WI-FI PERFORMANCE LAB REPORT

Enrico Di Stasio s323075, Alberto Ameglio s330946, Giovanni Luca Di Bella s332088
Lag4life

1 INTRODUCTION

In this lab our aim is to show the differences in behaviour and performance when using wireless connectivity instead of wired connectivity or a mix of the two. Specifically we are sending streams of packet from a client to a server exploring three different situations:

- Both client and server connected through Ethernet
- Both client and server connected through WiFi
- Client connected through Wi-Fi and server connected through Ethernet

In particular, for each one of these cases, we sent a single flow of data first from client to server and then from server to client. Then we proceeded to measure the Bitrate of each flow and successively computed minimum, maximum, average and standard deviation of such Bitrates. We performed the test using both TCP and UDP protocols.

2 TOOLS AND THEORY

To perform the measurements we used the iperf3 tool [2] which we set up on both server and client to achieve a connection. Iperf3 is a network testing tool primarily used to measure the bandwidth and performance of IP networks. It is designed to test the throughput of a network and diagnose issues related to network speed and capacity. In our test in particular we set up the server by first installing the tool using the apt packet manager

```
$ sudo apt install iperf3
```

and then running it as a server using the command

```
$ iperf3 -s
```

with the option *-s* to indicate that it was the server. Depending on the experiment we also modified the port on which the server was running (default port is 5201). On the client instead, after installing the iperf3 tool, we used a python script [5] to perform the measurements ten times. The script started the iperf3 client with the following commands: In case of TCP client-to-server flow

```
$ iperf3 -c <server_ip_address> -p <server_port>
```

In case of UDP client-to-server flow

```
$ iperf3 -c <server_ip_address> -p <server_port>
    -u -b 50M
```

In case of TCP server-to-client flow

```
$ iperf3 -c <server_ip_address> -p <server_port>
    -R
```

In case of UDP server-to-client flow.

```
$ iperf3 -c <server_ip_address> -p <server_port>
    -u -b 100M -R
```

We first passed the IP address of the server and then we also used a couple of options depending on the scenario:

- option *-c* is to indicate that the device running iperf3 is the client
- option *-p <port number>* indicates the port on which the server is running (in case it's different from the default one)
- option *-u* indicates that we want to use the UDP protocol (default is TCP)
- option *-b <bandwidth>* indicates that we want to manually set the bandwidth, we didn't use it for TCP measurements since there isn't a limit by default, while we used it in the UDP ones to circumvent the 1 Mbits/sec deafult limit by setting it 100 Mbits/sec
- option *-R* indicates that we want the client to receive from the server instead of sending (used for the server-to-client flow)

We were expecting different results between experiments due to the different efficiencies of Wi-Fi and Ethernet computed during the lectures:

- on TCP-Ethernet we were a expecting a Goodput of around 94.9 Mbits/sec due to the efficiency μ being 0.949 and the link capacity of 100 Mbits/sec
- on UDP-Ethernet we were a expecting a Goodput of around 95,7 Mbits/sec due to the efficiency μ being 0.957 and the link capacity of 100 Mbits/sec
- on TCP-WiFi we were a expecting a Goodput of around 50 Mbits/sec due to the efficiency μ being 0.5 and the link capacity of 100 Mbits/sec
- on UDP-WiFi we were a expecting a Goodput of around 55 Mbits/sec due to the efficiency μ being 0.55 and the link capacity of 100 Mbits/sec

In the Results section we are going to show summary tables with the expectations and the actual results.

3 LAB SETUP AND SCENARIOS

3.1 Script

To automate the execution of multiple Iperf3 executions we wrote a python script called ***performance.py*** which, through the subprocess library [6], executes the Iperf3 command 10 times for each condition, i.e. for TCP and UDP and for TCP and UDP in receive mode. The script, through a regex [4] instruction, extrapolates the Bitrate from the output for each execution and finally calculates: minimum, maximum, average and standard deviation and prints it on a file while at the same time dumping the packets using the **tcpdump tool** [7] that can be called via command line. The script, via the argparse library [1], takes the following as arguments:

- *-a <ip_address>*: address of the iperf3 server
- *-p <port>*: port on which the iperf3 server is running (optional in case it's not the default one)
- *-i <interface>*: interface on which we want to run the experiments

- *-f <filename>*: filename of the file on which the **tcpdump** is going to be performed

The complete script is present in the appendix [A] of this report. To install the tool correctly refer to the README in the Github repository. [5] [A]

3.2 Network Setup

The experiments were performed on an home network with all devices connected to a switched LAN. The configuration of the LAN was the following:

- **Default Gateway: 192.168.1.1**
- **Subnet Mask: 255.255.255.0**
- **Default DNS: 192.168.1.1**

