and h4 is 30Mb. However, the minimum bandwidth between h7 and h9 is 20Mb. As a result, anytime data passes via L2.Since there is interference up till h4, h4 may receive the info more easily. after that, to L3. For this reason, the bandwidth between h7-h9 is less than that between h1-h4.

4.6.4 h1-h4 & h8-h9

I do the previous tests again. The two connections here simply share one router, and that is the sole difference. Let's now investigate its relationship to latency. I estimate that the delay between h1-h4 is around 30 ms, but it should be 10 ms for h8-h9. Now the the latency is again higher then what it should be which Rtt/2.

Pair	rtt avg	Expected	Bandwidth
h1-h4	68.137ms	30 Mbps	27.81 Mbps
h8-h9	25.393ms	20 Mbps	19.26 Mbps

Lets see if sharing a router will make a difference for the bandwidth. I think it should have anything to do with the bandwith and expect the bandwidth for h1-h4 should be 30Mb and 20Mb for h8-h9. The result we got here is that the connection between h8-h9 have very close to the bandiwdth set in the topology, but on the other hand the bandwidth between h1-h4 have more difference then what its set in the topology. That means the connection between h8-h9 is been more prioritized then the other connection.

4.6.5 Conclusion

Based on the tests conducted, I have reached several conclusions. Firstly, it is evident that a large number of connections on the same link leads to increased traffic on that link, thereby increasing latency. Secondly, when multiple connections share the same link, the available bandwidth is roughly divided among the connections, resulting in unfair distribution. The next step in this process plays a significant role in the distribution, with links having lower delay times receiving a larger share of the available bandwidth. It is important to note that this distribution is not equitable, with certain connections receiving more bandwidth than others. Finally, the tests also showed that when multiple connections share the same router, it has minimal effect on latency but can impact bandwidth depending on how the router prioritizes the connections.

4.7 Test case 5

In test case 5, I plan to connect three clients to three servers, with h1 establishing parallel connections to h4. Parallel connections refer to a situation where a single client establishes multiple connections to the same server, providing an advantage in terms of multiple paths to the server. The aim of this test is to assess the impact of parallel connections on the bandwidth of our connections. Prior to running the simpleperf test, I anticipate that, as they all share L1 and L2 caches, the minimum bandwidth of 30Mb will be divided by three for the three connections. However, since h1 has an additional connection, the bandwidth will be roughly divided by four and then distributed. Let's execute the test and examine the results.

Pair	**Expected**	**Bandwidth**
h1-h4	7.5 Mbps	7.02 Mbps
h1-h4	7.5 Mbps	8.30 Mbps
h2-h5	7.5 Mbps	7.49 Mbps
h3-h6	7.5 Mbps	7.07 Mbps

Based on the results obtained from test case 4 and test case 5, it is evident that the available bandwidth is divided among the connections in an unfair manner. It is worth noting that h1, which has two connections, receives a larger share of the link's bandwidth compared to the other clients. In conclusion, hosts that establish parallel connections to the server tend to receive a larger share of the available bandwidth when other connections are present. This approach can be seen as a method of tricking the network into providing more bandwidth by registering an additional connection.

5 Conclusions

I want to state at the end of this endeavor that I have learned a lot from it. In addition to coding, I now have a better knowledge of how networks communicate.

I faced a lot of challenge while implementing the client side. specifically the interval and numb part. I would say it was time consuming cause i have never done it before. I too encountered difficulties while doing tests on my computer. The value I received was completely different from what I had anticipated. I ran the test on a different PC than mine because of this.

All in all, I attempted to be user-friendly and account for any potential corner circumstances while implementing. The user receives extremely detailed instructions on what they did incorrectly and how to run again.