Wifi Lab-Performance Test

Antonello Di Pede

1 INTRODUCTION

The goal of this lab is to evaluate the performance of different network setups by measuring the actual goodput and comparing it to theoretical predictions. Goodput, the amount of useful data transferred over time, is a key indicator of network efficiency. Using **iperf3**, we performed controlled tests across various configurations to assess how closely real results match theoretical models.
We analyzed three scenarios involving a client and a server: both connected via Wi-Fi 6 (IEEE 802.11ax), both via Ethernet using Cat 5E UTP cables, and a mixed case with the server on Ethernet and the client on Wi-Fi. These setups allowed us to observe differences across wireless, wired, and hybrid environments. By comparing theoretical formulas taught in class with real measurements, we aimed to highlight discrepancies and identify factors such as protocol overhead or interference that may impact performance.

2 TOOLS AND THEORY

To better understand and compare how different network setups affect performance, we relied on two main tools: iperf3 and Wireshark. These tools helped us carry out real-time measurements and take a closer look at how data flows through the network in various scenarios. iperf3 is an open-source utility that's commonly used for testing network speed and measuring throughput. It works by setting up a connection between two devices—one acting as a server and the other as a client. The server side is launched using the command **iperf3 -s**, which opens the service on port 5201 by default. On the other end, the client initiates the test using **iperf3 -c <server IP address>**. In our case, both devices ran Kali Linux, a Debian-based distribution that offered all the needed tools out of the box and ensured compatibility across our setup. To explore both TCP and UDP protocols, we adjusted our iperf3 tests accordingly. While TCP runs by default, we enabled UDP using the **-u** flag when needed. Another useful option was **-b**, which allowed us to set a custom bandwidth value—especially crucial for UDP, which otherwise defaults to just 1 Mbps. By configuring both protocols to use higher bitrates, we aimed to push each network configuration to its capacity and evaluate the real goodput in practical terms. Alongside iperf3, we made use of Wireshark, a well-known tool for packet analysis. It helped us monitor the traffic generated during testing and provided insights at a more detailed level. With it, we were able to generate graphs showing throughput over time and track the progression of TCP sequence numbers throughout the tests. These visual tools were especially helpful for spotting patterns or unexpected behavior, like retransmissions or performance drops caused by interference. On the theoretical side, we calculated the expected goodput by starting from the maximum link speed and accounting for the overhead introduced by the different layers of the protocol stack such as Ethernet, IP, and transport layer headers. We also took into account factors such as retransmissions and acknowledgment delays, which can significantly affect performance, especially in wireless environments. Unlike Ethernet technology, which can operate in either full-duplex or half-duplex mode, allowing simultaneous transmission and reception over separate channels, Wi-Fi operates strictly in half-duplex mode, since both transmission and reception share the same channel. This constraint introduces interference and forces devices to take turns when communicating. For instance, when a client sends a packet to a server via an access point, it must wait for the access point to finish transmitting before it can send the next packet. As a result, the maximum theoretical goodput is effectively halved. Before diving into the experimental results, we first provide the theoretical background used to estimate the expected goodput. Below, we present the formula used for the goodput calculation, followed by the specifications of the devices involved and the nominal link capacities for both Ethernet and Wi-Fi connections. These values form the basis for our theoretical predictions in each scenario.

$$\text{Goodput: } G = \frac{\text{Useful data at the Application Layer}}{\text{Time to complete the transfer}}$$

There are several components that can reduce the goodput:

- **Shared physical link** with other protocols and transmissions
- **Errors**: Bit error, collisions, congestion events and packet drops that do not deliver useful data, but that consume time
- **Protocol overhead**

So, the theoretical maximum throughput or maximum goodput will be calculated as:

$$\text{Goodput G } < \eta_{\text{protocol}} C$$

3 LAB SETUP AND SCENARIOS

The following devices and equipment were used throughout the experiments:

- Access Point: Technicolor DGA4331TIM
- Server: MSI PULSE GL66 11UEK-040IT with 802.11ax Wi-Fi 6 support
- Client: ASUS TUF Dash F15 with 802.11ax Wi-Fi 6 support
- Ethernet Cable: UTP Cat 5E with bandwidth up to 1 Gbps

3.1 Link Capacities

Ethernet: 1Gbps using a Cat 5e Ethernet and also the FTTH (Fiber-To-The-Home) internet connection provided by the ISP had a maximum capacity of 1 Gbps.
Wi-Fi: While Wi-Fi 6 (IEEE 802.11ax) has a theoretical maximum of 9.6 Gbps, the actual bitrate observed during our tests—retrieved using the `iwlist wlan0 bitrate` command—was approximately between 960 Mbps and 1000 Mbps. Both devices were connected to the 5 GHz band, specifically channel 5.32 GHz, to maximize performance and reduce interference compared to the 2.4 GHz band.