The IP addresses of the clients will be specified in the corresponding experiments

3.3 Server Setup

The server was running on a Raspberry-Pi 3 Model B using a Rasberry Pi OS Lite [3] based on the Debian GNU/Linux distribution with two network interfaces:

- *eth0*, the Ethernet interface with IP address 192.168.1.28 and maximum link speed of 100.0 Mbits/sec used for the Ethernet and mixed experiments
- *wlan0*, the Wi-Fi interface with IP address 192.168.1.27 and maximum link speed of 65.0 Mbits/sec used for the Wi-Fi experiment

3.4 Ethernet-Ethernet Client Setup

For the client in the only-Ethernet conditions we used a laptop running an Ubuntu OS with a USB network interface *enx000ec6ac1f51* with maximum link speed of 100 Mbits/sec. We executed the python script using the following command:

```
$ python3 performance.py -i enx000ec6ac1f51
-a 192.168.1.28 -f eth-eth
```

It performs 10 times the command iperf3 for each of the scenarios (TCP, UDP, TCP reverse and UDP reverse) dumping the the packet exchanges in 4 different .cap files and finally the Min, Max, Avg and Std computations in a .txt file.

3.5 Wi-Fi-Wi-Fi Client Setup

For the client in the only-Wi-fi conditions we used a laptop running an Ubuntu OS with with Intel(R) Wi-Fi 6 AX201 160MHz as a network interface *wlan0* with maximum link speed of 100 Mbit/sec: We ran the Python script using the following command:

```
$ python3 performance.py -i wlan0
-a 192.168.1.27 -f Wifi-Wifi
```

and the script runs the code as described in the previous paragraph. [3.4] [3.1]

3.6 Wi-Fi-Ethernet Client Setup

In the final scenario, we opted to connect the server running on a Raspberry Pi 3 via Ethernet while the client connects via Wi-Fi. For the client we used a laptop running an Ubuntu OS and an USB Wi-Fi antenna with a maximum link speed of 100 Mbits/sec. As in the previous cases, we executed the script using the following command:

```
$ python3 performance.py -i wlp4s0
-a 192.168.1.28 -f Eth-WiFi
```

4 RESULTS

4.1 Ethernet-Ethernet Results

4.1.1 TCP Client-to-Server Here we transmitted a number of packets from the Ubuntu Laptop to the Raspberry Server and recorded the average speed for each iperf3 run. The results we obtained are as follows:

Prediction	Average	Min	Max	Std
94.9 Mb/s	94.1 Mb/s	94.1 Mb/s	94.1 Mb/s	0.0 Mb/s

The results are very close to the theoretical maximum throughputs and most of all are very consistent throughout the entire experiment. In figure [1] we can observe the progression of the Sequence Number of the TCP packets, observing the positive and constant slope of the function we can tell that the throughput was mostly constant as it can be expected from our measurements and the reliability of Ethernet. This is confirmed in figure [7] which shows

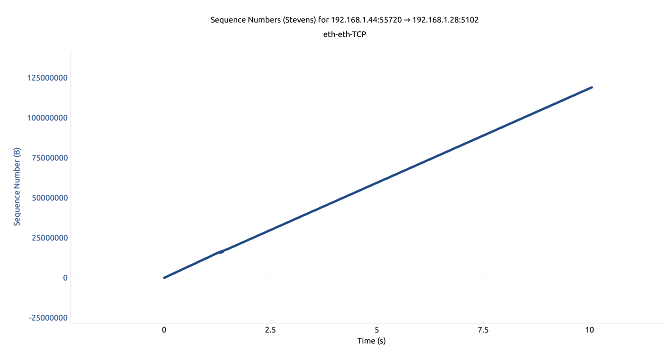

Figure 1: Stevens TCP Plot TCP of Eth-Eth

an high average Throughput with the exception of a small window between 1 second and 2.5 seconds where we probably experienced a bit of congestion in the network interface. As a matter of fact the Segment Length shown is way higher than the default Maximum Segment Size of TCP (1500 B), this is because the client OS supports TCP Segmentation Offload which leaves to the NIC (Network Interface Card) the role of splitting the Packets into MSS compliant sizes. Combined with the fact that the dumping is performed at the client side, the Wireshark application sees bigger Segments and detects some buffer congestion at the NIC. Although we had lower throughput we still did not encounter any packet loss, but just some increased Round-Trip-Time as evident in figure [2], where we have a peak in that same small window.

