portfolio-1

Shatha Amer

S362064@oslomet.no

Oslo Metropolitan University

April 17, 2023

[1 Introduction 2](#_Toc131105778)

[2 Simpleperf 2](#_Toc131105779)

[3 Experimental setup 3](#_Toc131105780)

[4 Results and discussion Error! Bookmark not defined.](#_Toc131105781)

[4.1 Network tools 4](#_Toc131105782)

[4.2 Performance metrics 4](#_Toc131105783)

[4.3 Test case 1: measuring bandwidth with iperf in UDP mode 5](#_Toc131105784)

[4.3.1 Results 5](#_Toc131105785)

[4.3.2 Discussion 5](#_Toc131105786)

[4.4 Test case 2: link latency and throughput 6](#_Toc131105787)

[4.4.1 Results 6](#_Toc131105788)

[4.4.2 Discussion 7](#_Toc131105789)

[4.5 Test case 3: path Latency and throughput 7](#_Toc131105790)

[4.5.1 Results 7](#_Toc131105791)

[4.5.2 Discussion 8](#_Toc131105792)

[4.6 Test case 4: effects of multiplexing and latency 9](#_Toc131105793)

[4.6.1 Results 9](#_Toc131105794)

[4.6.2 Discussion 12](#_Toc131105795)

[4.7 Test case 5: effects of parallel connections 13](#_Toc131105796)

[4.7.1 Results 13](#_Toc131105797)

[4.7.2 Discussion 13](#_Toc131105798)

[5 Conclusions 14](#_Toc131105799)

[6 References 14](#_Toc131105800)

# Introduction

In this report I will discuss The document which presents a Python-based network performance measurement tool called "simpleperf" that can be used to measure network performance between a client and a server. The tool allows users to assess the performance of their network connection by measuring metrics such as transfer rate and transfer amount in different units (B, KB, MB) for a specified duration of time.

Key topics: The key topics of this document are network performance measurement, client-server communication, and data transfer.

Problems being solved: The tool aims to solve the problem of measuring network performance accurately and efficiently by providing a simple and flexible way to measure various performance metrics in different units. It also allows users to customize the duration of the measurement, the number of parallel connections, and the interval at which statistics are collected during the transfer.

Relevant work references: This tool is inspired by the popular network performance measurement tool "iperf" which is widely used for network benchmarking and testing. Some of the concepts and approaches used in simpleperf are based on the functionality provided by iperf.

Approach to the solution: The solution is implemented in Python and utilizes the socket module for communication over the network. It provides both client and server modes, where the server listens for incoming connections and the client initiates data transfer. The tool uses multi-threading to support parallel connections and measures various performance metrics during the transfer.

Limitations and outcomes: The tool has some limitations such as being limited to measuring network performance between a single client and server, not supporting advanced features like encryption or authentication, and relying on the accuracy of the system clock for measuring time. However, the tool provides an easy-to-use and customizable way to measure network performance and can be used as a starting point for further enhancements or as a standalone tool for basic network performance measurement.

# Simpleperf

The simpleperf code appears to implement a simple network performance measurement tool using socket programming in Python. It allows the user to run the tool in either server mode or client mode, and provides options to configure various parameters such as IP address, port number, transfer size, transfer rate format, duration, and interval for statistics.

Let's go through the different parts of the code:

**Importing Modules**: The code imports several Python modules including socket for enabling network communication, time for measuring time intervals, threading for concurrent execution, sys for system-related operations, and argparse for parsing command-line arguments.

**Parsing Command-line Arguments:** The parse\_args() function uses the argparse module to parse the command-line arguments provided by the user. It defines the available options and their corresponding descriptions, and stores the parsed arguments in an object called args. It also includes some validation checks for the format and interval options.

**Printing Summary**: The print\_summary() function takes the server address, total bytes transferred, transfer time, and format as input arguments, and prints a summary of the results. It calculates the transfer amount in the specified format (B, KB, MB), the elapsed time in seconds, and the transfer rate in Mbps.

