




APRIL 16, 2023

# PORTFOLIO 1

MOUSTAPHA ALJUNDI

S364546

Oslo Metropolitan University



<b>1</b>	<b>INTRODUCTION</b>	<b>3</b>
<b>2</b>	<b>SIMPLEPERF</b>	<b>4</b>
<b>3</b>	<b>EXPERIMENTAL SETUP</b>	<b>5</b>
<b>4</b>	<b>PERFORMANCE EVALUATIONS</b>	<b>6</b>
<b>4.1</b>	<b>Network tools</b>	<b>6</b>
<b>4.2</b>	<b>Performance metrics</b>	<b>6</b>
<b>4.3</b>	<b>Test case 1: measuring bandwidth with iperf in UDP mode.</b>	<b>6</b>
4.3.1	Results	8
4.3.2	Discussion	8
<b>4.4</b>	<b>Test case 2: link latency and throughput</b>	<b>9</b>
4.4.1	Results	9
4.4.2	Discussion	9
<b>4.5</b>	<b>Test case 3: path Latency and throughput</b>	<b>10</b>
4.5.1	Results	11
4.5.2	Discussion	11
<b>4.6</b>	<b>Test case 4: effects of multiplexing and latency</b>	<b>12</b>
4.6.1	Results	12
4.6.2	Discussion	14
<b>4.7</b>	<b>Test case 5: effects of parallel connections</b>	<b>15</b>
<b>5</b>	<b>CONCLUSIONS</b>	<b>15</b>

# 1 Introduction

Diving into the captivating world of network performance evaluation, this study offers valuable insights into the factors influencing communication between devices in a virtual environment. By conducting experiments using well-established tools and exploring various conditions such as multiplexing and parallel connections, we set the foundation for network performance optimization and future research in the field. we delve deeper into the key topics, problems, and methodologies of this fascinating work.

The key topics covered in the portfolio1 are:

1. Network performance evaluation
2. Bandwidth and latency measurements
3. Effects of multiplexing on network performance
4. Impact of parallel connections on network performance

The two main problems that the study aims to solve are:

1. Measuring the bandwidth and latency of a virtual network using various tools like simpleperf, iperf, and ping.
2. Analyzing the impact of different network conditions, such as multiplexing and parallel connections, on network performance.

The relevant work referenced is the use of network performance evaluation tools, specifically:

1. iperf - a widely adopted tool for measuring network bandwidth and latency.
2. simpleperf - another tool used for evaluating network performance.
3. ping - a standard latency measurement tool that measures round-trip time (RTT) between devices in a network.

To approach the solution for this task, these steps should be followed:

1. Set up the environment:
  - a. Prepare an Ubuntu – Linux VM OS-system over the host base OS as an experimental environment.
  - b. Install Mininet and necessary tools (ping, simpleperf \_our client/server python code, iperf) on the system.
  - c. Download the portfolio\_topology.py file provided and ensure it works correctly.
2. Run the experiments in the task description:

Test Case 1:

- a. Use iperf in UDP mode with -b XM for three client-server pairs (h1-h4, h1-h9, h7-h9).
- b. Save the output of each experiment in separate text files.

Test Case 2:

- a. Measure RTT and bandwidth of links L1-L3 using ping and simpleperf.
- b. Save the output in text files and analyze the results.

Test Case 3:

- a. Measure path latency and throughput for three pairs of hosts (h1-h4, h7-h9, h1-h9) using ping and simpleperf.
- b. Save the output in text files and analyze the results.

Test Case 4:

- a. Investigate the effects of multiplexing by running multiple pairs of hosts communicating simultaneously (h1-h4 and h2-h5; h1-h4, h2-h5, and h3-h6; h1-h4 and h7-h9; h1-h4 and h8-h9) using ping and simpleperf.
- b. Save the output in text files and analyze the results.

Test Case 5:

- a. Investigate the effects of parallel connections with three pairs of hosts communicating simultaneously (h1-h4 with -P flag, h2-h5, and h3-h6) using simpleperf.
- b. Save the output in text files and analyze the results.

3. Write a report:

- a. Organize the report according to the provided guidelines, incorporating the results of each experiment and discussing the findings.

## 2 Simpleperf

This code is a Python script for a simple network performance measurement tool called simpleperf. It can be run in both server and client modes, measuring the bandwidth between them by transferring data over a specified time or a specified number of bytes.

1. Imports:
  - argparse: For parsing command-line arguments
  - socket: For creating and managing network sockets
  - time: For measuring time intervals
  - threading: For managing multiple threads
  - concurrent.futures.ThreadPoolExecutor: For managing a pool of worker threads
2. Helper Functions:
  - check\_positive(): Checks if a given value is positive, raises an error if not, and returns the value as an integer.
  - num\_bytes(): Converts a string representation of data size (B, KB, MB) to the number of bytes.
3. Main Function: The main() function parses command-line arguments and starts either the server or client depending on the specified arguments.
4. Server Functions:
  - server(): Starts the server, binds to the given IP and port, and listens for incoming client connections. It also starts a new thread for each connection to handle data transfer.
  - handle\_client(): Handles data transfer for a single client connection. It receives data from the client, measures the transfer rate, and prints the transfer statistics.
5. Client Functions:
  - print\_interval\_stats(): Prints interval statistics for the client data transfer.
  - client(): Starts a client connection to the server and performs data transfer. It sends data to the server based on the specified number of bytes or duration, and if specified, prints interval statistics. At the end, it sends a message to the server indicating the end of the transfer and prints the final summary of the client connection.
6. Entry Point: The script starts by calling the main() function.