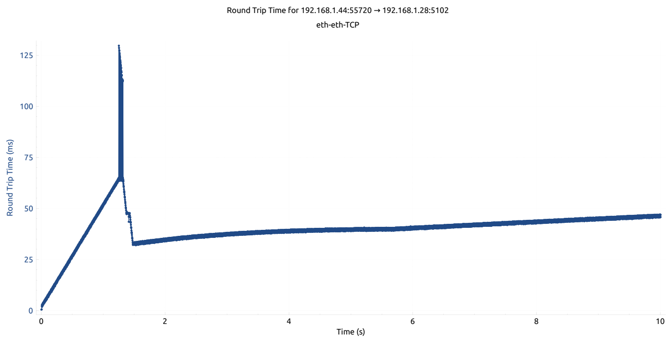

Figure 2: Round-Trip-Time Plot TCP of Eth-Eth

4.1.2 TCP Server-to-Client

Prediction	Average	Min	Max	Std
94.9 Mb/s	94.1 Mb/s	94.1 Mb/s	94.1 Mb/s	0.0 Mb/s

The results in the reverse-flow experiment are very consistent with the regular one, with the measurements being basically the same so we omit the relative plots, except for the Throughput one in figure [8] which shows that when the dumping is performed at the receiver side (this time the client) the correct MSS is shown

4.1.3 UDP Client-to-Server Using the same conditions as in the TCP case we ran the experiment and recorded these data:

Prediction	Average	Min	Max	Std
95.7 Mb/s	95.6 Mb/s	95.6 Mb/s	95.6 Mb/s	0.0 Mb/s

As in the TCP case Ethernet proves to deliver a constant high throughput with very few late or loss packets. We show the I/O graph in figure [9] showing the flow of the bits from client to server in the interval from 0 to 100 seconds with a very constant throughput, with dips only when ending an iperf3 run and starting a new one.

4.1.4 UDP Server-to-Client

Prediction	Average	Min	Max	Std
95.7 Mb/s	95.6 Mb/s	95.6 Mb/s	95.7 Mb/s	0.0 Mb/s

Again, the results are very close to the ideal case as shown in the plot in figure [9] from the 100 seconds mark up to the end, with the same situation as in the client to server case.

4.2 Wi-Fi-Wi-Fi Results

4.2.1 TCP Client-to-Server of Wifi-Wifi

Prediction	Average	Min	Max	Std
50.0 Mb/s	45.2 Mb/s	40.1 Mb/s	48.6 Mb/s	2.3 Mb/s

The results are very close to the theoretical maximum throughtput and above all we were able to observe consistent results during the entire experiment. The standard deviation, as we expected, differs a little from that obtained during the Ethernet-Ethernet experiment, but always falling within values acceptable. As expected from the lower reliability of Wifi, in the figure [10], we can observe the progression of the Sequence Number of the TCP packets, analyzing the positive and constant slope of the function we can say that the throughput was quite constant with some slight visible differences compared to Ethernet [1]. This is confirmed in the figure [3] which shows the average connection throughput, indicated by the brown line, which grows rapidly in the first seconds, reaching a level of around 50 Mbps. The blue dots indicate individual TCP segments that have been sent over time, showing a variety of lengths ranging from less than 10,000 bytes to over 20,000 bytes. This variation highlights dynamic and variable data traffic. The continuous variation of TCP segments suggests a dynamic nature of the connection and variability in data transfer over time. These differences are more evident comparing this graph [3] with that of Ethernet [7].

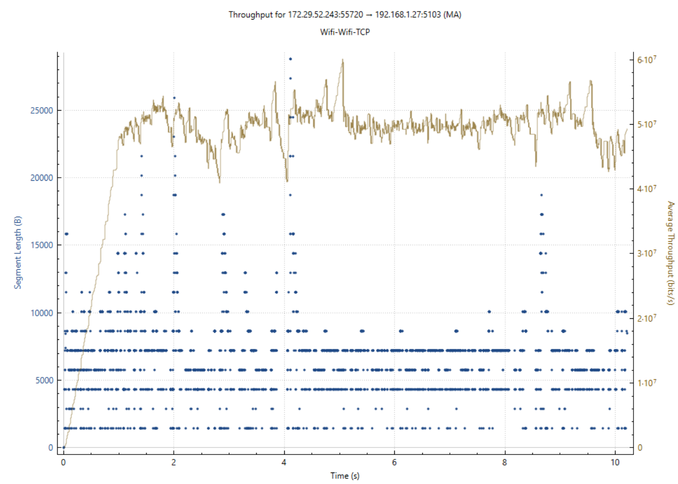

Figure 3: Throughput Plot TCP of Wifi-Wifi

The throughput was slightly lower than what we expected and can also be explained via the graph [4].