**Server Function:** The server() function implements the server mode functionality. It creates a TCP socket using the socket module, binds it to a specific address and port, and starts listening for incoming connections. When a new client connects, it spawns a new thread using threading.Thread to handle the connection concurrently. The handleConnection() function is called to handle the communication with the client.

**Start Connection Function**: The handleConnection() function is called for each client connection received by the server. It takes the connected socket object, client address, and parsed arguments as input arguments. It receives data from the client in chunks of BUFFER\_SIZE bytes, calculates the total bytes received, and measures the transfer time. It also prints the progress and statistics at the specified interval. Finally, it closes the connection.

**Client Function: The client()** function implements the client mode functionality. It creates a TCP socket using the socket module, connects to the specified server address and port, sends data in chunks of BUFFER\_SIZE bytes, calculates the total bytes sent, and measures the transfer time. It also prints the progress and statistics at the specified interval. Finally, it closes the connection.

**Main Code**: The main code block calls the parse\_args() function to parse the command-line arguments, and checks if the user has specified either client mode or server mode. If neither is specified, an error is raised. If server mode is specified, the server() function is called with the parsed arguments. If client mode is specified, the client() function is called with the parsed arguments.

Overall, this code provides a basic implementation of a network performance measurement tool with server-client functionality and options to configure various parameters. However, it may require further testing, error handling, and optimization for production use.

# Experimental setup

The code appears to implement a simpleperf tool for measuring network performance. The setup involves a client-server architecture where the tool can operate in either client mode or server mode. The server listens on a specified port and the client connects to the server using the server's IP address and port number. The client and server can communicate over a virtual network or topology, which is not explicitly described in the code.

Diagram

Description automatically generated

# Performance evaluations

## **Network tools**

The code does not explicitly use any network tools such as iperf or ping. However, it implements its own custom network performance measurement tool using the Python socket module for communication between the client and server.

## **Performance metrics**

The performance metrics used to evaluate the simpleperf tool include:

Transfer time (elapsedTime): Measured in seconds, it represents the time taken to transfer data between the client and server.

Transfer amount (transferAmount): Measured in bytes, it represents the total amount of data transferred between the client and server.

Transfer rate (transferRate): Measured in Mbps (megabits per second), it represents the rate at which data is transferred between the client and server.

Format (args.format): Specifies the desired format (B, KB, MB) for the summary of results, allowing for customization of the output format.

Interval (args.interval): Specifies the interval at which statistics should be collected during the transfer, allowing for customization of the data collection frequency.

Parallel connections (args.parallel): Specifies the number of parallel connections to use during the transfer, allowing for customization of the parallelism level.

Number of bytes to transfer (args.num): Specifies the number of bytes to transfer, allowing for customization of the amount of data to be transferred.

Total duration of the operation (args.time): Specifies the total duration of the operation in seconds, allowing for customization of the measurement duratio.

## **Test case 1: measuring bandwidth with iperf in UDP mode**

### **Results**

throughput\_udp\_iperf\_h1-h4

[ 1] local 10.0.0.2 port 40867 connected with 10.0.5.2 port 5001

[ ID] Interval Transfer Bandwidth

[ 1] 0.0000-10.0002 sec 37.5 MBytes 31.5 Mbits/sec

[ 1] Sent 26752 datagrams

[ 1] Server Report:

[ ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams

[ 1] 0.0000-10.0392 sec 27.5 MBytes 23.0 Mbits/sec 0.355 ms 7101/26751 (27%)

[ 1] 0.0000-10.0392 sec 124 datagrams received out-of-order

### 

### throughput\_udp\_iperf\_h1-h9

[ 1] local 10.0.0.2 port 35583 connected with 10.0.7.2 port 5001

[ ID] Interval Transfer Bandwidth

[ 1] 0.0000-10.0005 sec 25.0 MBytes 21.0 Mbits/sec

[ 1] Sent 17836 datagrams

[ 1] Server Report:

[ ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams

[ 1] 0.0000-10.0142 sec 20.2 MBytes 17.0 Mbits/sec 0.648 ms 3391/17835 (19%)