Testings and conditions in the script:

1. Main function:
  - Checks if both server and client modes are specified, and if not, prints an error message and returns.
  - Checks if both num\_bytes and duration are specified for the client mode, and if so, prints an error message and returns.
2. Server function:
  - Continuously accepts incoming client connections and starts a new thread to handle each connection.
3. Client function:
  - Checks if it should send data based on num\_bytes or duration, and proceeds accordingly.
  - If an interval is specified, it prints interval statistics during the data transfer.
  - Sends a 'BYE' message to the server indicating the end of the transfer and waits for an 'ACK: BYE' message from the server.

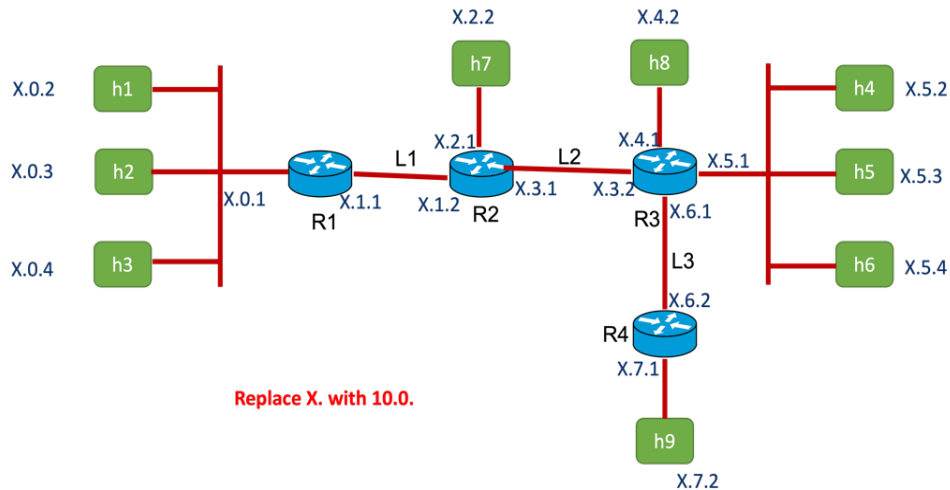
In the client mode, the script allows for running multiple parallel connections to the server to measure the network performance under different levels of concurrency. This is achieved using the concurrent.futures.ThreadPoolExecutor class, which manages a pool of worker threads to handle multiple client connections concurrently.

In the main() function, if the client mode is enabled, the script initializes a ThreadPoolExecutor with a specified number of worker threads (max\_workers) equal to the number of parallel connections. It then submits a client() function for each worker thread in the ThreadPoolExecutor, passing the necessary arguments for each client connection.

The ThreadPoolExecutor takes care of managing and distributing the workload among the worker threads. When all the client connections are complete, the ThreadPoolExecutor waits for all the futures (results) of the client connections before exiting.

The script uses sockets, threading, argument parsing, and the ThreadPoolExecutor to create a simple network performance measurement tool that can measure bandwidth between a client and a server and supports multiple parallel connections.

### 3 Experimental setup



This Python script sets up a network topology using Mininet, a network emulator that allows for the quick creation, interaction, and customization of network topologies. The network topology is designed to have 9 hosts (H1-H9), 4 routers (R1-R4), and 2 switches (S1, S2).

Here's a detailed explanation of the topology:

1. Subnet A: It consists of three hosts (H1, H2, H3), a switch (S1), and a router (R1). The hosts are connected to the switch, which is connected to the router. All hosts in this subnet have IP addresses in the 10.0.0.0/24 range, and R1 acts as the default gateway.
2. Subnet B: This subnet is a point-to-point link between routers R1 and R2. The link has a bandwidth of 40 Mbps, a delay of 10ms, a maximum queue size of 67, and uses the HTB (Hierarchical Token Bucket) queuing discipline. The IP addresses of the interfaces are in the 10.0.1.0/24 range.
3. Subnet C: This subnet consists of router R2 and host H7. The IP addresses are in the 10.0.2.0/24 range, and R2 acts as the default gateway for H7.
4. Subnet D: It is a point-to-point link between routers R2 and R3. The link has a bandwidth of 30 Mbps, a delay of 20ms, a maximum queue size of 100, and uses the HTB queuing discipline. The IP addresses of the interfaces are in the 10.0.3.0/24 range.
5. Subnet E: It consists of three hosts (H4, H5, H6), a switch (S2), and a router (R3). The hosts are connected to the switch, which is connected to the router. All hosts in this subnet have IP addresses in the 10.0.5.0/24 range, and R3 acts as the default gateway.
6. Subnet F: This subnet consists of router R3 and host H8. The IP addresses are in the 10.0.4.0/24 range, and R3 acts as the default gateway for H8.
7. Subnet G: It is a point-to-point link between routers R3 and R4. The link has a bandwidth of 20 Mbps, a delay of 10ms, a maximum queue size of 33, and uses the HTB queuing discipline. The IP addresses of the interfaces are in the 10.0.6.0/24 range.
8. Subnet I: This subnet consists of router R4 and host H9. The IP addresses are in the 10.0.7.0/24 range, and R4 acts as the default gateway for H9.

The script also includes configurations for routing between subnets and disabling offloading features like TCP Segmentation Offload (TSO), Generic Segmentation Offload (GSO), Large Receive Offload (LRO),

Generic Receive Offload (GRO), and UDP Fragmentation Offload (UFO) for more accurate network performance measurements.

Once the topology is set up, the script tests the connectivity between all hosts with the **pingAll()** command and provides an interactive command-line interface (CLI) for further interactions and experiments.