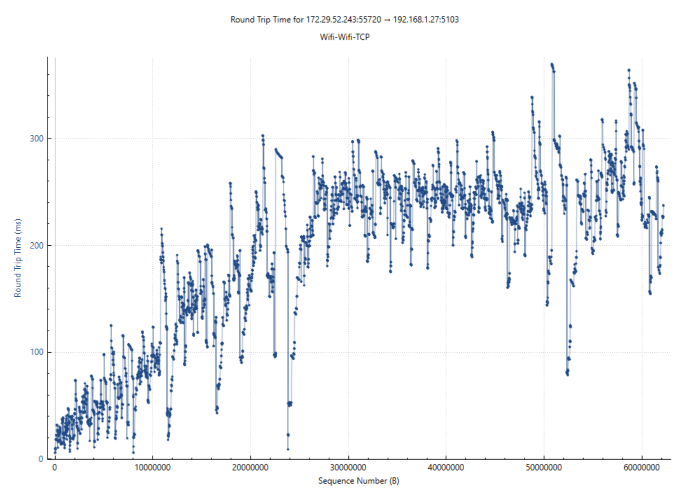

Figure 4: Round-Trip-Time Plot TCP of Wifi-Wifi

Each dot on it represents the round-trip time in milliseconds for sequences of data packets sent between the two hosts. The graph shows various fluctuations and peaks that may have been caused by network congestion problems or by the wireless interference that characterizes Wi-Fi. This difference is accentuated when compared with the Ethernet graph [2].

4.2.2 TCP Server-to-Client

Prediction	Average	Min	Max	Std
50.0 Mb/s	35.8 Mb/s	35.5 Mb/s	36.1 Mb/s	0.2 Mb/s

The results in the reverse flow experiment are consistent with those shown previously [4.2.1], with a lower average that we can associate with the Rasberry network card, so we omit the related graphs, except than that of the throughput in the figure [11] which shows that greater stability in the fluctuations in the throughput.

4.2.3 UDP Client-to-Server Using the same conditions as in the TCP case we ran the experiment and recorded these data:

Prediction	Average	Min	Max	Std
55.0 Mb/s	50.8 Mb/s	48.8 Mb/s	55.0 Mb/s	2.7 Mb/s

As in the TCP [4.2.1] case Ethernet proves to deliver a constant high throughput with very few late or loss packets.

4.2.4 UDP Server-to-Client

Prediction	Average	Min	Max	Std
55.0 Mb/s	42.2 Mb/s	41.1 Mb/s	43.4 Mb/s	0.8 Mb/s

Again, the results are very close to the ideal case as in the client to server case [4.2.3].

4.3 Wi-Fi-Ethernet Results

4.3.1 TCP Client-to-Server In this case we sent data from the laptop (client) connected via Wi-Fi using the internal wifi interface to the Raspberry connected via Ethernet (Server). The results are showed in the table below:

Prediction	Average	Min	Max	Std
50 Mb/s	46.5 Mb/s	42.4 Mb/s	47.8 Mb/s	2.1 Mb/s

$$G_{\text{TCP Max}} \leq \mu_{\text{wi-fiTCP}} * C_{\text{bottleneck}} = 0.50 * 100 = 50 \text{ Mb/s} \quad (1)$$

As shown in Figure [12] the sequence number flow follows a linear trend without any re-transmitted packet, this could be referred to the fact that even though the laptop was connected through Wi-Fi, there was no obstruction nor traffic that made some packet to be lost, therefore, since the server was connected to Ethernet no interferences showed up. Very similar results can be seen in the Throughput plot (Figure [13] in which after the firsts instants during which the iperf command was just executed and the packet started being transmitted, there's an almost constant behaviour with a mean value of $4.8 \times 10^7 bits/s$ (48 Mb/s) and characterized by small fluctuations in the order of 4 Mb (as showed computing the difference between the Min and Max value).

4.3.2 TCP Server-to-Client

Prediction	Average	Min	Max	Std
50 Mb/s	48 Mb/s	42.8 Mb/s	48.6 Mb/s	1.8 Mb/s

In the opposite case, instead, we notice a larger discontinuity that, as showed in Figure [5] causes not just idle instants in which the server waits for the ack packet, but also packets with a lower sequence number acknowledged late, causing, at certain instants a negative slope for the plot.

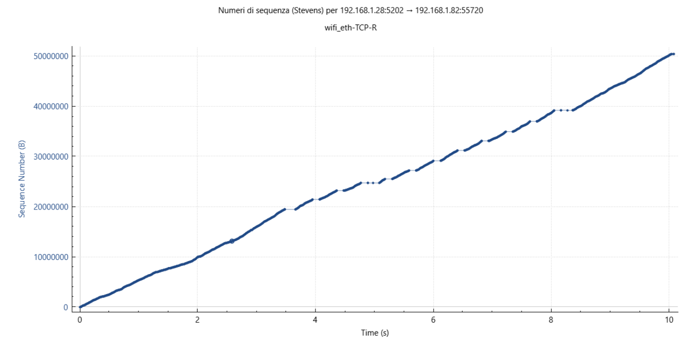

Figure 5: Sequence Number TCP Reversed of Wifi-Eth

The reasons for this discontinuity must are made clear when analyzing the RTT plot in Figure [6]: As we can see, there are some peaks in correspondence with the instants where the slope is zero. From this consideration we expect that during transmission over the wi-fi network there have been some interferences, or packet loss, that enlarged the RTT.

