APRIL 17, 2023

SIMPLEPERF REPORT

PORTFOLIO-1

ALI FARIZ ALGHADBAN
S362111
Oslo Metropolitan University

1 INTRODUCTION 2

2 SIMPLEPERF 2

3 EXPERIMENTAL SETUP 2

4 RESULTS AND DISCUSSION ERROR! BOOKMARK NOT DEFINED.

4.1 Network tools 3

4.2 Performance metrics 3

4.3 Test case 1: measuring bandwidth with iperf in UDP mode 4
4.3.1 Results 4
4.3.2 Discussion 4

4.4 Test case 2: link latency and throughput 4
4.4.1 Results 4
4.4.2 Discussion 5

4.5 Test case 3: path Latency and throughput 5
4.5.1 Results 5
4.5.2 Discussion 5

4.6 Test case 4: effects of multiplexing and latency 6
4.6.1 Results 6
4.6.2 Discussion 6

4.7 Test case 5: effects of parallel connections 7
4.7.1 Results 7
4.7.2 Discussion 7

5 CONCLUSIONS 8

6 REFERENCES 8

1 Introduction

Network performance is a critical aspect of modern computing systems, as it directly impacts the user experience, the efficiency of applications, and the overall effectiveness of networked systems. In this project, I aim to provide a comprehensive analysis of network performance by creating a simplified version of the popular network measurement tool iPerf, called simpleperf, and using it in various test cases in a Mininet virtual network environment.
The key topics addressed in this project are network throughput, latency, multiplexing, and the effects of parallel connections. I aim to solve the problem of accurately measuring network performance parameters, such as latency and throughput, under various network conditions and configurations. My work is built upon existing network performance measurement tools, such as iPerf, and the Mininet virtual network environment.
Me approach to the solution involves designing and implementing simpleperf using sockets, and then executing a series of test cases that involve different network configurations and scenarios. The tests include measuring bandwidth with iPerf in UDP mode, link latency and throughput, path latency and throughput, effects of multiplexing and latency, and effects of parallel connections. I collect the output of these experiments, analyze the results, and discuss the implications of my findings.
The limitations of my work include the fact that I am using a simplified version of iPerf, which may not have all the features and optimizations of the full version. Additionally, my experiments are conducted in a virtual network environment, which may not exactly replicate real-world network conditions.

2 Simpleperf

This code implements a simple network performance testing tool called "simpleperf" that works in both server and client mode. The main building blocks of the code include command line argument parsing, the server function, the client function, and helper functions for data manipulation and statistics calculation. Below is an overview of each component and the communication process between the server and client.
Command Line Argument Parsing: The main function parses the command line arguments using the argparse library. It defines several arguments to configure the behavior of simpleperf, such as server/client mode, IP addresses, ports, format, number of bytes to transfer, duration, interval for displaying statistics, and the number of parallel connections.
Server Function: The server function creates a socket and binds it to the specified IP address and port. It then listens for incoming client connections. When a connection is accepted, a new thread is created to handle the client connection using the "handle_client" function.
Handle Client Function: The handle_client function takes the client connection, its address, and the desired data format as input. It receives data from the client and keeps track of the total amount of data received. If the client sends a "BYE" message, the server sends an "ACK: BYE" message and terminates the connection. Finally, the function calculates the bandwidth and displays the connection statistics.
Client Function: The client function takes the server's IP address, port, number of bytes to transfer, duration, format, interval, and connection ID as input. It creates a socket, connects to the server, and sends an initial "START" message. The client then sends data to the server based on the specified number of bytes or duration. The client function also checks for and prints interval statistics during data transfer using the "print_connection_interval_stats" helper function.
Helper Functions:

check_positive: Checks if a given value is positive.
num_bytes: Converts a given string to bytes based on the specified format (B, KB, or MB).
print_connection_interval_stats: Prints connection interval statistics during data transfer. The function calculates the transfer and bandwidth during the interval and displays the information in the specified format.
Communication between Server and Client:
The client connects to the server and sends an initial "START" message with the start time.
The client sends data packets to the server based on the specified number of bytes or duration.
The server receives data packets and calculates the total received data and bandwidth.
The client sends a "BYE" message to the server when the transfer is complete.
The server sends an "ACK: BYE" message to the client to confirm the end of the connection.
Both the server and client display the connection statistics at the end of the transfer.