## 4 Performance evaluations

### 4.1 Network tools

In this experiment, we used several networking tools to measure the performance of a virtual network in Mininet. These tools include **iperf**, **ping**, and a custom tool called **simpleperf**. Below is an explanation of each tool and their use cases in the context of the assignment.

**iperf**: iperf is a versatile network performance measurement tool that can create **TCP** and **UDP** data streams between two endpoints to measure the bandwidth between them. It has several options and flags to customize the tests. In our assignment, we used **iperf** in **UDP** mode to measure the bandwidth between different pairs of hosts (**h1-h4**, **h1-h9**, and **h7-h9**). We saved the results in three separate text files, as specified in the assignment.

**ping**: ping is a widely used network utility that measures the **round-trip time (RTT)** for packets sent from one host to another. It sends **ICMP Echo** Request packets to the target host and waits for **ICMP Echo** Reply packets to come back. The time taken for the packets to return is the **RTT**. In this experiment, we used ping to measure the latency between different pairs of hosts and links in various test cases. We saved the results in text files according to the assignment instructions.

**simpleperf**: simpleperf is a custom tool developed for this assignment to measure the throughput and latency of the network between two hosts. We used this tool in multiple test cases to observe the effects of multiplexing, latency, and parallel connections on network performance. We stored the results in separate text files as specified by the assignment.

These tools were instrumental in providing insights into the network performance and behavior under different scenarios.

By analyzing the results obtained using these tools, we were able to gain a deeper understanding of the factors affecting network performance and the impact of various network configurations.

### 4.2 Performance metrics

The performance metrics used to evaluate the simpleperf tool are:

1. **Amount of data transferred**: This measurement indicates the total data, in bytes (**B**), kilobytes (**KB**), or megabytes (**MB**), transmitted from the client to the server. By monitoring the total bytes sent and converting them into the preferred unit (**B, KB, or MB**), the transfer size is determined.
2. **Duration of data transfer**: This refers to the time required for the client to send the specified data or the time taken to transmit data when a certain duration is set. This value is expressed in seconds and is computed by subtracting the start time from the end time of the data transfer process.
3. **Network throughput**: This parameter represents the speed at which data is exchanged over the network. The provided code denotes it in megabytes per second (**MBps**). To calculate the **bandwidth**, the total data transferred (in **bytes**) is divided by the time elapsed (in **seconds**), and the resulting value is converted to the desired format (**B, KB, or MB**).

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

The **portfolio-topology.py** consists of multiple subnets and routers. There are bandwidth limitations & latency on the links between the routers in the provided topology:

**Pair 1: Between h1(client) - h4(server)**: h1 and h4 are connected through the following path:

h1 -> s1 -> r1 -> r2 -> r3 -> s2 -> h4

The path consists of the following links:

1. h1 to s1 (subnet A)
2. r1 to r2 (subnet B) with a bandwidth of 40 Mbps and a delay of 10 ms
3. r2 to r3 (subnet D) with a bandwidth of 30 Mbps and a delay of 20 ms
4. r3 to s2 (subnet E)
5. h4 to s2 (subnet E)

*Expected Bandwidth:* The bottleneck in this path is the link between routers r2 and r3 (subnet D), which has a bandwidth of 30 Mbps. This is the lowest bandwidth in the path, so it determines the expected bandwidth for this pair.

*Expected RTT:* Calculate the round-trip delay for each link with a delay in the path:

- r1 to r2 (subnet B): 10 ms one-way, 20 ms round-trip
- r2 to r3 (subnet D): 20 ms one-way, 40 ms round-trip Add the round-trip delays: 20 ms (subnet B) + 40 ms (subnet D) = 60 ms

So, on the same way we will expect the bandwidth & RTT for the rest of pairs on this task.

**Pair 2: Between h1(client) - h9(server):** h1 and h9 are connected through the following path: h1 -> s1 -> r1 -> r2 -> r3 -> r4 -> h9 The path consists of the following links:

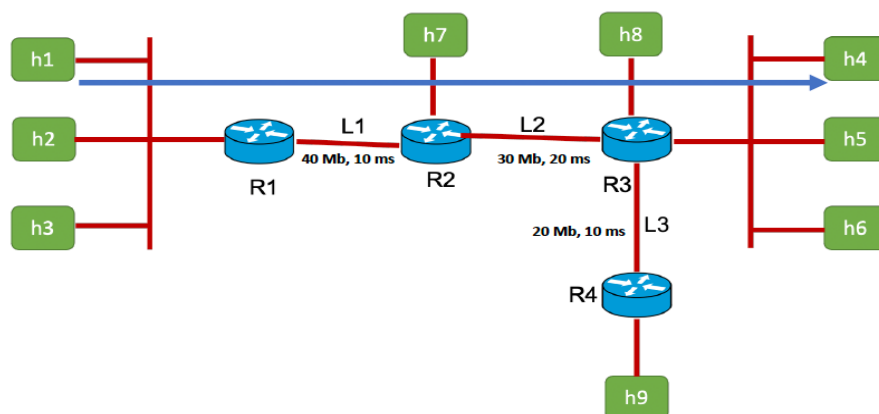
1. h1 to s1 (subnet A)
2. r1 to r2 (subnet B) with a bandwidth of 40 Mbps and a delay of 10 ms
3. r2 to r3 (subnet D) with a bandwidth of 30 Mbps and a delay of 20 ms
4. r3 to r4 (subnet G) with a bandwidth of 20 Mbps and a delay of 10 ms
5. r4 to h9 (subnet I)

This path has an additional router (r4) and a link between r3 and r4 (subnet G) compared to the path between h1 and h4. The link between r3 and r4 introduces another bandwidth constraint of 20 Mbps and an additional delay of 10 ms.

