APRIL 17, 2023

NETWORKING AND CLOUD COMPUTING DATA2410

PORTFOLIO1

MOHAMMAD KHIR KHALED ALMOHAMMAD
S343988
Oslo Metropolitan University

1 Introduction

This work provides a practical understanding of how different network configurations, traffic patterns and communication scenarios impact key network performance metrics such as bandwidth, latency and throughput. By conducting experiments in a controlled virtual environment, it enables the exploration of various network setups and the identification of potential bottlenecks and limitations.

The key topics in this task are:

- Network performance evaluation: Assessing the performance of a virtual network by measuring crucial metrics such as bandwidth, latency and throughput.
- Network tools: Utilizing well-established network measurement tools like ping, iperf, and simpleperf to perform the experiments and obtain quantitative results.
- Network configurations and traffic patterns: Investigating the effects of different factors such as multiplexing, parallel connections, and network topology on network performance.
- Virtual network emulation: Using Mininet as the platform to create and test the virtual network, which allows for the controlled experimentation of various network scenarios.

The problems being addressed in this task are:

- Understanding the performance implications of different network configurations and traffic patterns on key network metrics such as bandwidth, latency and throughput.
- Identifying bottlenecks or limitations in the network that could affect the efficiency and reliability of communication between hosts.
- Evaluating the effects of multiplexing and parallel connections on network performance, which can have significant implications in real-world networking scenarios.
- Gaining practical insights into using network measurement tools like ping, iperf and simpleperf to assess the performance of a virtual network in Mininet, an emulation platform widely used for network experimentation and research

Relevant works for this task include:

- iPerf: A popular network performance measurement tool used to evaluate bandwidth and throughput in both TCP and UDP modes.
- Ping: A standard network utility for measuring Round-Trip Time (RTT) between devices providing a simple method for assessing network latency.
- Simpleperf: A tool for measuring network throughput, which is closely related to bandwidth. It can help analyze the amount of data transmitted over the network within a given time frame.
- Mininet: A widely-used network emulation tool that enables researchers to create, test and experiment with virtual networks.

Approach to the solution involves the following steps:

- Set up the virtual network: Create a virtual network in Mininet using the provided topology, which consists of nine hosts (h1 to h9) and four routers (r1 to r4).
- Conduct experiments using network measurement tools: Use ping, iperf and simpleperf to perform a series of experiments designed to evaluate network performance in various scenarios. These scenarios include measuring bandwidth and latency for different client-

server pairs, investigating the effects of multiplexing and parallel connections, and exploring the impact of network configurations and traffic patterns on performance metrics.

- Analyze the results: Collect the output data from each experiment and analyze the findings. Calculate key metrics such as average RTT, measured throughput, and bandwidth and compare the results across different test cases to identify patterns and trends.
- Discuss the implications: Interpret the results in the context of real-world networking scenarios and evaluate the practical implications of the findings. Identify potential bottlenecks, limitations, and opportunities for improving network performance.
- Document the process and findings: Prepare a comprehensive report detailing the experimental setup, test cases, results and discussion.

2 Simpleperf

Simpleperf is a network performance assessment tool that enables users to evaluate network throughput and data transfer rates between a server and a client. It comprises several components that manage argument parsing, server and client mode execution, data size parsing and validation, and thread administration for multiple simultaneous connections. The communication between the server and client includes sending data packets, determining data rates, and signaling the completion of data transmission using distinct messages.

Simpleperf operates in two modes, server and client. The server mode listens on a specific IP address and port number, while the client mode establishes a connection to a specified server IP and port. In client mode, users can configure the data size or duration of transmission and the number of parallel connections to the server. Data transmission between the server and client uses data chunks and Simpleperf calculates data rates based on the received data.

In essence, Simpleperf is a utility for measuring network performance between a server and a client. It contains several functional units responsible for various tasks such as input parsing, server-client interactions, and thread management for concurrent connections. The communication between the two parties involves exchanging data segments, estimating transfer rates, and using unique message codes('BYE and 'ACK: BYE') to indicate the end of data transmission.