3 Experimental setup

The network topology for evaluating the tool consists of nine hosts, four routers, and two switches. Hosts H1 to H3, switch S1, and router R1 form one subnet, while H4 to H6, switch S2, and router R3 form another subnet. Router R2 connects to H7, and router R4 connects to H9. The routers are connected through subnets with specified bandwidth, delay, and maximum queue sizes.
The topology is built using Mininet, a network emulator. The script creates custom classes, sets up the network, and disables offloaded features for all nodes. The network is tested with a ping test and the Mininet CLI for further evaluation.

4 Performance evaluations

4.1 Network tools

Ping: Ping is a basic network utility that checks the reachability of a host on an Internet Protocol (IP) network. It works by sending Internet Control Message Protocol (ICMP) echo request packets to the target host and waiting for an ICMP echo reply. By measuring the time it takes for the echo request and reply to be exchanged, ping can provide an estimate of the round-trip time (RTT) and whether the target host is reachable or not.

iPerf: iPerf is a more advanced networking tool used to measure the bandwidth and the performance of TCP and UDP data streams between two network nodes. It can run in both client and server modes, with the server listening for incoming connections and the client sending data to the server. iPerf allows users to set various parameters, such as the duration of the test, the packet size, and the number of parallel connections. The tool provides detailed information on the throughput, packet loss, and jitter experienced during the test.

Simpleperf: Simpleperf is a custom Python script that combines elements of both Ping and iPerf. It allows users to run in either client or server mode and supports measuring the bandwidth of a connection. It can also display connection statistics at specified intervals and supports various data formats (B, KB, MB). While it is not as feature-rich as iPerf, Simpleperf provides a lightweight and simple alternative for basic network performance testing.

4.2 Performance metrics

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

4.3.1 Results

After assessing experiment outcomes, it's clear that selecting an 28 Mbps between h1 and h4, 18 Mbps between h1 and h9 and 18 Mbps between h7 and h9 is optimal for gauging the bandwidth. This rate minimizes data loss, delivering a stable link between both hosts.

To gauge bandwidth with iPerf (UDP) without knowledge of the network structure, initiate a low bandwidth and slowly raise it. Monitor packet loss and jitter to pinpoint the rate preceding excessive values. This method offers a ballpark estimate but may be impractical due to time consumption and possible effects on other users' network experience.

4.3.2 Discussion

I expect 30 MB between h1 and h4, 20 MB between h1 and h9 and 20 MB between h7 and h9.
The discrepancy between expected and observed bandwidths may be due to link capacity sharing, TCP congestion control, and queuing/buffer management in the network. These factors can cause the actual bandwidth to be different from the expected values.

4.4 Test case 2: link latency and throughput

4.4.1 Results

Explaing how I ecpected the throughput
Throughput: Throughput depends on the link capacity (bandwidth) and the number of connections sharing that bandwidth. Here's a quick summary of the links' bandwidths in the topology:
R1 to R2: 40 Mbps
R2 to R3: 30 Mbps
R3 to R4: 20 Mbps
Since the bottleneck link determines the maximum throughput, you should analyze the path between the source and destination hosts and identify the bottleneck link. Then, divide the bottleneck link's bandwidth by the number of connections sharing that bandwidth to get a rough estimate of the throughput.
For example, if you want to estimate the throughput between H1 and H4, the path would be H1-R1-R2-R3-H4. The bottleneck link in this path is R2-R3, which has a bandwidth of 30 Mbps. If there are two parallel connections (e.g., H1-H4 and H2-H5),