[ 1] 0.0000-10.0142 sec 176 datagrams received out-of-order

throughput\_udp\_iperf\_h7-h9

[ 1] local 10.0.2.2 port 46389 connected with 10.0.7.2 port 5001

[ ID] Interval Transfer Bandwidth

[ 1] 0.0000-10.0009 sec 25.0 MBytes 21.0 Mbits/sec

[ 1] Sent 17837 datagrams

[ 1] Server Report:

[ ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams

[ 1] 0.0000-10.0128 sec 20.4 MBytes 17.1 Mbits/sec 0.723 ms 3301/17836 (19%)

[ 1] 0.0000-10.0128 sec 79 datagrams received out-of-order

### 

### **Discussion**

The results in these three cases indicate that there are significant packet losses. The bandwidth also appears to be lower than the expected bandwidth, which may indicate network congestion or other network issues causing packet losses and delays. The Jitter value in the server indicates the variation in delay between received packets, with higher values indicating more variability in packet arrival times. These issues could be caused by network congestion, high network latency, insufficient buffer sizes, or other network issues.

Explain your results (what you expected vs what you got)

## **Test case 2: link latency and throughput**

### **Results**

Latency\_L1

--- 10.0.1.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24048ms

rtt min/avg/max/mdev = 20.500/21.763/23.212/0.755 ms

Latency\_L2

--- 10.0.3.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24039ms

rtt min/avg/max/mdev = 40.240/41.338/42.361/0.635 ms

Latency\_L3

--- 10.0.6.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24059ms

rtt min/avg/max/mdev = 21.201/22.535/25.560/1.035 ms

throughput\_L1

A simpleperf client connecting to server 10.0.1.2, port ('10.0.1.1', 45648)

---------------------------------------------

ID Interval Transferred Rate

('10.0.1.1', 45648) [0.0 - 25.27] 76.51 24.22 MBps

throughput\_L2

---------------------------------------------

A simpleperf client connecting to server 10.0.3.2, port ('10.0.3.1', 53344)

---------------------------------------------

ID Interval Transferred Rate

('10.0.3.1', 53344) [0.0 - 25.57] 65.94 20.63 MBps

throughput\_L3

---------------------------------------------

A simpleperf client connecting to server 10.0.6.2, port ('10.0.6.1', 51992)

---------------------------------------------

ID Interval Transferred Rate

('10.0.6.1', 51992) [0.0 - 25.39] 45.54 14.35 MBps

### **Discussion**

All 25 packets were received without any loss, indicating a 0% packet loss. the ping commands were able to successfully send and receive packets to both IP addresses without any loss. A 0% packet loss indicates that all packets sent were successfully received without any loss, which is desirable for a healthy network connection.

The round-trip time (RTT) is the time taken for a packet to travel from the source to the destination and back to the source, and it is measured in milliseconds (ms). The average RTT gives an indication of the overall latency of the network, with lower values indicating faster response times.

Lower RTT values and minimal packet loss generally indicate a better network performance, while higher RTT values and significant packet loss may suggest network congestion, latency issues, or other network problems.

In the simpleperf and based on the results. It appears that the client connected to three different servers and measured the performance of the network connection in terms of data transfer rate. These results may indicate the efficiency and speed of data transfer between the client and the servers, with higher transfer rates generally indicating better performance.

## **Test case 3: path Latency and throughput**

### **Results**

latency\_h1-h4

--- 10.0.5.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24047ms

rtt min/avg/max/mdev = 61.261/63.173/66.518/1.324 ms

latency\_h7-h9

--- 10.0.7.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24030ms

rtt min/avg/max/mdev = 61.328/63.399/73.114/2.216 ms

latency\_h1-h9

--- 10.0.7.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24047ms

rtt min/avg/max/mdev = 82.472/85.180/97.821/3.333 ms

throughput\_h1-h4

---------------------------------------------

A simpleperf client connecting to server 10.0.5.2, port ('10.0.0.2', 50376)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.2', 50376) [0.0 - 25.51] 71.67 22.47 MBps

throughput\_h7-h9

---------------------------------------------

A simpleperf client connecting to server 10.0.7.2, port ('10.0.2.2', 37238)