3.2 Network configuration

Parameter	Client	Server
Operating System	Kali Linux (Debian-based)	Kali Linux (Debian-based)
IP Address	192.168.1.211	192.168.1.*
Subnet Mask	255.255.255.0	255.255.255.0
Default Gateway	192.168.1.1	192.168.1.1
Interface	wlan0 or eth0	wlan0 or eth0

Note: The server has various IP addresses because the experiments were conducted on separate days and the DHCP protocol assigned different addresses.
List of all server addresses: .250 .189. .125 .59 .88

3.3 Automation Using a Python Script

To thoroughly assess the network performance, we crafted a Python script that automates the iperf3 tests. For every scenario and for both TCP and UDP, the script executes 10 separate runs. It then computes key statistics, such as the minimum, maximum, average, and standard deviation of the goodput measurements, to ensure our results are robust and meaningful. This method enables us to effectively compare the experimental data with our theoretical predictions.
The complete Python script is available in the Appendix (Figure 12)

4 RESULTS

4.1 Scenario 1: WiFi-WiFi TCP

In this scenario, both the transmitter and the receiver are connected via a Wi-Fi link in infrastructure mode. Data is transmitted from the source to the Access Point (AP) and then from the AP to the destination using the same channel, which effectively halves the end-to-end throughput.
The theoretical goodput, G, for a Wi-Fi-to-Wi-Fi connection can be estimated by considering both the transport layer TCP and the MAC/PHY layer efficiencies. So, we define:

$$G = \frac{\eta_{\text{TCP}} \times \eta_{802.11ax} \times C_{\text{PHY}}}{2},$$

where:

- η_{TCP} is the efficiency at the TCP level, defined as:

$$\eta_{\text{TCP}} = \frac{\text{MSS}}{\text{MSS} + \text{TCP Header} + \text{IP Header}}$$

with an MSS of 1460 bytes, a 20-byte TCP header, and a 20-byte IP header,

$$\eta_{\text{TCP}} = \frac{1460}{1460 + 20 + 20} \approx 0.9733 \ (97.33\%)$$

- $\eta_{802.11ax}$ is the efficiency of the 802.11ax MAC layer. Under ideal conditions, we assume an efficiency of approximately 80%:

$$\eta_{802.11ax} \approx 0.80 \ (80\%)$$

- The capacity C of the link is given as:

$$C_{\text{PHY}} = 960.7 \text{ Mbit/s}.$$

This value was measured using the command `iwlist wlan0 bitrate` (figure 2) on the device with the lower throughput, since in a Wi-Fi-to-Wi-Fi scenario the overall throughput is determined by the weakest link.

So the theoretical goodput is:

$$G = \frac{0.9733 \times 0.80 \times 960.7}{2} = 374 Mb/s$$

In table 1 , we summarize the results obtained from two different tests performed. Detailed outputs for these tests are reported in figure 1a and 1b in the Appendix.

Wifi - Wifi	TCP: Goodput per flow				
	Prediction	Max	Min	Average	Std
TEST 1	374	312.43	288.49	300.95	9.40
TEST 2	374	346.76	296.10	308.86	14.74

Table 1: TCP: Goodput per flow for different tests

Although the theoretical goodput prediction for the WiFi-to-WiFi TCP scenario accounts for protocol overhead and the half-duplex nature of the wireless medium, the actual measured throughput was consistently lower. This performance gap can be attributed to several factors inherent to Wi-Fi environments: contention and channel access delays, retransmissions at the MAC layer, variable link rate (rate adaptation), TCP congestion control and delayed ACKs, bufferbloat and queuing delays.
Additional Scenario
It is worth noting that, in some tests, the TCP goodput over Wi-Fi was significantly lower than expected, even when accounting for contention and retransmissions. One specific case involved placing the Wi-Fi client inside a metallic cabinet (figure 10), effectively turning it into a makeshift Faraday cage. As a result, the signal quality degraded substantially due to physical shielding and multipath distortion. The access point had to compensate by lowering the modulation and coding scheme (MCS), leading to a reduced PHY rate and increased retransmissions at the MAC layer. This setup, while unintentional at first, provided a concrete demonstration of how environmental factors can severely impact Wi-Fi performance. Table 2 shows the results obtained (figure 9a and 9b):

Wifi - Wifi	TCP: Goodput per flow				
	Prediction	Max	Min	Average	Std
TEST 1	374	79.09	57.20	64.30	7.40
TEST 2	374	75.98	52.84	64.92	5.81

Table 2: TCP: Goodput per flow for different tests

4.2 Scenario 2: WiFi-WiFi UDP

In this scenario, both the transmitter and the receiver are connected via a Wi-Fi link in infrastructure mode. As in the TCP case, data is transmitted from the source to the Access Point (AP) and then from the AP to the destination using the same channel. This half-duplex operation effectively halves the end-to-end throughput. Here, UDP is used at the transport layer.

The theoretical goodput, G, for a Wi-Fi-to-WiFi UDP connection is estimated by considering both the UDP-layer efficiency and the MAC/PHY layer efficiencies.

$$G = \frac{\eta_{\text{UDP}} \times \eta_{802.11ax} \times C_{\text{PHY}}}{2},$$

where:

- η_{UDP} is the efficiency at the UDP level, defined as:

$$\eta_{\text{UDP}} = \frac{\text{UDP Payload}}{\text{UDP Payload} + \text{UDP Header} + \text{IP Header}}$$

with a UDP payload of 1472 bytes, an 8-byte UDP header, and a 20-byte IP header,

$$\eta_{\text{UDP}} = \frac{1472}{1472 + 8 + 20} = \frac{1472}{1500} \approx 0.9813 \ (98.13\%)$$

- $\eta_{802.11ax}$ is the efficiency of the 802.11ax MAC layer. Under ideal conditions, we assume:

$$\eta_{802.11ax} \approx 0.80 \ (80\%).$$

- C_{PHY} is the capacity of the weakest link, measured as:

$$C_{\text{PHY}} = 960.7 \text{ Mbit/s}.$$

This value was obtained using the command `iwlist wlan0 bitrate` on the device with the lower throughput, since in a Wi-Fi-to-Wi-Fi scenario the overall throughput is determined by the weakest link.