The implementation of Simpleperf can be broken down into the following building blocks:

Argument parsing: The **create_arg_parser()** function defines a set of command-line arguments that allow users to configure Simpleperfs behavior, such as operating in server or client mode, setting IP addresses and port numbers, choosing the data size specifying the transmission time and setting the number of parallel connections.

Server mode: The **server_mode()** function sets up a server that listens on a specified IP address and port number. It calls the **accept_client()** function in a loop, which accepts incoming client connections and creates a new thread for each client using the **process_client()** function. The **process_client()** function receives data from clients, measures the amount of data received, and calculates the data rate.

Client mode: The **client_mode()** function establishes a connection to a specified server IP address and port number. It sends data to the server according to the specified data size or duration, and optionally displays interval statistics using the **display_interval_stats()** function. After sending data, the client sends a 'BYE' message to the server to signal the end of the data transmission and waits for an 'ACK: BYE' response from the server.

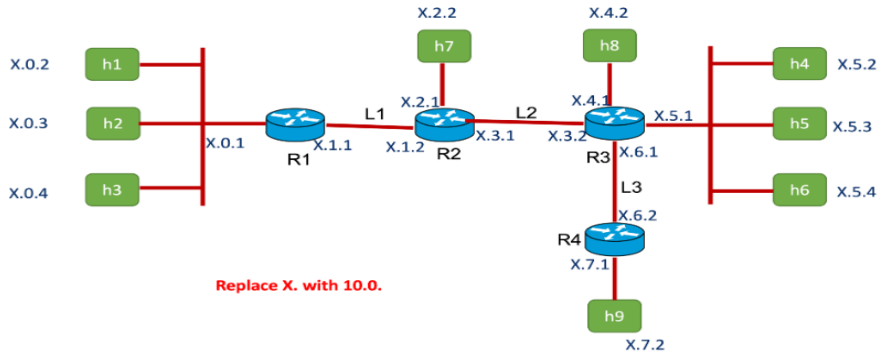
Thread pool management: In client mode, when multiple parallel connections are specified a **ThreadPoolExecutor** is used to manage the threads for each connection. The client threads are created by submitting the **client_mode()** function with their corresponding connection IDs.

Data size parsing and validation: The **parse_data_size()** and **validate_positive()** functions help parse and validate input values for data size and positive integers, respectively.

Main function: The **simpleperf_main()** function is responsible for orchestrating the overall flow of the script. It processes the command-line arguments and decides whether to run the script in server mode or client mode. In client mode it also validates the input for data size and duration before proceeding.

Entry point: The **run_simpleperf()** function acts as the entry point of the script. It creates an argument parser, parses the command-line arguments, and calls the **simpleperf_main()** function.

3 Experimental setup



The portfolio-topology is a virtual network created using Mininet a network emulator to evaluate the Simpleperf tool. The network consists of multiple subnets, interconnected routers, and hosts. The topology is designed to simulate real-life network conditions with varying bandwidth, delay, and queue sizes. Here's overview of the topology:

Subnet A: Consists of three hosts (H1, H2, and H3), a switch (S1) and a router (R1). The hosts are connected to the switch, which is connected to the router. Hosts H1, H2, and H3 have IP addresses in the 10.0.0._ subnet.
Subnet B: Connects router R1 to router R2 with a link (L1) having a bandwidth of 40 Mbps and a 10ms delay.
Subnet C: Connects router R2 to host H7. H7 has an IP address in the 10.0.2._ subnet.
Subnet D: Connects router R2 to router R3 with a link (L2) having a bandwidth of 30 Mbps and a20ms delay.
Subnet E: Connects router R3 to hosts H4, H5, and H6 through switch S2. Hosts H4, H5, and H6 have IP addresses in the 10.0.5._ subnet.
Subnet F: Connects router R3 to host H8. H8 has an IP address in the 10.0.4._ subnet.
subnet G: Connects router R3 to router R4 with a link (L3) having a bandwidth of 20 Mbps and a 10ms delay.
Subnet I: Connects router R4 to host H9. H9 has an IP address in the 10.0.7._ subnet.