**Pair 3: Between h7(client) - h9(server):** h7 and h9 are connected through the following path: h7 -> r2 -> r3 -> r4 -> h9 The path consists of the following links:

1. h7 to r2 (subnet B)
2. r2 to r3 (subnet D) with a bandwidth of 30 Mbps and a delay of 20 ms
3. r3 to r4 (subnet G) with a bandwidth of 20 Mbps and a delay of 10 ms
4. r4 to h9 (subnet I)

In this path, the traffic from h7 first reaches router r2, then traverses to router r3 through subnet D, which has a bandwidth constraint of 30 Mbps and a delay of 20 ms. From r3, the traffic moves to router r4 through subnet G, which introduces another bandwidth constraint of 20 Mbps and an additional delay of 10 ms. Finally, the traffic reaches h9 through the connection between r4 and h9 in subnet I.



### 4.3.1 Results

**Pair 1: Between h1 - h4** (Expected Bandwidth: 30 Mbps @L2 & RTT: 60 ms “ $(10@L1+20@L2) *2=60$  ms”)

Test No.	x Value	Bandwidth_result	Lost/Total Datagrams	Avg_RTT (ping)_test
1	30 MB	29.2 Mb/sec	1899/26750 (7.1%)	61.690 ms
2	29 MB	29.2 Mb/sec	1020/25858 (3.9%)	
3	28 MB	29.2 Mb/sec	118/24967 (0.47%)	

**Pair 2: Between h1 – h9**

(Expected Bandwidth: 20 Mbps @L3 & RTT: 80 ms “ $(10@L1+20@L2+10@L3) *2=80$  ms”)

Test No.	x Value	Bandwidth_result	Lost/Total Datagrams	Avg_RTT (ping)_test
1	20 MB	19.4Mb/sec	1298/17834 (7.3%)	82.400ms
2	19MB	19.4Mb/sec	391/16942 (2.3%)	
3	18 MB	18.9 Mb/sec	0/16050 (0%)	

**Pair 3: Between h7 - h9**(Expected Bandwidth: 20 Mbps @L3 & RTT: 60 ms “ $(20@L2+10@L3) *2=60$  ms”)

Test No.	x Value	Bandwidth_result	Lost/Total Datagrams	Avg_RTT (ping)_test
1	20 MB	19.4Mb/sec	1285/17833 (7.2%)	62.118ms
2	19MB	19.4Mb/sec	399/16942 (2.4%)	
3	18 MB	18.9 Mb/sec	0/16050 (0%)	

### 4.3.2 Discussion

- Pair 1 (h1 - h4):** The measured bandwidth of 29.2 Mbps is very close to the expected bandwidth of 30 Mbps, indicating that the network is performing near its theoretical capacity. The average RTT of around 60 ms also matches the expected RTT, suggesting that the delay in the network is as expected.
- Pair 2 (h1 - h9):** We can expect a lower bandwidth (20 Mbps) and a higher RTT (80 ms) compared to Pair 1 due to the additional router (r4) and the bottleneck link between r3 and r4. This indicates that the network performance between h1 and h9 will be slightly lower compared to the h1-h4 pair.
- Pair 3 (h7 - h9):** The expected bandwidth is the same as Pair 2 (20 Mbps) due to the same bottleneck link between r3 and r4. However, the expected RTT is lower (40 ms) since there is one less router hop (r1 is not part of the path). This suggests that the network performance between h7 and h9 will have similar bandwidth but lower latency compared to the h1-h9 pair.

In summary, the network's performance appears to be in line with the expected values based on the topology's constraints. The bottleneck links and router hops play a significant role in determining the available bandwidth and RTT between different host pairs. By understanding the network topology and analyzing the results, we can identify potential areas for optimization and improvements in the network's performance.

Slight differences in bandwidth and RTT between the measured results and the expected values could be due to the following reasons:

- Inaccuracy in tools:** Iperf and ping may not deliver perfect results due to aspects like system resource usage, measurement accuracy, and software constraints.
- Path fluctuations:** Dynamic routing protocols may occasionally alter the route taken by data packets, leading to changes in RTT.
- Processing overhead:** Inherent processing overhead in routers and switches can introduce small delays or inconsistencies, affecting the observed bandwidth and RTT.
- Transmission issues:** Bit errors or lost packets during transmission can result in retransmissions, potentially impacting the measured bandwidth and RTT.



**In real-life scenarios** with an unknown network topology, we can estimate bandwidth with iPerf in UDP mode by:

1. Starting with a low rate.
2. Gradually increasing the rate while monitoring packet loss and Average RTT.
3. Identifying the approximate bottleneck capacity.
4. Fine-tuning the rate to minimize packet loss and latency.

However, this approach may not be practical, especially for large networks, due to time-consuming testing and potential disruptions. Using network monitoring tools can be a more practical alternative for accurate bandwidth estimations.