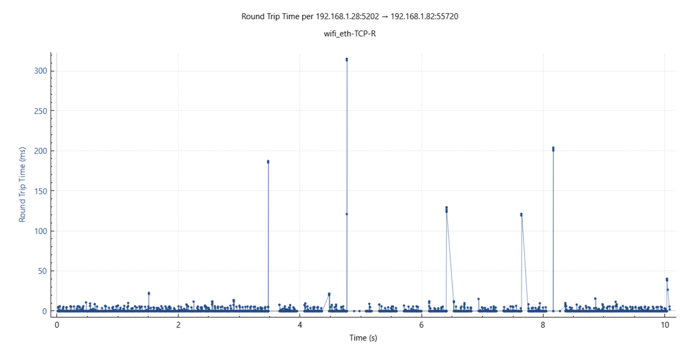

Figure 6: RTT TCP Reversed of Wifi-Eth

4.3.3 UDP Server-to-Client and Reverse Also in this case, using the same conditions of TCP, we obtained these results with UDP server to client and then client to server.

Prediction	Average	Min	Max	Std
55 Mb/s	43.5 Mb/s	38.9 Mb/s	53.6 Mb/s	5.0 Mb/s
55 Mb/s	48.5 Mb/s	46.2 Mb/s	49.6 Mb/s	1.2 Mb/s

$$G_{\text{UDP Max}} \leq \mu_{\text{wi-fiUDP}} * C_{\text{bottleneck}} = 0.55 * 100 = 55 \text{ Mb/s} \quad (2)$$

In contrast to TCP, UDP doesn't show many problems, analyzing Figure [14] we can see that except from second 50 the Bytes transmitted mantain a constant decreasing trend just in the moment in which the new iperf command is executed.

5 CONCLUSION

In conclusion we evaluated 12 different scenarios to test the performance of Wi-Fi link compared to the wired one. As expected, the results, demonstrate that a fully wired connection provides a very stable and interference-free link between the two devices, granting a reliable communication with no re-transmissions. The Ethernet-WiFi communication, despite not being as reliable as the fully wired one, still offers a stable communication channel. However, it exhibits some delays in the communication from server to client due to the wireless channel, as discussed in the results. In contrast with the other two cases, the performances of the fully wireless communication system supports the notion that the channel is unreliable and prone to interference, resulting in a very unstable throughput and an oscillating round trip time dependent on the channel conditions.

REFERENCES

[1] Python Software Foundation [n.d.]. *argparse Documentation*. Python Software Foundation. https://docs.python.org/3/library/argparse.html

[2] Iperf Foundation [n.d.]. *Iperf Documentation*. Iperf Foundation. https://iperf.fr/

[3] Rasberry Foundation [n.d.]. *Rasberry OS LITE Documentation*. Rasberry Foundation. https://www.raspberrypi.com/documentation/computers/getting-started.html

[4] Python Software Foundation [n.d.]. *regex Documentation*. Python Software Foundation. https://docs.python.org/3/library/re.html

[5] Project Group [n.d.]. *Repository GitHUb WifiPerformance-Iperf3*. Project Group. https://github.com/MR-NBD/WifiPerformance-Iperf3

[6] Python Software Foundation [n.d.]. *subprocess Documentation*. Python Software Foundation. https://docs.python.org/3/library/subprocess.html

[7] tcpdump Foundation [n.d.]. *tcpdum Documentation*. tcpdump Foundation. https://www.tcpdump.org/