This topology utilizes various layers of the protocol stack, which include the Application, Transport, Network, Link, and Physical layers. The Mininet network simulator used in the topology creates a virtual network environment, that emulates the behavior of the protocol stack. In this topology:

Application Layer: The hosts in the topology can run various applications, but the script itself does not explicitly define any application layer protocols.
Transport Layer: The hosts can utilize transport layer protocols such as TCP or UDP to facilitate communication between applications.
Network Layer: The topology uses IP addresses to facilitate communication between hosts in different subnets. Routers (R1, R2, R3, and R4) are configured to forward packets based on IP addresses, and routing rules are set to establish the correct paths for packet delivery.
Link Layer: In this layer, switches (s1 and s2) are used to forward data between hosts within the same subnet based on MAC addresses. The topology also uses the LinuxRouter class to enable IP forwarding on the routers, allowing them to communicate between subnets.
Physical Layer: The Mininet network simulator emulates the physical layer by creating virtual links between nodes. In this topology, the links are created using the addLink method and some links have specified bandwidth, delay, and queue size properties, which emulate the characteristics of physical connections.

4 Performance evaluations

4.1 Network tools

I am going to use the following tools:

• **ping**: Ping is a basic network diagnostic tool that tests the connectivity between two network devices, typically a sender and a receiver. The tool works by sending a series of small data packets, called Internet Control Message Protocol (ICMP) Echo Requests, from the sender to the receiver. If the receiver is reachable and operational, it responds with ICMP Echo Reply packets. Ping measures the Round-Trip Time (RTT) which is the time it takes for a packet to travel from the sender to the receiver and back.

• **iPerf**: iPerf is a widely-used network performance measurement tool that evaluates the throughput and other performance metrics of a network connection. Unlike Ping, iPerf uses Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) to measure the achievable bandwidth between two endpoints. iPerf operates in a client-server model. where a server is set up on one end of the connection, and a client on the other end sends data to the server. By using iPerf, network administrators can assess the maximum achievable data transfer rate, packet loss, jitter, and other important network performance characteristics.

• **simpleperf:** I have explained it earlier.

4.2 Performance metrics

performance metrics that i use to evaluate meg simpleperf tool.

Throughput:
Throughput is the actual amount of data transferred through a network or network connection in a given time period also measured in (bps). Throughput can vary depending on several factors, such as network load, signal strength, packet loss, error rate, and the quality of the network components. Unlike bandwidth which is a theoretical limit, throughput is the actual performance achieved under real conditions.

Here it is worth mentioning the difference between bandwidth and throughput
Bandwidth:
Bandwidth refers to the maximum data transfer capacity of a network or network connection.measured in bits per second (bps). it is a theoretical value that indicates how much data can be transferred simultaneously under ideal conditions. Bandwidth often used to describe the speed of Internet connections, and it is common to refer to it as "bandwidth" even though it is actually the data transfer capacity.

The difference between bandwidth and throughput lies in how they describe network performance. Bandwidth indicates the theoretical maximum capacity, while throughput refers to the actual amount of data transferred under given conditions. It is common for throughput to be lower than bandwidth as factors such as network load and packet loss can affect performance.

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

I have measured the Throughput between h1 (client) and h4 (server) using commands:
On h4 (server): iperf -s -u
On h1 (client): iperf -c <h4-ip> -u -b XM
And to find the RTT between h1-h2, I used Ping <h4-ip> -c 25
I did the same for h1-h9 and h7-h9

4.3.1 Results

Below are the results of the tests conducted between the specified hosts. The results include the average Round Trip Time (RTT) and the throughput for each test.

Test	Interval (us)	Transfer (MBytes)	Troughput (Mbits/sec)	Jitter (ms)	Lost/Total Datagrams	RTT Measured (ms)
h1-h4	560.76	24.9	20.9	0.858	0.29%	75.820
h1-h9	862.71	16.2	13.6	0.936	0.06%	93.798
h7-h9	934.60	15.0	12.5	1.472	0.24%	75.429

The table above shows that the connection between h1-h4 has the highest throughput and the lowest average RTT, followed by the connections between h1-h9 and h7-h9, respectively.