Thus, the theoretical goodput for UDP is:

$$G = \frac{0.9813 \times 0.80 \times 960.7}{2} \approx \frac{754.18}{2} \approx 377.1 \text{ Mbit/s}.$$

In table 3 , we summarize the results obtained from two different tests performed. Detailed outputs for these tests are reported in figure 3a and 3b in the Appendix.

WiFi - WiFi	UDP: Goodput per flow				
	Prediction	**Max**	**Min**	**Average**	**Std**
TEST 1	377	355.6	329.42	344.50	8.24
TEST 2	377	361.02	308.34	345.94	17.84

Table 3: UDP: Goodput per flow for different tests.

UDP showed slightly better results than TCP, as it doesn't implement congestion control and is inherently more aggressive. We observed zero packet loss (figure 4), likely because the 5GHz band was exclusively used by the server and client during the test.

4.3 Scenario 3: WiFi-Eth TCP

In this scenario, the client is connected via Wi-Fi while the server is connected through a wired Ethernet link. This eliminates the double-hop transmission typical of WiFi-to-WiFi communication, allowing for higher efficiency and reduced contention over the wireless medium.
The theoretical TCP goodput is computed as:

$$G = \eta_{\text{TCP}} \times \eta_{802.11ax} \times C_{\text{PHY}}$$

Assuming:

- $$\eta_{\text{TCP}} = \frac{1460}{1460 + 20 + 20} \approx 0.9733 \ (97.33\%)$$
- $\eta_{802.11ax} = 0.80$,
- $C_{\text{PHY}} = 1000$ Mbit/s, since these tests were performed with a different actual bitrate.

Then:

$$G = 0.9733 \times 0.80 \times 1000 = 778.64 \text{ Mbit/s}$$

Compared to the WiFi-to-WiFi scenario, there is no need to divide the result by two, since the wireless channel is only traversed in one direction. The actual results, taken from figure 5a and 5b are summarized in Table 4. Differences between predicted and observed goodput can be attributed to MAC-layer retransmissions, channel contention, and physical link rate variability, as confirmed by Wireshark trace analysis.
As shown in Figure 11a, the nearly linear progression of the TCP sequence number demonstrates that the data is being transmitted steadily throughout the test, implying a stable data flow with minimal disruptions or frequent retransmissions. Figure 11b illustrates both the average TCP segment size and the corresponding throughput over time. The high segment sizes indicate efficient utilization of the Maximum Segment Size, while the fluctuations in throughput are typical of TCP dynamics, such as congestion control adjustments, acknowledgment delays, and occasional retransmissions. The observed decrease in throughput towards the end of the measurement period may indicate the final stages of the data transfer or temporary congestion-related slowdowns.

WiFi - Eth	TCP: Goodput per flow				
	Prediction	**Max**	**Min**	**Average**	**Std**
TEST 1	778.64	887.84	525.69	756.44	116.37
TEST 2	778.64	910.79	533.00	698.19	153.68

Table 4: TCP: Goodput per flow for different tests.

4.4 Scenario 4: WiFi-Eth UDP

This scenario replicates the previous one but uses UDP as the transport protocol. As with TCP, only one side of the communication crosses the wireless link, improving goodput compared to the WiFi-WiFi case.
We estimate the theoretical goodput using:

$$\eta_{\text{UDP}} = \frac{1472}{1472 + 8 + 20} \approx 0.981$$

$$G = \eta_{\text{UDP}} \times \eta_{802.11ax} \times C_{\text{PHY}} = 0.981 \times 0.80 \times 1000.0 = 784.80 \text{ Mbit/s}$$

Table 5 summarizes the results for this test (figure 6a and 5b). As expected, the results are higher than WiFi-WiFi UDP, though still impacted by the unpredictability of the wireless link. Occasional spikes or drops in goodput may be explained by interference, temporary buffer overruns, or packet loss.

WiFi - Eth	UDP: Goodput per flow				
	Prediction	Max	Min	Average	Std
TEST 1	784.80	783.82	620.95	755.59	55.60
TEST 2	784.80	783.86	670.71	769.95	35.73

Table 5: UDP: Goodput per flow in the WiFi-Ethernet scenario

4.5 Scenario 5: Eth-Eth TCP

In this scenario, we perform a test where the server and the client are connected via an Ethernet cable, using TCP at the transport layer.For each MSS-sized segment:

- MSS bytes are what the application receives
- MSS + TCP + IP + ETH bytes are transmitted at the physical layer

Thus, the per-packet efficiency is:

$$\eta_{\text{TCP}} \leq \frac{L_7\ (\text{Data})}{\text{Transmitted Data}} = \frac{1460}{1460 + 20 + 20 + 38} = 0,9492$$

The Theoretical Maximum Throughput – or maximum Goodput will thus be:

$$G < \eta_{\text{protocol}}\, C$$

The link capacity is 1000 Mbit/s, since we are using a Category 5e Ethernet cable. So we will have:

$$G < 1000 * 0,9492 = 949,2 Mbit/s$$

In table 6, we summarize the results obtained from two different tests performed. Detailed outputs for these tests are reported in Figure 7a and Figure 7b in the Appendix.