A APPENDIX

```
import subprocess
import argparse
from colorama import Fore, Style
import re
import math
from datetime import datetime


def perf(options, protocol, receive, port):
    # Run the hyperf3 command and capture the output
    attempts = 0
    while attempts < 4:
        attempts += 1
        if protocol == "udp":
            result = subprocess.run(
                [
                    "iperf3",
                    "-c",
                    options.address,
                    port,
                    "--cport",
                    "55720",
                    "-u",
                    "-b",
                    "50M",
                    receive,
                ],
                capture_output=True,
                text=True,
            )
        else:
            result = subprocess.run(
                ["iperf3", "-c", options.address, port, "--cport", "55720", receive],
                capture_output=True,
                text=True,
            )
        # Check if the command was executed successfully
        if result.returncode == 0:
            output = result.stdout
            list = str.splitlines(output)

            # Use regex to find the bitrates of the sender and receiver
            sender_bitrate = re.search(
                r"MBytes  ([+-]?(?=\.\d|\d)(?:\d+)?(?:\.?\d*))(?:[Ee]([+-]?\d+))? Mbits/sec",
                list[-3],
            )
            # DEBUG
            # print(output)
            # print(list[-3])

            if sender_bitrate:
                try:
                    return float(sender_bitrate.group(1))
                except (ValueError, AttributeError) as e:
                    print(Fore.RED + f"[-] Error parsing the bitrate: {e}")
        else:
            print(
                Fore.RED
                + f"[-] Error executing the iperf3 command. Error message: {result.stderr}"
            )
    print(Fore.RED + f"[-] All 4   attempts failed.")
    return None


def dump(options, protocol, receive, file):
    # Set vars

    try:
        dump = subprocess.Popen(
            [
                "sudo",
                "tcpdump",
                "-i",
                options.interface,
                "-w",
```

```
                options.filename + file,
                protocol,
            ],
            stdin=subprocess.PIPE,
        )
        dump.stdin.write(b"password\n")
        dump.stdin.flush()

        # Run iperf3 test
        port = ""
        if options.port != None:
            port = "-p " + str(options.port)

        result = []
        # extract relevant data
        for i in range(10):
            result.append(perf(options, protocol, receive, port))
            print(Fore.GREEN + f"[{i}] Sender Bitrate: {result[-1]}" + Style.RESET_ALL)

        Min = min(result)  # MIN
        Max = max(result)  # MAX
        mean_value = sum(result) / len(result)  # MEAN
        sum_squared_diff = sum((x - mean_value) ** 2 for x in result)
        std_dev_value = math.sqrt(sum_squared_diff / (len(result) - 1))
        return Min, Max, mean_value, std_dev_value

    except Exception as e:
        print(Fore.RED + f"[-] Error during Dumping: {e}" + Style.RESET_ALL)

    finally:
        dump.terminate()
        print(
            Fore.GREEN
            + f"[+] --> DUMP Made Correctly for the file : {options.filename}{file}"
            + Style.RESET_ALL
        )


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-f", "--file", dest="filename", help="write report to FILE")
    parser.add_argument(
        "-i", "--interface", dest="interface", help="write the interface"
    )
    parser.add_argument("-a", "--address", dest="address", help="write the address")
    parser.add_argument("-p", "--port", dest="port", help="write the port number")

    options = parser.parse_args()

    current_datetime = datetime.now()
    formatted_datetime = current_datetime.strftime("%Y-%m-%d %H:%M:%S")

    print(Fore.BLUE + f"[+] TCP test" + Style.RESET_ALL)
    Min_TCP, Max_TCP, avg_TCP, std_dev_TCP = dump(
        options, protocol="tcp", receive="", file="-TCP"
    )
    print(Fore.BLUE + f"[+] UDP test" + Style.RESET_ALL)
    Min_UDP, Max_UDP, avg_UDP, std_dev_UDP = dump(
        options, protocol="udp", receive="", file="-UDP"
    )
    print(Fore.BLUE + f"[+] TCP test with receive instead of sending" + Style.RESET_ALL)
    Min_TCP_R, Max_TCP_R, avg_TCP_R, std_dev_TCP_R = dump(
        options, protocol="tcp", receive="-R", file="-TCP-R"
    )
    print(Fore.BLUE + f"[+] UDP with receive instead of sending" + Style.RESET_ALL)
    Min_UDP_R, Max_UDP_R, avg_UDP_R, std_dev_UDP_R = dump(
        options, protocol="udp", receive="-R", file="-UDP-R"
    )

    print(Fore.BLUE + f"|----------- FINAL RESULT -----------|")
    print(Fore.BLUE + f"|       | Avg  | Min  | Max  | StdD  |")
    print(
        Fore.BLUE
        + f"|  TCP  | {avg_TCP:.1f} | {Min_TCP:.1f} | {Max_TCP:.1f} | {std_dev_TCP:.1f}   |"
    )
    print(
        Fore.BLUE
        + f"|  UDP  | {avg_UDP:.1f} | {Min_UDP:.1f} | {Max_UDP:.1f} | {std_dev_UDP:.1f}   |"
```

```
    )
    print(
        Fore.BLUE
        + f"| TCP_R | {avg_TCP_R:.1f} | {Min_TCP_R:.1f} | {Max_TCP_R:.1f} | {std_dev_TCP_R:.1f}   |"
    )
    print(
        Fore.BLUE
        + f"| UDP_R | {avg_UDP_R:.1f} | {Min_UDP_R:.1f} | {Max_UDP_R:.1f} | {std_dev_UDP_R:.1f}   |"
    )
    print(Style.RESET_ALL)

    with open(f"RESULT-{options.filename}.txt", "w") as file:

        file.write(formatted_datetime + "\n")


        file.write(f"|----------- FINAL RESULT -----------|\n")
        file.write(f"|       | Avg  | Min  | Max  | StdD  |\n")
        file.write(
            f"|  TCP  | {avg_TCP:.1f} | {Min_TCP:.1f} | {Max_TCP:.1f} | {std_dev_TCP:.1f}   |\n"
        )
        file.write(
            f"|  UDP  | {avg_UDP:.1f} | {Min_UDP:.1f} | {Max_UDP:.1f} | {std_dev_UDP:.1f}   |\n"
        )
        file.write(
            f"| TCP_R | {avg_TCP_R:.1f} | {Min_TCP_R:.1f} | {Max_TCP_R:.1f} | {std_dev_TCP_R:.1f}   |\n"
        )
        file.write(
            f"| UDP_R | {avg_UDP_R:.1f} | {Min_UDP_R:.1f} | {Max_UDP_R:.1f} | {std_dev_UDP_R:.1f}   |\n"
        )

    print(Fore.GREEN + f"[+] --> RESULT SAVE ON : {options.filename}" + Style.RESET_ALL)


if __name__ == "__main__":
    main()
```