4.3.2 Discussion

To measure the bandwidth between h1 and h4, I should choose a rate higher than the bottleneck bandwidth along the path. In the given topology, the path from h1 to h4 is: h1 -> s1 -> r1 -> r2 -> r3 -> s2 -> h4. There are two links with defined bandwidth constraints along this path: r1-r2 (40 Mbps) and r2-r3 (30 Mbps). The bottleneck bandwidth is the minimum of these, which is 30 Mbps. So, I should choose a rate higher than 30 Mbps.

Similarly, the path from h1 to h9 is: h1 -> s1 -> r1 -> r2 -> r3 -> r4 -> h9. There are three links with defined bandwidth constraints: r1-r2 (40 Mbps), r2-r3 (30 Mbps), and r3-r4 (20 Mbps). The bottleneck bandwidth is the minimum of these, which is 20 Mbps. So, I should choose a rate higher than 20 Mbps.

the path from h7 to h9 is: h7 -> r2 -> r3 -> r4 -> h9. There are two links with defined bandwidth constraints: r2-r3 (30 Mbps) and r3-r4 (20 Mbps). The bottleneck bandwidth is the minimum of these, which is 20 Mbps. So, I should choose a rate higher than 20 Mbps.

My test results e.g. for h1-h4 indicate that the highest measured bandwidth is 20.9 Mbits/sec, which is lower than the expected 30 Mbps. The reasons for this deviation can be:

- **Packet loss**: I can see from the test results that there is a significant percentage of packet loss at higher sending rates. Packet loss can be due to the router buffers being overloaded or other network factors, leading to lower effective bandwidth.
- **Jitter**: The variation in packet arrival times (jitter) can also affect the measured bandwidth. In my test results, jitter values are relatively low, but they still might impact the measurement accuracy.
- **Network conditions**: The tests were conducted over a specific period, and network conditions might have changed during the tests, leading to lower bandwidth measurements than expected.
- **Implementation or configuration factors**: Factors like buffer sizes, algorithms used by the iperf tool, or the configuration of the devices in the network may also impact the measurements.

To evaluate bandwidth using iPerf in UDP mode on an unknown network topology, follow these steps:
- Choose a reasonable packet size and test length.
- Start with a low bandwidth rate and gradually increase it.
- Monitor packet loss and latency.
- Identify the saturation threshold. Continue to increase the bandwidth until I observe a substantial increase in packet loss or latency
- Conduct bidirectional tests. Execute tests in both directions.

This process can be time-consuming and impractical for large networks.

4.4 Test case 2: link latency and throughput

I have measured RTT and bandwidth of each of the three individual links between routers (L1 - L3) . Let's consider L1 (r1-r2). using commands:

- RTT with ping
  Ping r2 from r1: Ping <r2-eth0> -c 25
- Throughput with simpleperf
  Server: run server on r2 : Python3 simpleperf –s -b <r2-eth0> –p 8088
  Client (r1): Python simpleperf –c –I <r2-eth0> –p 8088 –t 25

And repeated it for L2 and L3.

4.4.1 Results

Link L1 (R1-R2):
Measured RTT (average): 33.4ms
Measured Throughput: 10.7 Mbps

Link L2 (R2-R3):
Measured RTT (average): 47.692ms
Measured Throughput: 10.3 Mbps

Link L3 (R3-R4):
Measured RTT (average): 24.754ms
Measured Throughput: 9.6 Mbps

4.4.2 Discussion

When measuring the RTT and Throughput, the results will depend on the configured links between routers, hosts, and switches. The link configurations are as follows:

Link L1 (R1-R2): Bandwidth = 40 Mbps, Delay = 10 ms, Max Queue Size = 67
Link L2 (R2-R3): Bandwidth = 30 Mbps, Delay = 20 ms, Max Queue Size = 100
Link L3 (R3-R4): Bandwidth = 20 Mbps, Delay = 10 ms, Max Queue Size = 33
The expected results for each of the individual links between routers (L1, L2, and L3) should be as follows:

Link L1 (R1-R2):
RTT: Approximately 20 ms (10 ms delay * 2)
Bandwidth: 40 Mbps

Link L2 (R2 -R3):
RTT: Approximately 40 ms (20 ms delay * 2)
Bandwidth: 30 Mbps

Link L3 (R3-R4):
RTT: Approximately 20 ms (10 ms delay * 2)
Bandwidth: 20 Mbps

This table shows the summary of results:

Link	RTT Expected (ms)	RTT Measured (ms)	Expected Throughput (Mbps)	Measured Throughput (Mbps)
L1	20	33.4	40	10.7
L2	40	47.692	30	10.3
L3	20	24.754	20	9.6

There can be multiple explanations for the discrepancy between the expected results and the measured observed results:

Buffering and queuing: Additional delays introduced by the network devices (switches and routers) queuing and buffering mechanisms, which can impact the RTT and bandwidth measurements. This is particularly true when network devices are emulated or their configurations do not align with the anticipated settings.

Network emulation: Utilizing a single computer, Mininet simulates networks, which may not deliver the same level of performance as dedicated hardware. The Mininet performance is reliant on the resources available on the host system, including processing power, memory, and input /output capabilities. The host machine's resource limitations may affect the measured data.

Additional processes: The host system may be executing other tasks that utilize system resources, impacting the Mininet networks performance. This may result in higher latency and reduced throughput compared to the expected values.

Protocol overhead: The anticipated results do not consider the overhead created by the TCP/IP protocol stack. Factors such as retransmissions, congestion control, and flow control can contribute to this overhead, influencing the Round Trip Time (RTT) and bandwidth.

Testing instruments: The network performance measurement tools, like ping and simpleperf.py, could introduce their own overhead and inaccuracies, affecting the observed results.

4.5 Test case 3: path Latency and throughput

I have measured RTT and Throughput between of each of the h1-h4, h1-h9 and h7-h9 . Let's consider h1-h4. I used commands:

- RTT with ping
  Ping h4 from h1: Ping 10.0.5.2 -c 25
- Throughput with simpleperf
  Server: run server on h4 : Python3 simpleperf –s -b 10.0.5.2 –p 8088
  Client (h1): Python simpleperf –c –I 10.0.5.2 –p 8088 –t 25

And repeated it for h1-h9 and h7-h9.

4.5.1 Results

h1-h4:

Measured RTT (average): 75.820 ms
Measured Throughput: 14.7 Mbps

h1-h9:

Measured RTT (average): 93.798 ms
Measured Throughput: 8.5 Mbps

h7-h9:

Measured RTT (average): 75.429 ms (average)
Measured Throughput: 9.5 Mbps

4.5.2 Discussion

h1-h4:

To determine the expected latency and throughput of the path between h1 and h4, we need to analyze the portfolio-topology and identify the links and their properties along the path. The path between h1 and h4 goes through the following links:

h1 -> r1: Through switch s1 (no properties mentioned)
r1 -> r2: Latency = 10ms, Bandwidth = 40Mbps, max_queue_size = 67
r2 -> r3: Latency = 20ms, Bandwidth = 30Mbps, max_queue_size = 100
r3 -> h4: Through switch s2 (no properties mentioned)
Total latency is the sum of the latencies of individual links:
Total latency = 10ms + 20ms = 30ms ➔ RTT = 30ms * 2 = 60ms

The throughput is determined by the bottleneck link along the path, which is the link with the lowest bandwidth:
Throughput = min(40Mbps, 30Mbps) = 30Mbps

So, the expected latency between h1 and h4 is 60ms, and the expected throughput is 30Mbps.

we repeat it for h1-h9 and h7-h9

h1-h9:

The expected latency = 10ms + 20ms + 10ms = 40ms ➔ RTT = 40ms * 2 = 80ms
The expected throughput = min(40Mbps, 30Mbps, 20Mbps) = 20Mbps

h7-h9

The expected latency = 20ms + 10ms = 30ms ➔ RTT = 30ms * 2 = 60ms
The expected throughput = min(30Mbps, 20Mbps) = 20Mbps

