APRIL 17, 2023

PORTFOLIO-1

SHATHA AMER
S362064@OSLOMET.NO
Oslo Metropolitan University

1 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.

2 Simpleperf

```python
import argparse
import socket     # Import the socket module to enable network communication
import time  # Import the time module to measure time intervals
import threading  # Import the threading module for concurrent execution
import sys  # Import the sys module for system-related operations

BUFFER_SIZE = 1000
# Dictionary to convert bytes to different formats (B, KB, MB)
bytesDict = {'B': 1, 'KB': 1000, 'MB': 1000000}

# Function to parse command line arguments

def parse_args():
    # Define an ArgumentParser object with a description of the arguments
    parser = argparse.ArgumentParser(description='simpleperf arguments')
```

```python
    parser.add_argument('-s', '--server', action='store_true',
                        help='enable the server mode')  # Add the -s or --server option to enable server mode
    parser.add_argument('-c', '--client', action='store_true',
                        help='enable the client mode')  # Add the -c or --client option to enable client mode
    parser.add_argument('-b', '--bind', default='127.0.0.1',
                        help='allows to select the ip address of the server’s interface where the client should connect')  # Add the -b or --bind option to select the IP address of the server's interface
    parser.add_argument('-I', '--serverAddr', default='127.0.0.1',
                        help='address of the server to connect to (required in client mode)')  # Add the -I or --serverAddr option to specify the server address
    parser.add_argument('-p', '--port', type=int, default=12000,
                        help='allows to use select port number on which the server should listen')  # Add the -p or --port option to specify the server's port number
    parser.add_argument('-f', '--format', choices=['B', 'KB', 'MB'], default='MB', help='to choose the format of the summary of results'
                        )  # Add the -f or --format option to specify the format of the results
    parser.add_argument('-P', '--parallel', type=int, default=1,
                        help='Number of parallel connections')  # Add the -P or --parallel option to specify the number of parallel connections to use during the transfer
    parser.add_argument('-t', '--time', type=int, default=25,
                        help='the total duration in seconds')  # Add the -t or --time option to specify the total duration of the operation in seconds
    parser.add_argument('-n', '--num', type=str,
                        help='Transfer number of bytes')  # Add the -n or --num option to specify the number of bytes
    parser.add_argument('-i', '--interval', type=int, default=1,
                        help='Interval for statistics')  # Add the -i or --interval option to specify the interval at which should be collected during the transfer
    # Parse the command-line arguments and store them in an object called "args"
    args = parser.parse_args()
    if args.format:  # If the -f or --format option was used
        # Check if the unit of the input value is valid
        if args.format not in ['B', 'KB', 'MB']:
            # Raise an error if the unit is invalid
            parser.error("the unit Should be B, KB, MB.")
    if args.interval <= 0:  # If the -i or --interval option is less than or equal to 0
        # an error
```

```python
        parser.error("Interval has a wrong values.")
    if not args.client and not args.server:
        parser.error('One of the client or the server is required.')
    return args  # Return the parsed arguments object

    # Function to print the results after data transfer

def print_summary(serverAddr, total_bytes, transfer_time, format):
    # Convert the size to the desired format and round to two decimal
    transfer_amount = round(total_bytes / bytesDict[format], 2)
    # Round the time to two decimal
    elapsed_time = round(transfer_time, 2)
    # Calculate the transfer rate in Mbps and round to two decimal
    if (elapsed_time > 0):
        transfer_rate = round(
            total_bytes * 8 / (elapsed_time * bytesDict[format]), 2)
        # Print the results
        print(
            f"{serverAddr}\t[0.0 - {elapsed_time}]\t{transfer_amount}\t{transfer_rate} {format}ps")
    else:
        print("the transfer_time is zero")

# Function to run the server

def server(args):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        # Bind the server to a specific address and port
        s.bind((args.bind, args.port))
        # Start listening
        s.listen(10)
        # Print the server header
        print("---------------------------------------------")
        print(f"A simpleperf server is listening on port {args.port}")
        print("---------------------------------------------")

        while True:  # Loop indefinitely
            conn, addr = s.accept()  # Accept a new client connection
            # Start a new thread
            threading.Thread(target=start_connection,
                             args=(conn, addr, args)).start()

def start_connection(conn, addr, args):
    with conn:
        print("---------------------------------------------")
        print(
            f"A simpleperf client connecting to server  {args.serverAddr}, port {args.port}")
```

```python
        # Print the client header
        print("---------------------------------------------")