Eth - Eth	TCP: Goodput per flow				
	Prediction	Max	Min	Average	Std
TEST 1	949.2	941.33	940.25	941.06	0.33
TEST 2	949.2	941.36	941.27	941.33	0.03

Table 6: TCP: Goodput per flow for different tests

4.6 Scenario 6: Eth-Eth UDP

In this scenario, we perform a test where the server and the client are connected via an Ethernet cable, using UDP at the transport layer. Below are the relevant points and the formula for maximum efficiency:
For each MSS-sized segment

- MSS bytes are what the application receives, which is equal to 1472 bytes
- Overhead considerations:
  - UDP Header: 8 bytes
  - IP Header (IPv4): 20 bytes
  - Ethernet (Layer-2): Typically 38 bytes total
  - Thus, each transmitted packet of size MSS+UDP+IP+ETH includes both payload and protocol overhead.

Thus, the per-packet efficiency is:

$$\eta_{\text{UDP}} \leq \frac{L_7\ (\text{Data})}{\text{Transmitted Data}} = \frac{1472}{1472 + 8 + 20 + 38} = 0,957$$

The Theoretical Maximum Throughput – or maximum Goodput will thus be:

$$G < \eta_{\text{protocol}}\, C$$

The link capacity is 1000 Mbit/s, since we are using a Category 5e Ethernet cable. So we will have:

$$G < 1000 * 0,957\% = 957 Mbit/s$$

In table 7, we summarize the results obtained from two different tests performed. Detailed outputs for these tests are reported in Figure 8a and Figure 8b in the Appendix.

Eth - Eth	UDP: Goodput per flow				
	Prediction	Max	Min	Average	Std
TEST 1	957	956,48	955,76	956,15	0,25
TEST 2	957	956,47	956,13	956,40	0,10

Table 7: UDP: Goodput per flow for different tests

Overall, our UDP measurements closely match the theoretical predictions, with really small discrepancies due to practical factors such as packet queuing, interrupt overhead, and minor packet loss events. The results confirm that an efficient UDP transfer at near-gigabit rates is achievable than TCP, provided the underlying network hardware and OS stack are configured properly.

5 CONCLUSION

To sum up, these experiments offered us a revealing look into the contrast between theoretical throughput calculations and real-life network performance. While wired Ethernet generally matched the numbers we had anticipated, Wi-Fi's half-duplex nature, interference, and even physical surroundings, like walls or metal cabinets, often slashed speeds by a noticeable margin. This gap served as a powerful reminder that real-world performance can deviate significantly from neat laboratory equations, especially under less-than-ideal conditions. Yet, it was also reassuring to see that, with optimal setups and minimal interference, Wi-Fi can still come impressively close to its advertised speeds. Overall, these tests underscore the need to consider not just the protocols and hardware but also the environment when setting up or troubleshooting a network.

6 APPENDIX

```
┌──(gioele㉿LAPTOP-6DJRVHAK)-[~/Desktop]
└─$ python3 iperf3_client.py 192.168.1.189 MAX tcp
Running attempt 1/10 ...
Receiver Bandwidth for Attempt 1: 312.28 Mbps

Running attempt 2/10 ...
Receiver Bandwidth for Attempt 2: 311.19 Mbps

Running attempt 3/10 ...
Receiver Bandwidth for Attempt 3: 304.51 Mbps

Running attempt 4/10 ...
Receiver Bandwidth for Attempt 4: 312.43 Mbps

Running attempt 5/10 ...
Receiver Bandwidth for Attempt 5: 300.32 Mbps

Running attempt 6/10 ...
Receiver Bandwidth for Attempt 6: 291.14 Mbps

Running attempt 7/10 ...
Receiver Bandwidth for Attempt 7: 290.66 Mbps

Running attempt 8/10 ...
Receiver Bandwidth for Attempt 8: 288.49 Mbps

Running attempt 9/10 ...
Receiver Bandwidth for Attempt 9: 294.02 Mbps

Running attempt 10/10 ...
Receiver Bandwidth for Attempt 10: 304.46 Mbps


Results Summary:
Max Receiver Bandwidth: 312.43 Mbps
Min Receiver Bandwidth: 288.49 Mbps
Average Receiver Bandwidth: 300.95 Mbps
Standard Deviation: 9.40 Mbps
```

(a) Detailed results for TCP Test 1 (Wifi-Wifi scenario)

```
┌──(gioele㉿LAPTOP-6DJRVHAK)-[~/Desktop]
└─$ python3 iperf3_client.py 192.168.1.189 MAX tcp
Running attempt 1/10 ...
Receiver Bandwidth for Attempt 1: 304.07 Mbps

Running attempt 2/10 ...
Receiver Bandwidth for Attempt 2: 308.46 Mbps

Running attempt 3/10 ...
Receiver Bandwidth for Attempt 3: 297.90 Mbps

Running attempt 4/10 ...
Receiver Bandwidth for Attempt 4: 301.06 Mbps

Running attempt 5/10 ...
Receiver Bandwidth for Attempt 5: 296.10 Mbps

Running attempt 6/10 ...
Receiver Bandwidth for Attempt 6: 299.94 Mbps

Running attempt 7/10 ...
Receiver Bandwidth for Attempt 7: 305.98 Mbps

Running attempt 8/10 ...
Receiver Bandwidth for Attempt 8: 316.21 Mbps

Running attempt 9/10 ...
Receiver Bandwidth for Attempt 9: 312.10 Mbps

Running attempt 10/10 ...
Receiver Bandwidth for Attempt 10: 346.76 Mbps


Results Summary:
Max Receiver Bandwidth: 346.76 Mbps
Min Receiver Bandwidth: 296.10 Mbps
Average Receiver Bandwidth: 308.86 Mbps
Standard Deviation: 14.74 Mbps
```