---------------------------------------------

ID Interval Transferred Rate

('10.0.2.2', 37238) [0.0 - 25.46] 40.73 12.8 MBps

throughput\_h1-h9

---------------------------------------------

A simpleperf client connecting to server 10.0.7.2, port ('10.0.0.2', 39270)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.2', 39270) [0.0 - 25.56] 42.78 13.39 MBps

### **Discussion**

The output you provided shows the result of a network ping command executed on two different IP addresses: 10.0.5.2 and 10.0.7.2. For the first IP address, 10.0.5.2, the ping command sent 25 packets and received 25 packets in return. There was 0% packet loss. For the second IP address, 10.0.7.2, the ping command also sent 25 packets and received 25 packets in return. There was 0% packet loss.

The results indicate that the simpleperf client was able to transfer data at different rates to different servers with different IP addresses and port numbers. The specific reasons for the observed performance results can vary depending on various factors, such as the network conditions, server configurations, and the load on the servers at the time of the performance measurements. Factors such as network congestion, server processing capacity, and client-server communication efficiency can all impact the observed data transfer rates.

## **Test case 4: effects of multiplexing and latency**

### **Results**

Throughput\_h1-h4-1.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.2, port ('10.0.0.2', 48418)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.2', 48418) [0.0 - 25.63] 38.26 11.94 MBps

Throughput\_h2-h5-1.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.3, port ('10.0.0.3', 59516)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.3', 59516) [0.0 - 25.56] 49.65 15.54 MBps

Throughput\_h1-h4-2.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.2, port ('10.0.0.2', 54488)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.2', 54488) [0.0 - 25.51] 67.54 21.18 MBps

Throughput\_h2-h5-2.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.3, port ('10.0.0.3', 50956)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.3', 50956) [0.0 - 25.5] 65.66 20.6 MBps

Throughput\_h3-h6-2.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.4, port ('10.0.0.4', 40838)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.4', 40838) [0.0 - 25.82] 70.15 21.74 MBps

Throughput\_h1-h4-3.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.2, port ('10.0.0.2', 60916)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.2', 60916) [0.0 - 25.52] 70.04 21.96 MBps

Throughput\_h7-h9-3.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.7.2, port ('10.0.2.2', 40094)

---------------------------------------------

ID Interval Transferred Rate

('10.0.2.2', 40094) [0.0 - 25.78] 40.21 12.48 MBps

Throughput\_h1-h4-4.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.2, port ('10.0.0.2', 35534)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.2', 35534) [0.0 - 25.56] 71.84 22.49 MBps

Throughput\_h8-h9-4.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.7.2, port ('10.0.4.2', 56060)