        total_bytes = 0  # the total number of bytes received
        start_time = time.time()  # Start the clock to measure the time it takes to receive data
        while True:
            data = conn.recv(BUFFER_SIZE)  # Receive data from the client
            if "BYE" in data.decode():  # If no data is received, break the loop
                break
            total_bytes += len(data)  # Increase the number of bytes received
        # Stop the clock and calculate the time it took to receive the data
        elapsed_time = time.time() - start_time
        # Send a confirmation that all data has been received
        conn.sendall(b"ACK: BYE")
        conn.close()  # stop the connection to the client
        print(f"ID\tInterval\tReceived\tRate")  # Print results
        print_summary(addr, total_bytes, elapsed_time, args.format)
# Function to initiate the client mode

def client(args):
    # Create a TCP socket object
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as c:
        # Connect to the server
        try:
            c.connect((args.serverAddr, args.port))
        except ConnectionRefusedError:
            print("Error: Connection refused. Make sure the server is running and the IP address and port are correct.")
            sys.exit(1)
        # Print the client header
        print("---------------------------------------------")
        print(
            f"A simpleperf client connecting to server  {args.serverAddr}, port {c.getsockname()}")
        print("---------------------------------------------")
        print(f"ID\tInterval\tTransferred\tRate")  # Print results
        # Start the timer
        start_time = time.time()
        # the total number of bytes sent
        total_bytes = 0
        # Check if -n flag is provided
        if args.num:
            num_bytes = args.num
            numbers = ''
            letters = ''
```

```python
            for i in num_bytes:
                if i.isdigit():
                    numbers += i
                else:
                    letters += i
            get_format = letters.upper()
            if (get_format == 'B'):
                format = 'B'
            elif (get_format == 'K' or get_format == 'KB'):
                format = 'KB'
            elif (get_format == 'M' or get_format == 'MB'):
                format = 'MB'
            else:
                print('The format is not difined')
                sys.exit()
            # Convert input string to bytes
            num_bytes = bytesDict[format] * int(numbers)
            while total_bytes < num_bytes:
                # Generate some random data
                data = bytes(BUFFER_SIZE)
                # Send data in chunks of 1000 bytes
                c.sendall(data)
                total_bytes += len(data)
        elif args.interval:
            limit_time = args.time
            # Loop to send data
            interval = args.interval
            # the limited time for evry intervals
            interval_time = int(limit_time // interval)
            for i in range(interval):
                start_interval = i * interval_time
                end_interval = min((i + 1) * interval_time, limit_time)
                interval_bytes = 0
                while True:
                    if (time.time()-start_time >= end_interval):
                        break
                    # Generate some random data
                    data = bytes(BUFFER_SIZE)
                    # Send the data
                    c.sendall(data)
                    # Receive the response from the server and Convert the size to the desired format and round to two decimal places
                    total_bytes += len(data)
                    interval_bytes += len(data)
                format = args.format
                transfer_amount = round(interval_bytes / bytesDict[format], 2)
```

```python
                # Calculate the transfer rate and round to two decimal numbers
                transfer_rate = round(total_bytes * 8 / (end_interval - start_interval) /
                                      bytesDict[format], 2)
                if interval > 1:
                    # Print the results of the intervals
                    print(
                        f"{c.getsockname()}\t[{start_interval} - {end_interval}]\t{transfer_amount}\t{transfer_rate} {format}ps")

        # Send finish message to server
        c.send(b"BYE")
        # Check if the response is a BYE message or not
        response = c.recv(BUFFER_SIZE)
        if response != b"ACK: BYE":
            print("Error: Did not receive acknowledgement from server")
            # Close client socket
            c.close()
            return
        # Calculate and print total transfer time and bandwidth summary
        end_time = time.time()
        transfer_time = end_time - start_time
        if args.interval > 1:
            print("---------------------------------------------")
        print_summary(c.getsockname(), total_bytes, transfer_time, format)
        # Close and stop client socket
        c.close()

def start_threading(args):
    # Open the specified number of connections in parallel
    threads = []
    for i in range(args.parallel):
        t = threading.Thread(target=client, args=(args,))
        t.start()
        threads.append(t)

    # Wait for all threads
    for t in threads:
        t.join()

def main():
    args = parse_args()
    # Run the program in client or server mode, depending on the arguments which given.
    if args.client:
        start_threading(args)