(b) Detailed results for TCP Test 2 (Wifi-Wifi scenario)

Figure 1: Detailed results for TCP tests in the Wifi-Wifi scenario.

```
┌──(gioele㉿LAPTOP-6DJRVHAK)-[~/Desktop]
└─$ iwlist wlan0 bitrate
wlan0     unknown bit-rate information.
          Current Bit Rate:960.7 Mb/s
```

Figure 2: Wi-Fi weakest link capacity

```
└─$ python3 iperf3_client.py 192.168.1.125 MAX udp
Running attempt 1/10 ...
Receiver Bandwidth for Attempt 1: 343.31 Mbps

Running attempt 2/10 ...
Receiver Bandwidth for Attempt 2: 355.60 Mbps

Running attempt 3/10 ...
Receiver Bandwidth for Attempt 3: 348.38 Mbps

Running attempt 4/10 ...
Receiver Bandwidth for Attempt 4: 354.84 Mbps

Running attempt 5/10 ...
Receiver Bandwidth for Attempt 5: 345.98 Mbps

Running attempt 6/10 ...
Receiver Bandwidth for Attempt 6: 345.80 Mbps

Running attempt 7/10 ...
Receiver Bandwidth for Attempt 7: 340.76 Mbps

Running attempt 8/10 ...
Receiver Bandwidth for Attempt 8: 333.88 Mbps

Running attempt 9/10 ...
Receiver Bandwidth for Attempt 9: 329.42 Mbps

Running attempt 10/10 ...
Receiver Bandwidth for Attempt 10: 347.01 Mbps


Results Summary:
Max Receiver Bandwidth: 355.60 Mbps
Min Receiver Bandwidth: 329.42 Mbps
Average Receiver Bandwidth: 344.50 Mbps
Standard Deviation: 8.24 Mbps
```

(a) Detailed results for UDP Test 1 (Wifi-Wifi scenario)

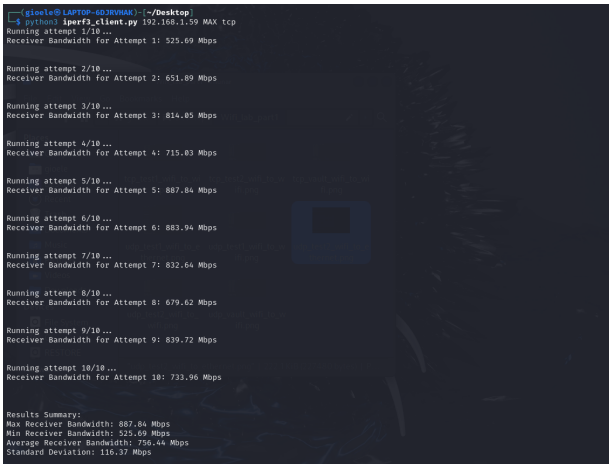
```
┌──(gioele㉿LAPTOP-6DJRVHAK)-[~/Desktop]
└─$ python3 iperf3_client.py 192.168.1.125 MAX udp
Running attempt 1/10 ...
Receiver Bandwidth for Attempt 1: 349.40 Mbps

Running attempt 2/10 ...
Receiver Bandwidth for Attempt 2: 347.12 Mbps

Running attempt 3/10 ...
Receiver Bandwidth for Attempt 3: 350.38 Mbps

Running attempt 4/10 ...
Receiver Bandwidth for Attempt 4: 349.73 Mbps

Running attempt 5/10 ...
Receiver Bandwidth for Attempt 5: 358.33 Mbps

Running attempt 6/10 ...
Receiver Bandwidth for Attempt 6: 361.02 Mbps

Running attempt 7/10 ...
Receiver Bandwidth for Attempt 7: 358.72 Mbps

Running attempt 8/10 ...
Receiver Bandwidth for Attempt 8: 357.38 Mbps

Running attempt 9/10 ...
Receiver Bandwidth for Attempt 9: 308.34 Mbps

Running attempt 10/10 ...
Receiver Bandwidth for Attempt 10: 318.93 Mbps


Results Summary:
Max Receiver Bandwidth: 361.02 Mbps
Min Receiver Bandwidth: 308.34 Mbps
Average Receiver Bandwidth: 345.94 Mbps
Standard Deviation: 17.84 Mbps
```

(b) Detailed results for UDP Test 2 (Wifi-Wifi scenario)