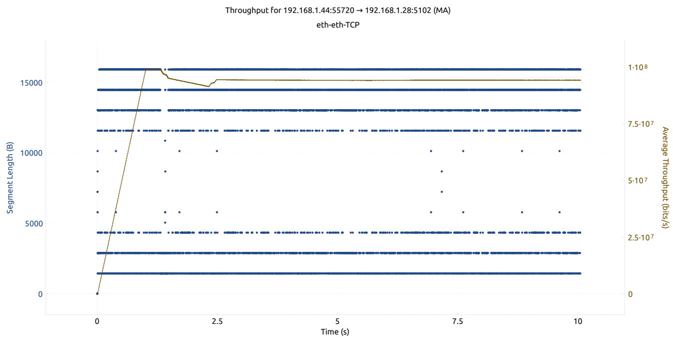

Figure 7: Throughput Plot TCP of Eth-Eth

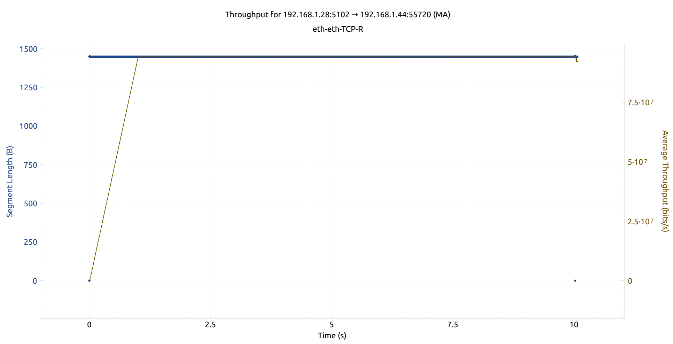

Figure 8: Throughput Plot TCP-Reverse of Eth-Eth

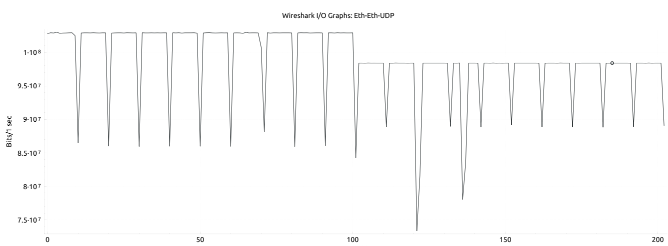

Figure 9: I/O UDP Regular and Reverse of Eth-Eth

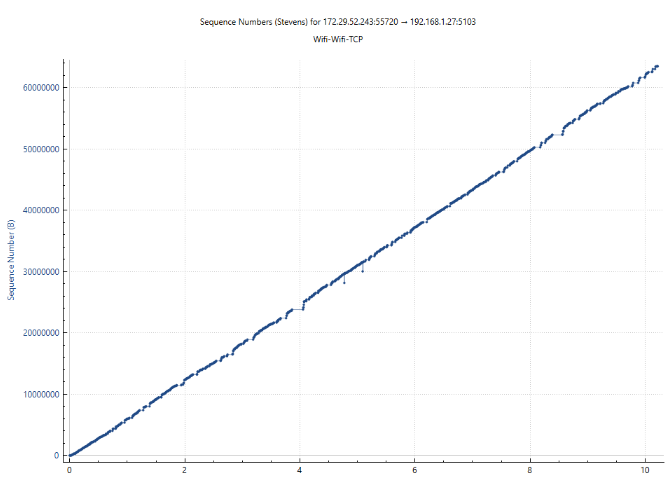

Figure 10: Stevens TCP Plot TCP of Wifi-Wifi