```

```
    elif args.server:
        server(args)

if __name__ == '__main__':
    main()

```

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.

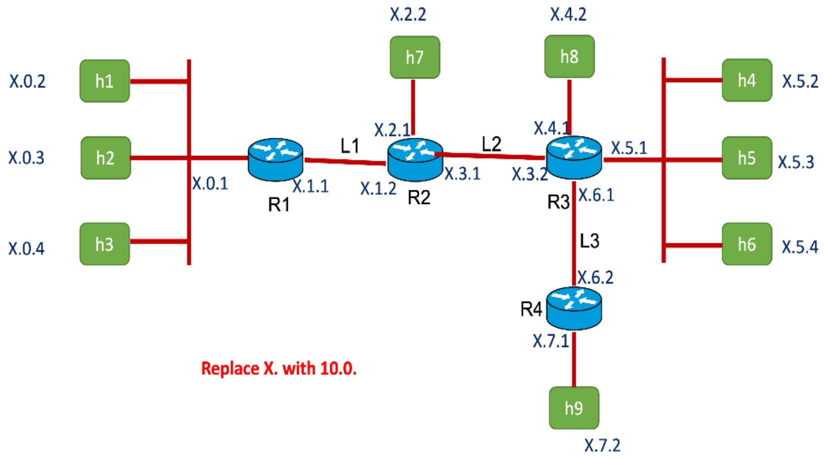
Client Function: The client() function implements the client mode functionality. It creates a TCP socket using the socket module, connects to the specified server address and port, sends data in chunks of BUFFER_SIZE bytes, calculates the total bytes sent, and measures the transfer time. It also prints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

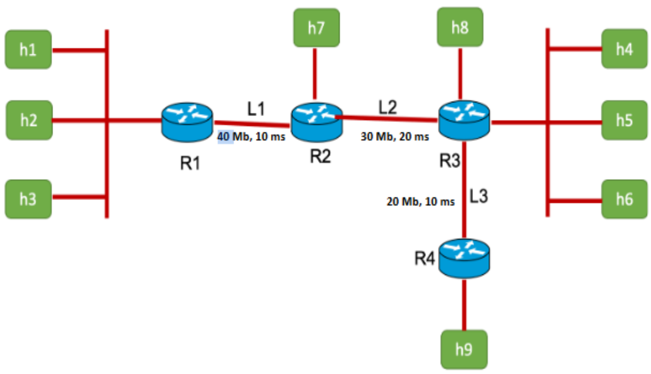
Main Code: The main code block calls the parse_args() function to parse the command-line arguments, and checks if the user has specified either client mode or server mode. If neither is specified, an error is raised. If server mode is specified, the server() function is called with the parsed arguments.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

230 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.





231 Performance evaluations

231.1 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.

231.2 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.

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

231.3.1 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.

231.4 Test case 2: link latency and throughput

231.4.1 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
```

231.4.2 Discussion

All 25 packets were received without any loss, indicating a 0% packet loss. A 0% packet loss indicates that all packets sent were successfully received without any loss, which is desirable for a healthy network connection.
In the 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.

1. L1:
Expected latency: 20 ms
Measured latency: 21.763 ms (average RTT)
Expected throughput: 40 MB
Measured throughput: 24.22 MB
2. L2:
Expected latency: 40 ms
Measured latency: 41.338ms (average RTT)
Expected throughput:30 MP
Measured throughput: 20.63 MB
3. L3:
Expected latency: 20 ms
Measured latency: 22.535 ms (average RTT)
Expected throughput: 20 MB
Measured throughput: 14.35 MB

231.5 Test case 3: path Latency and throughput

231.5.1 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
```

231.5.2 Discussion

1. h1 to h4:

Expected latency: 60 ms
Measured latency: 63.173ms (average RTT)
Measured throughput: 22.47 MBps

2. h7 to h9:

Expected latency: 60 ms
Measured latency: 63.399 ms (average RTT)
Measured throughput: 12.8 MBps

3. h1 to h9:

Expected latency: 80 ms
Measured latency: 85.180 ms (average RTT)
Measured throughput: 13.39 MBps

231.6 Test case 4: effects of multiplexing and latency

231.6.1 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
```

231.6.2 Discussion

When two pairs of hosts (h1-h4 and h2-h5) communicate at the same time, throughput and latency both suffer. Throughput is decreased as a result of the two pairs of hosts sharing the bandwidth. Since packets must wait before being transmitted, latency increases.

The simultaneous communication of three pairs of hosts (h1-h4, h2-h5, and h3-h6) causes further throughput and latency reductions. The throughput has now been further decreased as the bandwidth is split between three pairs of hosts.

A reduction in throughput and an increase in latency result from the simultaneous communication of two pairs of hosts (h1-h4 and h7-h9).
There is simultaneous communication between two pairs of hosts (h1-h4 and h8-h9), which lowers throughput and lengthens delay. Throughput has decreased because the bandwidth is now split between two host pairs on the same subnet.

231.7 Test case 5: effects of parallel connections

231.7.1 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
```

231.7.2 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.

232 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.

233 References (Optional)

Networkappers. (2023).*ping-toll.* https://networkappers.com/tools/ping-tool

Python Software Foundation. (2023, 16. Apr).*argpars.*

https://docs.python.org/3/library/argparse.html

W3schools. (2023). *Python.*https://www.w3schools.com/python

Iperf.fr.(2023).https://iperf.fr/