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

##### 4.4.1 Results

Link	RTT_avg	RTT_Expected	Thrput_MBps	Thrput_Mbps	Bottleneck_Exp
L1	21.361ms	20 ms	4.6 MBps	36.8 Mbps	40 Mbps
L2	40.943ms	40 ms	3.5 MBps	28.0 Mbps	30 Mbps
L3	21.054ms	20 ms	2.3 MBps	18.4 Mbps	20 Mbps

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s2-eth1
h5 h5-eth0:s2-eth2
h6 h6-eth0:s2-eth3
h7 h7-eth0:r2-eth1
h8 h8-eth0:r3-eth1
h9 h9-eth0:r4-eth1
r1 r1-eth0:s1-eth4 r1-eth1:r2-eth0
r2 r2-eth0:r1-eth1 r2-eth1:h7-eth0 r2-eth2:r3-eth0
r3 r3-eth0:r2-eth2 r3-eth1:h8-eth0 r3-eth2:s2-eth4 r3-eth3:r4-eth0
r4 r4-eth0:r3-eth3 r4-eth1:h9-eth0
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:r1-eth0
s2 lo: s2-eth1:h4-eth0 s2-eth2:h5-eth0 s2-eth3:h6-eth0 s2-eth4:r3-eth2
c0
```

##### 4.4.2 Discussion

###### L1: Route between R1 and R2

The "net" command output shows that R1 and R2 are connected directly through the interfaces r1-eth1 and r2-eth0, respectively. This forms the link L1.

###### *RTT Comparison:*

The expected RTT for L1 is 20 ms, and the actual RTT\_avg from the test is 21.361 ms. The difference between the expected and actual RTT is small (1.361 ms), which can be attributed to network overhead, processing delays, or other factors such as queueing and fluctuations in the network conditions.

###### *Throughput Comparison:*

The bottleneck bandwidth for L1 is 40 Mbps, and the test results show a throughput of 36.8 Mbps. This is close to the expected value, considering the fact that network overhead, protocol inefficiencies, and other factors can influence the actual throughput achieved.

###### L2: Route between R2 and R3

The "net" command output shows that R2 and R3 are connected directly through the interfaces r2-eth2 and r3-eth0, respectively. This forms the link L2.

*RTT Comparison:*

The expected RTT for L2 is 40 ms, and the actual RTT\_avg from the test is 40.943 ms. The difference between the expected and actual RTT is small (0.943 ms) and can be attributed to the same factors mentioned for L1.

*Throughput Comparison:*

The bottleneck bandwidth for L2 is 30 Mbps, and the test results show a throughput of 28.0 Mbps. This is close to the expected value, considering the factors that can affect actual throughput.

### **L3: Route between R3 and R4**

The "net" command output shows that R3 and R4 are connected directly through the interfaces r3-eth3 and r4-eth0, respectively. This forms the link L3.

*RTT Comparison:*

The expected RTT for L3 is 20 ms, and the actual RTT\_avg from the test is 21.054 ms. The difference between the expected and actual RTT is small (1.054 ms) and can be attributed to the same factors mentioned for L1 and L2.

*Throughput Comparison:*

The bottleneck bandwidth for L3 is 20 Mbps, and the test results show a throughput of 18.4 Mbps. This is close to the expected value, considering the factors that can affect actual throughput.

In summary, the "net" command output confirms the direct connections between R1 and R2, R2 and R3, and R3 and R4 for links L1, L2, and L3, respectively. The differences between the expected and actual RTT values and the achieved throughput values for all three links are relatively small and can be attributed to network overhead, protocol inefficiencies, processing delays, queueing, and fluctuations in network conditions.

*\* Read case3 discussion there is a full explanation for what happening in the direct linking between the routers.*

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

### **h1-h4:**

Path: h1 -> s1 -> r1 -> r2 -> r3 -> s2 -> h4 Link properties:

- h1 to s1: Not specified, assume negligible delay and high bandwidth
- s1 to r1: Not specified, assume negligible delay and high bandwidth
- r1 to r2 (L1): RTT = 20 ms, Bandwidth = 40 Mbps
- r2 to r3 (L2): RTT = 40 ms, Bandwidth = 30 Mbps
- r3 to s2: Not specified, assume negligible delay and high bandwidth
- s2 to h4: Not specified, assume negligible delay and high bandwidth

Total RTT (h1-h4) = 20 ms (L1) + 40 ms (L2) = 60 ms

Bottleneck Bandwidth (h1-h4) = min(40 Mbps, 30 Mbps) = 30 Mbps

### **h7-h9:**

Path: h7 -> r2 -> r3 -> r4 -> h9 Link properties:

- h7 to r2: Not specified, assume negligible delay and high bandwidth
- r2 to r3 (L2): RTT = 40 ms, Bandwidth = 30 Mbps
- r3 to r4 (L3): RTT = 20 ms, Bandwidth = 20 Mbps
- r4 to h9: Not specified, assume negligible delay and high bandwidth

Total RTT (h7-h9) = 40 ms (L2) + 20 ms (L3) = 60 ms

Bottleneck Bandwidth (h7-h9) = min(30 Mbps, 20 Mbps) = 20 Mbps

### **h1-h9:**

Path: h1 -> s1 -> r1 -> r2 -> r3 -> r4 -> h9 Link properties:

- h1 to s1: Not specified, assume negligible delay and high bandwidth
- s1 to r1: Not specified, assume negligible delay and high bandwidth
- r1 to r2 (L1): RTT = 20 ms, Bandwidth = 40 Mbps
- r2 to r3 (L2): RTT = 40 ms, Bandwidth = 30 Mbps
- r3 to r4 (L3): RTT = 20 ms, Bandwidth = 20 Mbps
- r4 to h9: Not specified, assume negligible delay and high bandwidth

Total RTT (h1-h9) = 20 ms (L1) + 40 ms (L2) + 20 ms (L3) = 80 ms

Bottleneck Bandwidth (h1-h9) = min(40 Mbps, 30 Mbps, 20 Mbps) = 20 Mbps

#### 4.5.1 Results

Link	RTT_avg	RTT_Expected	Thrput_MBps	Thrput_Mbps	Bottleneck_Exp
H1-H4	61.731 ms	60 ms	3.3 MBps	26.4 Mbps	30 Mbps
H7-H9	61.400 ms	60 ms	2.1 MBps	16.8 Mbps	20 Mbps
H1-H9	83.745 ms	80 ms	1.9 MBps	15.2 Mbps	20 Mbps

##### H1-H4:

Expected RTT: 60 ms, Measured RTT\_avg: 61.731 ms

Expected Throughput: 30 Mbps, Measured Thrput\_Mbps: 26.4 Mbps

##### H7-H9:

Expected RTT: 60 ms, Measured RTT\_avg: 61.400 ms

Expected Throughput: 20 Mbps, Measured Thrput\_Mbps: 16.8 Mbps