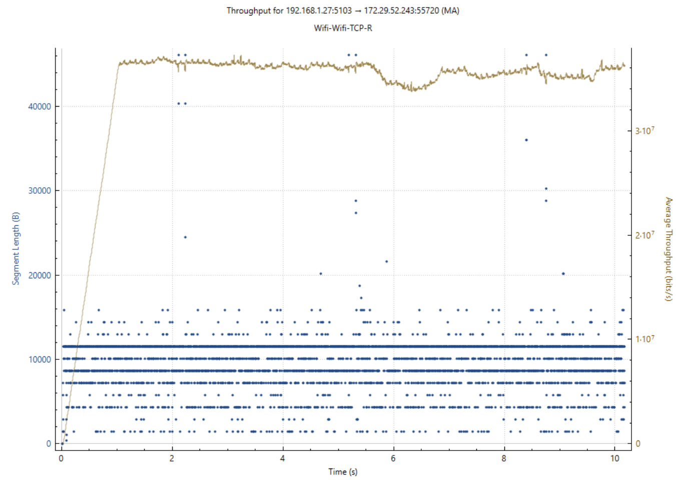

Figure 11: Throughput Plot TCP-Reverse of Wifi-Wifi

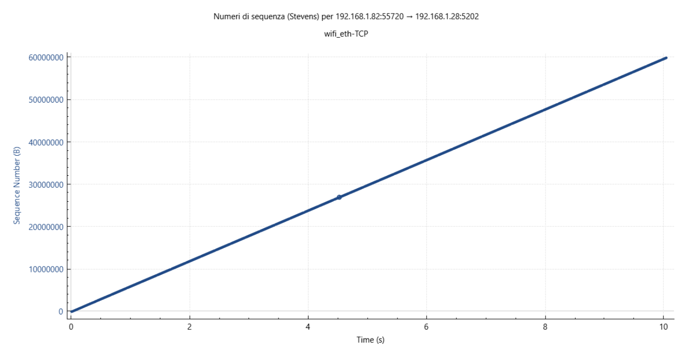

Figure 12: Sequence Number TCP of Wifi-Eth

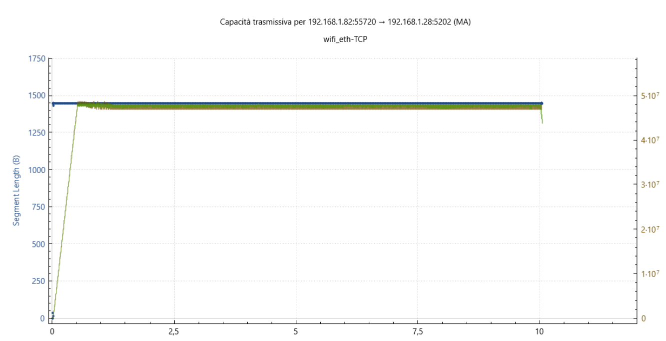

Figure 13: Throughput TCP of Wifi-Eth

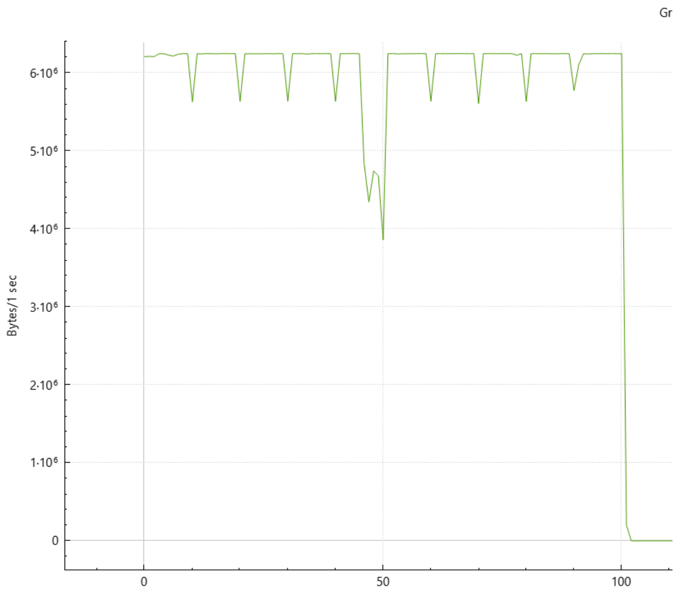

Figure 14: I/O UDP of Wifi-Eth