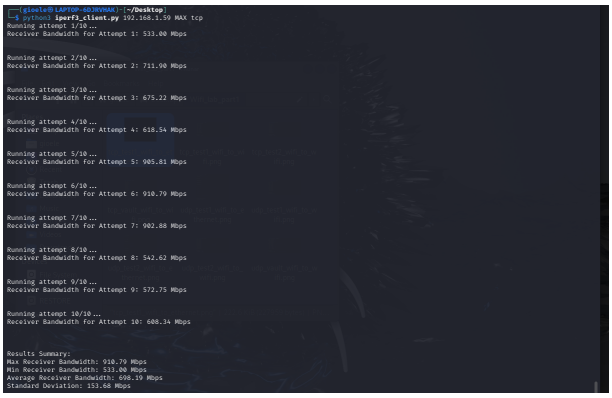
Figure 3: Detailed results for UDP tests in the Wifi-Wifi scenario.

```
-----------------------------------------------------------
Server listening on 5201 (test #64)
-----------------------------------------------------------
Accepted connection from 192.168.1.211, port 42022
[  5] local 192.168.1.125 port 5201 connected to 192.168.1.211 port 41636
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datag
rams
[  5]   0.00-1.01   sec  30.5 MBytes   255 Mbits/sec  0.021 ms  0/22098 (0%)
[  5]   1.01-2.00   sec  36.6 MBytes   309 Mbits/sec  0.013 ms  0/26522 (0%)
[  5]   2.00-3.00   sec  40.8 MBytes   342 Mbits/sec  0.014 ms  0/29512 (0%)
[  5]   3.00-4.00   sec  39.1 MBytes   327 Mbits/sec  0.029 ms  0/28301 (0%)
[  5]   4.00-5.00   sec  40.3 MBytes   338 Mbits/sec  0.011 ms  0/29166 (0%)
[  5]   5.00-6.00   sec  39.7 MBytes   333 Mbits/sec  0.028 ms  0/28723 (0%)
[  5]   6.00-7.00   sec  40.4 MBytes   340 Mbits/sec  0.010 ms  0/29278 (0%)
[  5]   7.00-8.00   sec  41.3 MBytes   347 Mbits/sec  0.029 ms  0/29939 (0%)
[  5]   8.00-9.01   sec  35.5 MBytes   297 Mbits/sec  0.017 ms  0/25694 (0%)
[  5]   9.01-10.00  sec  36.2 MBytes   304 Mbits/sec  0.024 ms  0/26214 (0%)
[  5]  10.00-10.01  sec   226 KBytes   319 Mbits/sec  0.117 ms  0/168 (0%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datag
rams
[  5]   0.00-10.01  sec   381 MBytes   319 Mbits/sec  0.117 ms  0/275687 (0%)  r
eceiver
-----------------------------------------------------------
```

Figure 4: UDP packet loss

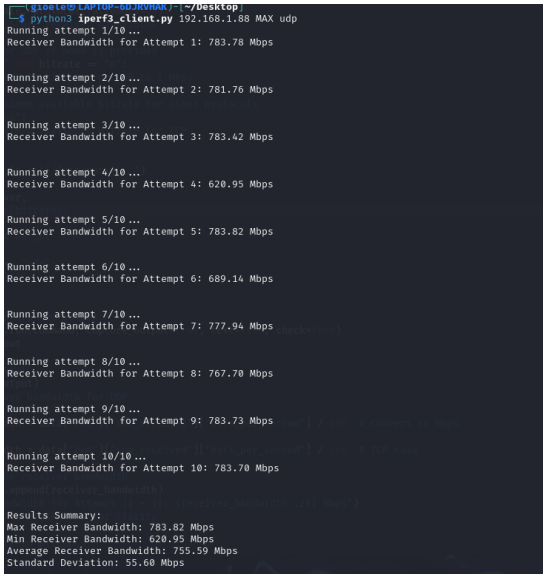


(a) Detailed results for TCP Test 1 (WiFi-Eth scenario)

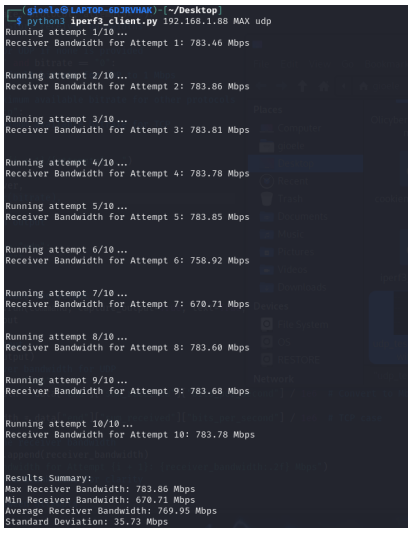


(b) Detailed results for TCP Test 2 (WiFi-Eth scenario)

Figure 5: Detailed results for TCP tests in the WiFi-Eth scenario.

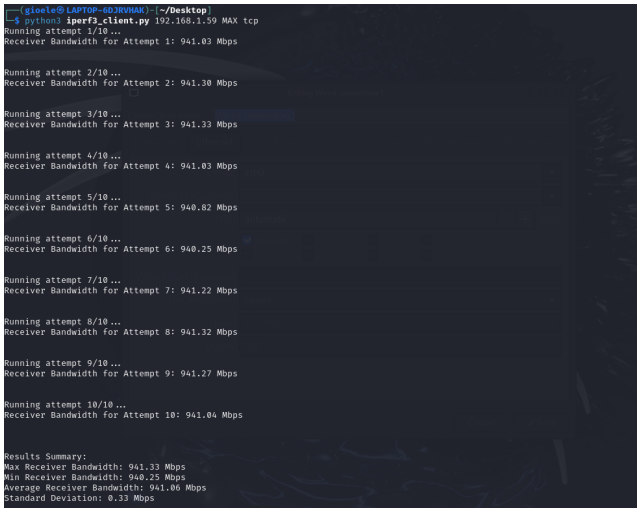


(a) Detailed results for UDP Test 1 (WiFi-Eth scenario)