you can roughly estimate the throughput for H1-H4 to be 15 Mbps, assuming both connections consume equal bandwidth.
Round Trip Time (RTT): RTT is the time it takes for a packet to travel from the source to the destination and back. To estimate the RTT, you need to consider the propagation delays and the queuing delays at each link along the path.
In this topology, the link delays are:
R1 to R2: 10ms
R2 to R3: 20ms
R3 to R4: 10ms
For the path H1-R1-R2-R3-H4, the total one-way delay would be 10ms (R1-R2) + 20ms (R2-R3) = 30ms. The RTT would be double the one-way delay, so the estimated RTT between H1 and H4 would be 60ms.

	RTT_exepted	RTT_reality	Throughput_expected	Throughput_reality
L1	20	22.088	40	32
L2	40	43.465	30	24
L3	20	22.276	20	16

4.4.2 Discussion

The observed RTT values are slightly higher than the expected values. This could be due to various factors like additional processing overhead at the routers, link congestion, or other network conditions.
The observed throughput values are lower than the expected values. This can be attributed to multiple factors, including the processing overhead at routers, or limitations in the testing tools.

4.5 Test case 3: path Latency and throughput

4.5.1 Results

	RTT_rxpected	RTT_reality	Throughput_expected	Throughput_reality
H1-h4	60	66.802	30	15.2
H1-h9	80	90.356	20	14.4
H7-h9	60	65.339	20	8.8

4.5.2 Discussion

The differences between the expected and actual values can be attributed to several factors:
Network conditions: The actual network conditions, such as congestion, packet loss, or jitter, can impact the RTT and throughput values. These factors can result in retransmissions or delays, leading to lower throughput and higher latency than expected.
Resource contention: The hosts and routers involved in the communication might be competing for resources such as CPU, memory, or bandwidth. This contention can lead to decreased throughput and increased latency.
Measurement tools: The tools used for the measurements, such as ping and simpleperf, may introduce some overhead or variability in the results. These tools might not always provide an accurate reflection of the network's actual performance.

Routing inefficiencies: The routing of packets through the network might not be optimal, resulting in longer paths and additional latency. This can be especially relevant when comparing the expected and actual RTT values.

4.6 Test case 4: effects of multiplexing and latency

4.6.1 Results

	RTT_expected	RTT_reality	Throughput_expected	Throughput_reality
H1-h4	60	65.159	15	13.6
H2-h5	60	68.295	15	11.2
H1-h4	60	69.77	10	6.4
H2-h5	60	67.735	10	10.4
H3-h6	60	65.619	10	8
H1-h4	60	66.802	15	15.2
H7-h9	60	65.339	15	8.8
H1-h4	60	67.005	30	23.2
H8-h9	20	22.738	20	16

4.6.2 Discussion

The actual RTT for H1-H4 is 65.159 ms and for H2-H5 is 68.295 ms. Both values are slightly higher than the expected RTT. This is likely due to additional processing delays and queuing times at the routers as both pairs of hosts are communicating simultaneously, leading to increased contention and congestion.
The actual throughput for H1-H4 is 13.6 Mbps, and for H2-H5 is 11.2 Mbps. The measured throughput values are lower than the expected throughput. This can be attributed to the increased RTT, which causes the TCP congestion window to grow slower, leading to suboptimal utilization of the available bandwidth. Additionally, congestion and packet drops at the routers can also contribute to the lower throughput.

The actual RTT for H1-H4 is 69.77 ms, for H2-H5 is 67.735 ms, and for H3-H6 is 65.619 ms. All three values are higher than the expected RTT. The increase in RTT compared to the previous experiment is due to even higher contention and congestion caused by the additional pair of hosts communicating simultaneously. This results in longer processing delays and queuing times at the routers.
The actual throughput for H1-H4 is 6.4 Mbps, for H2-H5 is 10.4 Mbps, and for H3-H6 is 8 Mbps. The measured throughput values deviate from the expected throughput due to the increased RTT and higher congestion in the network. This causes the TCP congestion window to grow slower and the suboptimal utilization of the available bandwidth. Moreover, the uneven distribution of throughput among the pairs of hosts can be attributed to factors like the variability in packet processing times, queue management, and packet drops at the routers.