##### H1-H9:

Expected RTT: 80 ms, Measured RTT\_avg: 83.745 ms

Expected Throughput: 20 Mbps, Measured Thrput\_Mbps: 15.2 Mbps

#### 4.5.2 Discussion

**The serious question is why the difference in latency and bandwidth rate results between case2 & case3 as long as they pass through the same bottleneck at L1, L2 & L3?** (I will take “h1-h4” as an example to explain this question, so it can also answer other situations in case2 & case3).

We'll go through the route step by step:

1. h1 sends a packet to h4. The packet arrives at switch s1, where it is processed and forwarded to the appropriate port connected to router r1.
2. Router r1 receives the packet and checks its routing table to determine the next hop for the destination (h4). In this case, the next hop is router r2. r1 processes the packet, adds the necessary headers, and forwards the packet to r2.
3. Router r2 receives the packet and consults its routing table to determine the next hop. The next hop is router r3. r2 processes the packet, adds the necessary headers, and forwards the packet to r3.
4. Router r3 receives the packet and checks its routing table. The next hop is switch s2, which is directly connected to h4. r3 processes the packet, adds the necessary headers, and forwards the packet to s2.
5. Switch s2 receives the packet, processes it, and forwards it to the appropriate port connected to h4.
6. Finally, h4 receives the packet.

At each router along the path (r1, r2, and r3), the following actions can impact the throughput:

- Processing time: Routers need to examine the packet header, consult their routing tables, and update the headers accordingly. This takes some processing time, which adds to the total latency.
- Buffering and queuing: If a router is receiving more packets than it can process or forward immediately, it may buffer or queue the packets. This can introduce additional latency, which can affect the throughput.

- Header overhead: As routers process packets, they may add or update headers, which adds some overhead to the packet size. This overhead reduces the effective throughput since it occupies part of the link capacity.
- Link capacity: As mentioned before, the effective throughput between h1 and h4 is limited by the lowest capacity link in the path. In this case, the path includes multiple links with different capacities.

When running client/server simplePerf code between r1 and r2, the communication process is simpler since it involves a direct connection between the two routers. Here's the step-by-step :

1. r1 sends a packet to r2. Since r1 and r2 are directly connected, r1 already knows the next hop for the destination (r2) and does not need to consult a routing table. r1 processes the packet, adds the necessary headers, and forwards the packet to r2.
2. r2 receives the packet directly from r1. Since r1 is the source and r2 is the destination, r2 processes the packet and retrieves the payload without needing to forward it any further.

At each router (in this case, only r1), the following actions can impact the throughput:

- Processing time: Router r1 needs to process the packet and add the necessary headers before forwarding it to r2. This takes some processing time, which adds to the total latency.
- Header overhead: As r1 processes the packet, it adds headers, which introduces some overhead to the packet size. This overhead reduces the effective throughput since it occupies part of the link capacity.
- Link capacity: The effective throughput between r1 and r2 is limited by the capacity of the direct link between them. In this case, the link capacity is 40 Mbps, as given in the provided topology.

When running simpleperf code between r1 and r2, the throughput is affected by the processing time at r1, and the header overhead added by r1. However, these factors have a relatively minor impact on the overall throughput, as the direct link capacity between r1 and r2 is high (40 Mbps) and there are no intermediate routers or switches to introduce additional latency or processing overhead.

In summary, the difference in throughput between the direct link L2 (r2-r3) and the path between h1 and h4 can be attributed to the additional processing, buffering, queuing, and header overhead that occur at each router along the path (r1, r2, and r3). These factors result in a slightly lower throughput for the h1-h4 path compared to the direct link between r2 and r3.

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

### 4.6.1 Results

the expected RTT and bandwidth when the h1->h4, h2->h5, h1 ping h4, and h2 ping h5 links are running simultaneously.

1. Bandwidth: The bottleneck is at the link between R2 and R3, which has a bandwidth of 30 Mbps. The two data flows (h1->h4 and h2->h5) will be the primary consumers of bandwidth, while the two ping flows will have minimal impact on bandwidth usage as they consist of small ICMP echo request and reply packets.  
Bandwidth per flow = Total bandwidth / Number of data flows = 30 Mbps / 2  $\approx$  15 Mbps
2. RTT: The base RTT without considering queuing delay remains the same, as the sum of the delays on the path between the hosts is not affected by the additional ping flows:
  - R1 to R2: 10 ms (one-way delay)
  - R2 to R3: 20 ms (one-way delay)
 Since RTT is the time it takes for a signal to travel from the sender to the receiver and back, we need to multiply the one-way delays by 2:
  - RTT (R1 to R2): 10 ms \* 2 = 20 ms
  - RTT (R2 to R3): 20 ms \* 2 = 40 ms