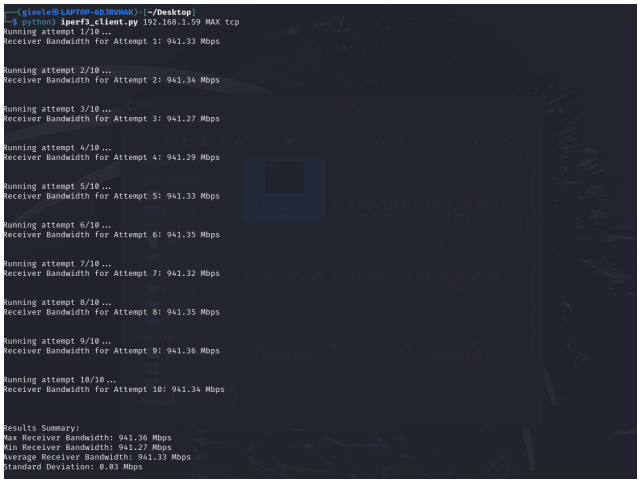


(b) Detailed results for UDP Test 2 (WiFi-Eth scenario)

Figure 6: Detailed results for UDP tests in the WiFi-Eth scenario.



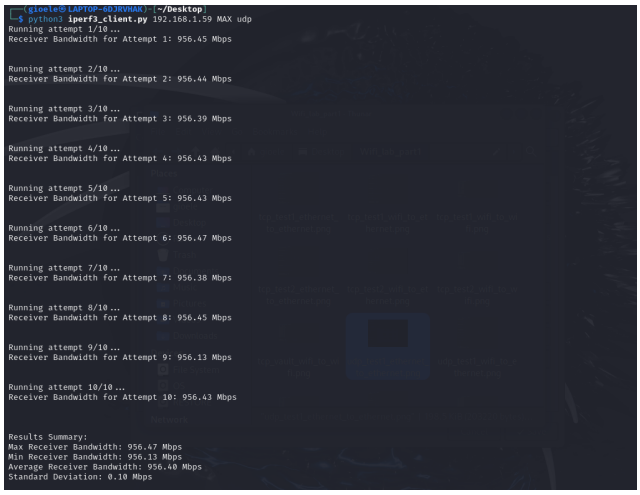
(a) Detailed results for TCP Test 1 (Eth-Eth scenario)



(b) Detailed results for TCP Test 2 (Eth-Eth scenario)

Figure 7: Detailed results for TCP tests in the Eth-Eth scenario.

(a) Detailed results for UDP Test 1 (Eth-Eth scenario)



(b) Detailed results for UDP Test 2 (Eth-Eth scenario)

Figure 8: Detailed results for UDP tests in the Eth-Eth scenario.

```
└─$ python3 iperf3_client.py 192.168.1.250  tcp
Running attempt 1/10 ...
Attempt 1: 79.09 Mbps
Running attempt 2/10 ...
Attempt 2: 65.91 Mbps
Running attempt 3/10 ...
Attempt 3: 67.36 Mbps
Running attempt 4/10 ...
Attempt 4: 74.62 Mbps
Running attempt 5/10 ...
Attempt 5: 60.75 Mbps
Running attempt 6/10 ...
Attempt 6: 58.85 Mbps
Running attempt 7/10 ...
Attempt 7: 59.29 Mbps
Running attempt 8/10 ...
Attempt 8: 59.58 Mbps
Running attempt 9/10 ...
Attempt 9: 60.35 Mbps
Running attempt 10/10 ...
Attempt 10: 57.20 Mbps

Results Summary:
Max Bandwidth: 79.09 Mbps
Min Bandwidth: 57.20 Mbps
Average Bandwidth: 64.30 Mbps
Standard Deviation: 7.40 Mbps
```

(a) Detailed results for TCP Test 1 inside metallic cabinet (WiFi-Wifi scenario)

```
┌──(gioele㉿LAPTOP-6DJRVHAK)-[~/Desktop]
└─$ python3 iperf3_client.py 192.168.1.250  tcp
Running attempt 1/10 ...
Attempt 1: 75.98 Mbps
Running attempt 2/10 ...
Attempt 2: 64.01 Mbps
Running attempt 3/10 ...
Attempt 3: 66.72 Mbps
Running attempt 4/10 ...
Attempt 4: 66.50 Mbps
Running attempt 5/10 ...
Attempt 5: 67.75 Mbps
Running attempt 6/10 ...
Attempt 6: 63.60 Mbps
Running attempt 7/10 ...
Attempt 7: 68.66 Mbps
Running attempt 8/10 ...
Attempt 8: 52.84 Mbps
Running attempt 9/10 ...
Attempt 9: 65.73 Mbps
Running attempt 10/10 ...
Attempt 10: 65.46 Mbps

Results Summary:
Max Bandwidth: 75.98 Mbps
Min Bandwidth: 52.84 Mbps
Average Bandwidth: 64.92 Mbps
Standard Deviation: 5.81 Mbps
```

(b) Detailed results for TCP Test 2 inside metallic cabinet (WiFi-Wifi scenario)

Figure 9: Detailed results for TCP tests in the Wifi-Wifi scenario inside metallic cabinet.