The actual RTT for H1-H4 is 66.802 ms and for H7-H9 is 65.339 ms. Both values are slightly higher than the expected RTT. The increase in RTT compared to the expected values could be due to a few factors such as small amounts of contention, processing delays, or queuing times at the routers, but the effect is significantly less than when the pairs share the same bottleneck link.
The actual throughput for H1-H4 is 15.2 Mbps and for H7-H9 is 8.8 Mbps. While H1-H4's throughput is close to the expected value, H7-H9's throughput is significantly lower. This discrepancy could be due to factors like a different bottleneck link in the path between H7 and H9, which could have a lower bandwidth than the link between R2 and R3. It is also possible that H7-H9 are experiencing higher contention or congestion in their path or that the TCP congestion window growth is affected by other factors.

The actual RTT for H1-H4 is 67.005 ms, and for H8-H9, it is 22.738 ms. Both values are higher than the expected RTT, but the difference is relatively small. The increase in RTT compared to the expected values could be due to factors such as contention, processing delays, or queuing times at the routers.
The actual throughput for H1-H4 is 23.2 Mbps, and for H8-H9, it is 16 Mbps. Both pairs of hosts achieved lower throughput than expected. This could be due to various factors, such as the TCP congestion control mechanism reacting to the small increase in RTT, or the routers handling simultaneous connections leading to queuing and packet drops. It's also possible that H8-H9 share a bottleneck link with other traffic, which could affect their throughput.

In summary, the results across all experiments highlight the impact of multiplexing on latency and throughput in a network with shared resources. As the number of pairs of hosts competing for these resources increases, the effects become more pronounced. The actual RTT and throughput values generally deviate from the expected values due to factors such as increased contention, congestion, and processing delays in the shared network paths. The deviation is less significant when the pairs of hosts communicate over separate paths with different bottleneck links, but the effects of multiplexing can still be observed, as seen in the experiments with H1-H4 and H7-H9, or H1-H4 and H8-H9.

4.7 Test case 5: effects of parallel connections

4.7.1 Results

	Throughput_expected	Throughput_reality
H1-h4	7.5/7.5	5.6/7.2
H2-h5	7.5	6.4
H3-h6	7.5	6.4

4.7.2 Discussion

The deviations in the throughput can be attributed to a few factors:
Resource contention: Since h1, h2, and h3 are all connected to R1 and communicating simultaneously, they are competing for the same resources (e.g., CPU, memory, and bandwidth). This contention can lead to reduced throughput for all connections, as we see in your results.

Parallel connections overhead: h1 established two parallel connections with h4, which increases the complexity of managing the connections on both sides. This overhead can impact the throughput of both connections, resulting in the lower values observed in the test.
Network latency and packet loss: The performance of the connections can be influenced by network factors such as latency and packet loss, which can lead to retransmissions and reduced throughput.

5 Conclusions

In this study, I investigated the performance of a network topology consisting of nine hosts, four routers, and two switches. I employed various network testing tools, including Ping, iPerf, and Simpleperf, to evaluate the performance of the network in terms of latency, throughput, and the effects of multiplexing and parallel connections.

My experimental results demonstrate that network latency and throughput can deviate from expected values due to factors such as resource contention, congestion, and processing delays. I also observed that increased multiplexing and the number of parallel connections can have a significant impact on network performance, leading to increased latency and reduced throughput.

The limitations of this study include the use of synthetic testing tools that may not accurately represent real-world network conditions and the inability to control all variables that can affect network performance. Additionally, the topology used in this study is relatively simple and may not fully capture the complexity of larger and more diverse networks.

6 References (Optional)