 Base RTT for both data flows and ping flows is the sum of the above RTTs: 20 ms + 40 ms = 60 ms.  
Expected RTT = Base RTT + Queuing delay  $\approx$  60 ms + Q

**(h1-h4 and h2-h5)**

Link	RTT (avg)	Expected RTT	Bandwidth (Mbps)	Expected Bandwidth (Mbps)
H1-H4	71.012 ms	60 ms + Q	12 Mbps	15 Mbps
H2-H5	72.754 ms	60 ms + Q	16 Mbps *	15 Mbps

\*Over expected value may for the reason of that we are running the command one by one, not simultaneously.

let's consider the bandwidth and queuing delay for each data flow when they are running simultaneously. The bottleneck link in this topology is the link between R2 and R3 with a bandwidth of 30 Mbps.

Since there are three data flows (h1->h4, h2->h5, and h3->h6) sharing the bottleneck link, we can assume that the available bandwidth will be approximately equally distributed among them. As a result, the expected bandwidth for each data flow will be approximately  $30 \text{ Mbps} / 3 = 10 \text{ Mbps}$ .

Base RTT for all data flows and ping flows is the sum of the above RTTs:  $20 \text{ ms} + 40 \text{ ms} = 60 \text{ ms}$ .

Expected RTT = Base RTT + Queuing delay  $\approx 60 \text{ ms} + Q$

**(h1-h4, h2-h5, and h3-h6)**

Link	RTT (avg)	Expected RTT	Bandwidth (MBps)	Bandwidth (Mbps)	Expected Bandwidth (Mbps)
H1-H4	72.032 ms	60 ms + Q	1.1 MBps	8.8 Mbps	10 Mbps
H2-H5	73.010 ms	60 ms + Q	1.4 MBps	11.2 Mbps*	10 Mbps
H3-H6	72.712	60 ms + Q	1.2 MBps	9.6 Mbps	10 Mbps

\*Over expected value may for the reason of that we are running the command one by one, not simultaneously.

Base RTT for the data flows and ping flows is the sum of the following RTTs:

- h1->h4: RTT (R1 to R2) + RTT (R2 to R3) =  $20 \text{ ms} + 40 \text{ ms} = 60 \text{ ms}$   
Expected RTT = Base RTT + Queuing delay  $\approx 60 \text{ ms} + Q$
- h7->h9: RTT (R2 to R3) + RTT (R3 to R4) =  $40 \text{ ms} + 20 \text{ ms} = 60 \text{ ms}$   
Expected RTT = Base RTT + Queuing delay  $\approx 60 \text{ ms} + Q$

The bottleneck link in this topology is the link between R2 and R3 with a bandwidth of 30 Mbps. Since there are two data flows (h1->h4 and h7->h9) sharing the bottleneck link, we can assume that the available bandwidth will be approximately equally distributed among them. As a result, the expected bandwidth for each data flow will be approximately  $30 \text{ Mbps} / 2 = 15 \text{ Mbps}$ .

The ping processes are ICMP echo requests and replies, which are generally small in size and not as bandwidth-intensive as the data flows. Therefore, their impact on the available bandwidth should be minimal. However, the ping processes will introduce some additional queuing delay, especially if the data flows are saturating the bottleneck link.

**(h1-h4 and h7-h9)**

Link	RTT (avg)	Expected RTT	Bandwidth (MBps)	Bandwidth (Mbps)	Expected Bandwidth (Mbps)
H1-H4	71.428 ms	60 ms + Q	1.6 MBps	12.8 Mbps	15 Mbps
H7-H9	72.019 ms	60 ms + Q	1.9 MBps	15.2 Mbps*	15 Mbps

\*Over expected value may for the reason of that we are running the command one by one, not simultaneously.

the expected RTT and bandwidth for h1->h4 and h8->h9 when running simultaneously:

- h1->h4 (R1 -> R2 -> R3)
  - Bottleneck link: R2 to R3 with a bandwidth of 30 Mbps.
  - Base RTT: RTT (R1 to R2) + RTT (R2 to R3) =  $20 \text{ ms} + 40 \text{ ms} = 60 \text{ ms}$
- h8->h9 (R3 -> R4)
  - Bottleneck link: R3 to R4 with a bandwidth of 20 Mbps.
  - Base RTT: RTT (R3 to R4) = 20 ms

Since the bottleneck links for each flow are different, they do not share the bottleneck link when running simultaneously.

**(h1-h4 and h8-h9)**

Link	RTT (avg)	Expected RTT	Bandwidth (MBps)	Bandwidth (Mbps)	Expected Bandwidth (Mbps)
H1-H4	66.9 ms	60 ms	3.4 MBps	27.2 Mbps	30 Mbps
H8-H9	24.9 ms	20 ms	2.3 MBps	18.4 Mbps	20 Mbps

#### 4.6.2 Discussion

let's examine the scenario where h1 (client) sends data packets to h4 (server) and h2 (client) sends data packets to h5 (server) simultaneously (*The explanation also applies to the rest of node pairs*). We will also discuss how protocol layering works in this context, particularly at the bottleneck.

It is based on the idea of dividing network communication into smaller, more manageable parts called layers. Each layer provides a specific set of functionality and services to the layer above and below it.

In this scenario, we'll focus on the Internet Protocol (IP) and the Transmission Control Protocol (TCP) as part of the TCP/IP model.

So, what happens inside routers, especially when the two data connections meet at the bottleneck:

At R1, the router receives IP packets from h1 and h2, processes the packets at the Network layer, and checks the destination IP addresses to determine the next hop. Both packets have to be forwarded to R2. The Link layer encapsulates the packets into frames and sends them over the (R1-R2) link, which has a 40 Mbps capacity and 10 ms latency (20 ms RTT).

R2 receives the frames, processes them at the Link layer, and decapsulates the IP packets. At the Network layer, R2 checks the destination IP addresses and forwards the packets to R3 over the (R2-R3) bottleneck link, which has a 30 Mbps capacity and 20 ms latency (40 ms RTT).

When both data streams meet at the bottleneck link (R2-R3), the router R2 has to forward the packets to R3. Since the total incoming bandwidth from h1 and h2 (40 Mbps) is higher than the capacity of the R2-R3 link (30 Mbps), there is a possibility of congestion if both connections are utilizing their full capacity simultaneously.