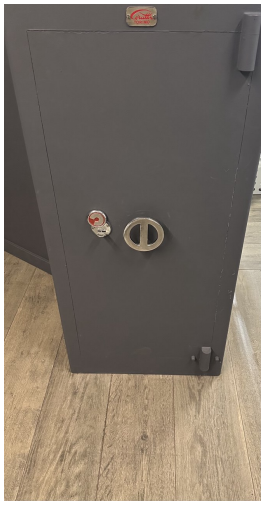
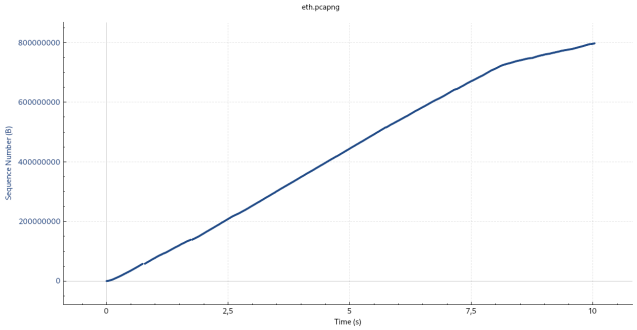
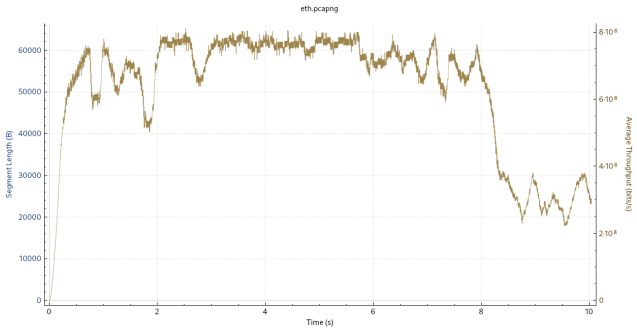


Figure 10: Vault



(a) TCP sequence number progression over time



(b) Average throughput

```python
import subprocess
import statistics
import argparse
import json
import re
import time

Tabnine | Edit | Test | Explain | Document
def run_iperf_test(server, bitrate, protocol, attempts=10):
    receiver_bandwidths = []

    # Set the default bitrate for UDP if none is provided
    if protocol.lower() == "udp" and bitrate == "0":
        bitrate = "1M"  # Set default bitrate for UDP to 1 Mbps
    elif bitrate == "MAX":
        bitrate = "0"  # Use maximum available bitrate for other protocols
    elif protocol.lower() == "tcp":
        bitrate = "0"  # Use maximum available bitrate for TCP

    for i in range(attempts):
        print(f"Running attempt {i + 1}/{attempts}...")
        command = [
            "iperf3", "-c", server,
            "-b", str(bitrate),
            "-t", "10",
            "-J"  # Request JSON output
        ]
        if protocol.lower() == "udp":
            command.append("-u")

        try:
            # Run iperf3 and capture the output
            result = subprocess.run(command, capture_output=True, text=True, check=True)
            output = result.stdout

            # Parse the JSON output
            data = json.loads(output)
            # Extract the receiver bandwidth for UDP
            if protocol.lower() == "udp":
                receiver_bandwidth = data["end"]["sum_received"]["bits_per_second"] / 1e6  # Convert to Mbps
            else:
                receiver_bandwidth = data["end"]["sum_received"]["bits_per_second"] / 1e6  # TCP case

            # Store and print the receiver bandwidth
            receiver_bandwidths.append(receiver_bandwidth)
            print(f"Receiver Bandwidth for Attempt {i + 1}: {receiver_bandwidth:.2f} Mbps")
            print("\n")  # Print a blank line for clarity

            # Optional pause between attempts
            time.sleep(1)

        except subprocess.CalledProcessError as e:
            print(f"Error in attempt {i + 1}: {e}\nOutput:\n{e.output}")
        except json.JSONDecodeError as e:
            print(f"JSON decoding error in attempt {i + 1}: {e}")
        except Exception as e:
            print(f"Unexpected error in attempt {i + 1}: {e}")

    # Calculate statistics if we have recorded any receiver bandwidths
    if receiver_bandwidths:
        max_bandwidth = max(receiver_bandwidths)
        min_bandwidth = min(receiver_bandwidths)
        avg_bandwidth = sum(receiver_bandwidths) / len(receiver_bandwidths)
        std_dev_bandwidth = statistics.stdev(receiver_bandwidths) if len(receiver_bandwidths) > 1 else 0  # Calculate standard deviation

        print("\nResults Summary:")
        print(f"Max Receiver Bandwidth: {max_bandwidth:.2f} Mbps")
        print(f"Min Receiver Bandwidth: {min_bandwidth:.2f} Mbps")
        print(f"Average Receiver Bandwidth: {avg_bandwidth:.2f} Mbps")
        print(f"Standard Deviation: {std_dev_bandwidth:.2f} Mbps")  # Output standard deviation
    else:
        print("No successful tests were conducted.")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Run iperf3 as a client and collect statistics.")
    parser.add_argument("server", help="Target IP address to test against")
    parser.add_argument("bitrate", type=str, nargs='?', default="0", help="Bitrate in bits per second (e.g., 1000000 for 1Mbps, 'MAX' for max available)")
    parser.add_argument("protocol", choices=["tcp", "udp"], help="Protocol to use (tcp/udp)")

    args = parser.parse_args()
    run_iperf_test(args.server, args.bitrate, args.protocol)

```

Figure 12: Python script