This table shows the differences between the expected values and the measured values for latency (RTT) and throughput between the host pairs

Path	RTT Expected (ms)	RTT Measured (ms)	Throughput Expected (Mbps)	Throughput Measured (Mbps)
h1-h4	60	75.820	30	14.7
h1-h9	80	93.798	20	8.5
h7-h9	60	75.429	20	9.5

Comparing the expected results with the measured results, we can observe that the actual latencies are slightly higher than the expected values. This difference could be due to network conditions or other factors that were not accounted for in the initial estimations.
Regarding throughput, the measured values are significantly lower than the expected values. This discrepancy might be due to factors such as network congestion, protocol overhead, or limitations in the testing setup (e.g the "simpleperf.py" itself or the devices used for testing)

4.6 Test case 4: effects of multiplexing and latency

In this experiment, i will look at the effects of multiplexing where many hosts will simultaneously communicate. And I will make a comparison between the expected results and the measured results. I used here two tools: ping to measure RTT and simpleperf to measure Throughput.

4.6.1 Results

Test Case	Host Pair	RTT (ms)	Throughput (Mbps)
1	h1-h4	160.444	11.2
1	h2-h5	134.952	7.7
2	h1-h4	136.009	8.3
2	h2-h5	122.190	4.2
2	h3-h6	121.316	7.2
3	h1-h4	157.152	10.5
3	h7-h9	148.714	5.6
4	h1-h4	132.222	14.9
4	h8-h9	76.032	11.5

4.6.2 Discussion

The expected latency is the sum of all delays across the path between the pair of hosts. Because the link propagation delays in the path are constant and do not change based on the number of communicating pairs.
The expected throughput is the minimum of the link capacities along the path between the pair of hosts, divided by the number of simultaneous connections sharing the bottleneck link.

Test Case	Host Pair	RTT Expected (ms)	RTT Measured (ms)	Throughput Expected (Mbps)	Throughput Measured (Mbps)
1	h1-h4	60	160.4	15	11.2
1	h2-h5	60	134.9	15	7.7
2	h1-h4	60	136.0	10	8.3
2	h2-h5	60	122.1	10	4.2
2	H3-h6	60	121.3	10	7.2
3	h1-h4	60	157.1	15	10.5
3	h7-h9	60	148.7	15	5.6
4	h1-h4	60	132.2	30	14.9
4	h8-h9	20	76.0	20	11.5

We note that the measured results varied significantly from the expected results, and this is due to several factors:

Resource contention: Multiplexing(Mux) in the transport layer enables multiple application processes on a single host to share a single transport layer connection. As a result multiple data streams may compete for the available bandwidth, potentially leading to increased latency and reduced throughput.

Demultiplexing overhead: Demultiplexing(Demux) at the receiving end can introduce additional processing overhead, as the transport layer needs to identify the appropriate application process for each incoming packet. This additional overhead may contribute to the increased latency and reduced throughput observed in the test results.

Buffering and queuing: When multiple data streams share a transport layer connection, the transport layer protocol may need to buffer and queue data packets, causing additional latency as packets wait to be transmitted or received. This can result in higher RTTs and lower bandwidth than expected.

Protocol inefficiencies: The transport layer protocols themselves, such as TCP or UDP, might have inefficiencies in their multiplexing and demultiplexing processes. For example, TCPs flow control and congestion control mechanisms may not efficiently handle the traffic patterns of multiple data streams sharing a connection, leading to suboptimal performance.

Synchronization issues: If multiple data streams are multiplexed together and experience similar patterns of traffic load, this can lead to a phenomenon called "global synchronization," where the streams' transmission rates become correlated. This can cause increased latency and reduced throughput as the streams collectively compete for bandwidth.