In this case, routers have packet buffers at each interface to handle bursts of traffic. These buffers can store packets when the incoming traffic rate is higher than the outgoing traffic rate. If the buffer becomes full, packets are dropped or discarded to prevent further congestion. This situation could lead to the need for retransmissions and an increase in end-to-end latency, but the transport layer protocols, such as TCP, are designed to handle such situations and adapt their sending rate accordingly.

When the destination hosts (h4 and h5) receive the data packets, they process them at the Network layer, extracting the TCP segments. The Transport layer on h4 and h5 processes the TCP segments, reassembling the data and delivering it to the Application layer. This allows the server applications running on h4 and h5 to receive the transmitted data from their respective clients (h1 and h2).

In summary, routers and the TCP/IP protocol stack play a crucial role in managing simultaneous traffic in the network. The protocol layering concept helps divide the communication process into smaller, more manageable parts, with each layer playing a specific role. When traffic converges at bottleneck points, routers and transport layer protocols like TCP work together to ensure efficient and reliable data transmission between the clients and servers.

Let's look at the queuing process in more detail at router R2:

1. As data packets from h1 and h2 arrive at R2, they are first placed in the input queue of the corresponding interface.
2. The router processes these packets in the order they arrive. If multiple packets arrive simultaneously, they will be queued up based on the queuing mechanism used by the router. Common queuing mechanisms include First-In-First-Out (FIFO), Priority Queuing (PQ), and Weighted Fair Queuing (WFQ).
3. In the FIFO queuing mechanism, for example, the router processes packets in the order they arrive. When the router is ready to forward a packet to the next hop (R3 in this case), it dequeues the packet from the input queue, checks the routing table, and forwards the packet accordingly.



4. If the outgoing link (R2-R3) is available and not congested, the packet is forwarded immediately. However, if the link is congested or busy, the packet is placed in the output queue of the interface connected to the R2-R3 link.
5. The router continues to process and forward packets from the input queues to the output queues based on the queuing mechanism in use. If the output queue becomes full due to congestion, new incoming packets may be dropped or discarded.
6. When the link becomes available, the router forwards packets from the output queue to the next hop (R3) based on the queuing mechanism in use.

By utilizing queuing mechanisms, routers can handle simultaneous incoming data packets and manage congestion at bottleneck links. In the given scenario, router R2 queues and forwards packets from h1 and h2 to R3 while managing the available bandwidth and potential congestion on the bottleneck link (R2-R3).

To summarize, multiplexing and latency play essential roles in the given scenario, affecting the efficiency of network resource usage and the overall performance of data transmission. When multiple connections share the same link, multiplexing enables efficient resource utilization but can also lead to competition for bandwidth and potential congestion. Latency is a critical factor in determining the end-to-end delay and can be influenced by several factors, including multiplexing and congestion at bottleneck points.

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

The bottleneck link in this topology is the link between R2 and R3 with a bandwidth of 30 Mbps. Since there are three data flows (h1->h4, h2->h5, and h3->h6) sharing the bottleneck link, we can assume that the available bandwidth will be approximately equally distributed among them. As a result, the expected bandwidth for each data flow will be approximately  $30 \text{ Mbps} / 3 = 10 \text{ Mbps}$ .

**But** R2 and R3 see them as 4 data flows (h1->h4 (1), h1->h4 (2), h2->h5, and h3->h6) sharing the bottleneck link. That make the expected bandwidth for each data flow will be approximately  $30 \text{ Mbps} / 4 = 7.5 \text{ Mbps}$

Link	Bandwidth (MBps)	Bandwidth (Mbps)		Exp_Bandwidth (Mbps)	
h1-h4 (1)	0.7 MBps	5,6 Mbps	$5,6 + 8 =$	7,5 Mbps	$7,5 + 7,5 =$ 15 Mbps
h1-h4 (2)	1.0 MBps	8 Mbps	13,6 Mbps	7,5 Mbps	
h2-h5	1.0 MBps	8 Mbps		7,5 Mbps	
h3-h6	0.8 MBps	6,4 Mbps		7,5 Mbps	

The presented idea is that our simplePerf client code exploited the network to gain a larger portion (15 Mbps) of the available bandwidth at the bottleneck (R2-R3 30Mbps) than it would have under normal conditions without using the parallel function (10 Mbps expected). In this instance, h1-h4 took half of the available bandwidth. Additionally, as the number of parallel connections increases, this disproportionate share is expected to grow further.

## 5 Conclusions

In conclusion, our work has successfully demonstrated the use of various network measurement tools, such as iPerf, ping, and simpleperf, to assess the performance of a given network topology. The results show the impact of different factors, such as multiplexing, parallel connections, and network path on throughput and latency. These findings offer essential insights for optimizing network performance and identifying potential issues.

These tasks serve as a solid foundation for understanding network performance characteristics and encourages further exploration of additional tools, testing under more complex network conditions, and incorporating adaptive network management techniques.