---------------------------------------------

ID Interval Transferred Rate

('10.0.4.2', 56060) [0.0 - 25.34] 46.65 14.73 MBps

Latency\_h1-h4-1.txt

--- 10.0.5.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24052ms

rtt min/avg/max/mdev = 63.531/65.199/69.142/1.413 ms

Latency\_h2-h5-1.txt

--- 10.0.5.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24044ms

rtt min/avg/max/mdev = 62.054/64.459/68.757/1.799 ms

Latency\_h1-h4-2.txt

--- 10.0.5.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24052ms

rtt min/avg/max/mdev = 63.031/65.456/69.162/1.582 ms

Latency\_h2-h5-2.txt

--- 10.0.5.3 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24043ms

rtt min/avg/max/mdev = 61.830/63.661/71.150/2.192 ms

Latency\_h3-h6-2.txt

--- 10.0.5.4 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24106ms

rtt min/avg/max/mdev = 62.359/65.660/84.503/4.311 ms

Latency\_h1-h4-3.txt

--- 10.0.5.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24070ms

rtt min/avg/max/mdev = 61.472/64.714/68.177/1.723 ms

Latency\_h7-h9-3.txt

--- 10.0.7.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24048ms

rtt min/avg/max/mdev = 62.877/65.392/68.381/1.441 ms

Latency\_h1-h4-4.txt

--- 10.0.5.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24047ms

rtt min/avg/max/mdev = 61.837/63.358/67.455/1.284 ms

Latency\_h8-h9-4.txt

--- 10.0.7.2 ping statistics ---

25 packets transmitted, 25 received, 0% packet loss, time 24043ms

rtt min/avg/max/mdev = 20.704/21.936/26.720/1.235 ms

### **Discussion**

Data transfer rates vary: The Rate column shows that the data transfer rates vary among the different server connections. For example, rates range from 11.94 MBps to 71.84 MBps, indicating differences in performance and efficiency of data transfer between servers.

The Rate column also reveals that the data transfer rates may fluctuate over time, as seen in the varying values within the same ID across different intervals. This could be due to various factors such as network congestion, server load, or other performance-related issues.

Potential impact of network topology: The results indicate that the network topology, including IP addresses and ports, can affect the performance of data transfer. For example, connections to servers with different IP addresses and ports may result in different data transfer rates, suggesting that network configuration and routing can impact performance.

In the simpleperf, the results indicate that all 25 packets were transmitted and received successfully with no packet loss (0% packet loss). This means that the network connection between the source IP address and the destination IP address was stable during the time of the tests.

Based on the results, we can see that the RTT values vary for each IP address tested, but generally fall within a relatively small range. The average RTT values range from around 61 ms to 65 ms, with some occasional higher values. The minimum and maximum RTT values also vary, indicating fluctuations in network latency. The mean deviation (mdev) values, which measure the dispersion of the RTT values, are also relatively small, ranging from around 1 ms to 4 ms.

## **Test case 5: effects of parallel connections**

### **Results**

Throughput\_h1-h4.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.2, port ('10.0.0.2', 47348)

---------------------------------------------

ID Interval Transferred Rate

---------------------------------------------

A simpleperf client connecting to server 10.0.5.2, port ('10.0.0.2', 47344)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.2', 47344) [0.0 - 25.45] 36.72 11.54 MBps

('10.0.0.2', 47348) [0.0 - 25.65] 34.5 10.76 MBps

Throughput\_h2-h5.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.3, port ('10.0.0.3', 47790)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.3', 47790) [0.0 - 25.52] 69.15 21.68 MBps

Throughput\_h3-h6.txt

---------------------------------------------

A simpleperf client connecting to server 10.0.5.4, port ('10.0.0.4', 43376)

---------------------------------------------

ID Interval Transferred Rate

('10.0.0.4', 43376) [0.0 - 25.45] 69.93 21.98 MBps

### **Discussion**

for the connection between h1 and h4, the throughput is 36.72 MBps over a duration of 25.45 seconds, while for the connection between h2 and h5, the throughput is 34.5 MBps over a duration of 25.65 seconds.

This indicates that the use of parallel connections with the -P flag has resulted in higher data transfer rates (throughput) compared to normal client connections. The parallel connections allow for concurrent data transfers, which can potentially lead to higher overall throughput by utilizing the available network bandwidth more efficiently.

the actual impact of using parallel connections with the -P flag may vary depending on various factors such as network conditions, server load, and client capabilities. In some scenarios, the use of parallel connections may significantly improve throughput, while in others, it may have minimal or no impact.

the results suggest that using parallel connections with the -P flag in simpleperf client mode can be beneficial for achieving higher throughput in communication between h1 and h4 compared to normal client mode used by h2 and h3 for communication with h5 and h6.

# Conclusions

In conclusion, the provided code offers a simple performance testing tool for measuring network performance using Python's socket module. It enables data transfer between a client and server, measures the transfer rate in different units (B, KB, MB), and calculates the total transfer time. However, it may have limitations in accounting for real-world network conditions such as latency and packet loss. Further improvements could be made to enhance the accuracy and reliability of the network performance measurements.

# References (Optional)