4.7 Test case 5: effects of parallel connections

In this experiment, h1, h2 and h3 connected to R1 will simultaneously talk to hosts (h4, h5 and h6) connected to R3. h1 will open two parallel connections (with -P flag in the client mode) to communicate with h4, and h2 and h3 will just invoke the normal client (without -P flag).
I used one tool here: simpleperf to measure Throughput.
In this experiment, I will look at the effects of parallel connections. And I will make a comparison between the expected results and the measured results.

4.7.1 Results

h1-h4:
6.4 Mbps for (Client 1) **&** 4.4 Mbps for (Client 2)
h2-h5:
3.1 Mbps
h3-h6:
6.0 Mbps

4.7.2 Discussion

First we need to consider the link capacities and any potential bottlenecks.
-Link L1 between R1 and R2 has a bandwidth of 40 Mbps and a delay of 10ms.
-Link L2 between R2 and R3 has a bandwidth of 30 Mbps and a delay of 20ms.

Since link L2 has the lowest bandwidth (30 Mbps) among the links in the path between R1 and R3, it will be the bottleneck for the communication between hosts h1, h2, h3 and hosts h4, h5, h6.

Now lets analyze the expected throughput for each pair:

h1-h4: h1 opens two parallel connections to communicate with h4. Given that the bottleneck link has a capacity of 30 Mbps, the total available bandwidth will be distributed among the three connections, with two connections for h1-h4 and one each for h2-h5 and h3-h6. So, the expected throughput for h1-h4 will be (30 Mbps * 2/4) = 15 Mbps. thus 7.5 for each client.

h2-h5: As there is a single connection between h2 and h5, the expected throughput for h2-h5 will be (30 Mbps * 1/4) = 7.5 Mbps.

h3-h6: Similarly for h3-h6, the expected throughput will also be (30 Mbps * 1/4) = 7.5 Mbps .

This table compares the expected throughput values with the measured throughput values for each host pair.

Host Pair	Expected Throughput (Mbps)	Measured Throughput (Mbps)
h1-h4 (parallel connections)	7.5 (Client 1) 7.5 (Client 2)	6.4 (Client 1) 4.4 (Client 2)
h2-h5 (single connection)	7.5	3.1
h3-h6(single connection)	7.5	6.0

The variation between the expected and reality results in my experiment can be attributed to both the effects of parallel connections and the limitations of the "simpleperf.py" code .

Effects of parallel connections:

- Contention: With multiple parallel connections sharing the same network link, there is contention for the available bandwidth. This can lead to reduced throughput for individual connections as they compete for resources.
- Flow control mechanisms: Transport protocols, such as TCP implement flow control mechanisms to adapt to the available bandwidth. In the presence of multiple parallel connections, these mechanisms can cause the throughput to be distributed unevenly among the connections, affecting the overall results.
- Bufferbloat: Parallel connections can lead to increased buffer occupancy in network devices which can cause increased latency and reduced throughput.

Limitations of "simpleperf.py" code:

- Fixed buffer size: The script uses a fixed buffer size of 1000 bytes, which might not efficiently utilize the available bandwidth. This can impact the throughput results especially in parallel connections.
- Use of threads: The script uses threads to manage multiple connections. Depending on the systems performance and the number of concurrent connections, the use of threads may introduce overhead that affects the throughput results.
- Python GIL: The Global Interpreter Lock (GIL) in Python can limit the performance of multi-threaded programs, which can lead to reduced throughput when using multiple parallel connections.

5 Conclusions

My work resulted in valuable insights into the impact of various network configurations, traffic patterns and communication scenarios on key performance metrics such as bandwidth, latency and throughput. I identified potential bottlenecks and limitations as well as opportunities for optimization in different network setups.

However, there were some shortcomings and limitations as for example :

The experiments were conducted in Mininet, a virtual network emulator which may not fully replicate the complexities and unpredictability of real-world networks. This could potentially limit the generalizability of my findings.
Experiments involving simultaneous communication between multiple hosts may have faced minor variations in starting times potentially introducing inconsistencies in the results.

6 References (Optional)

Kurose, J. F., & Ross, K. W. (2022). Computer Networking: A Top-Down Approach (8th ed.). Pearson Education.