WiFi performance

s322918, s331461, s333996

ACM Reference Format:
s322918, s331461, s333996. 2025. WiFi performance. In *Proceedings of WiFi performance Report*. ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

WiFi, the most popular technology in Wireless Local Area Networks (WLANs), has become an integral part of modern communication. In this laboratory session, we focused on understanding the performance of WiFi networks, specifically considering the infrastructure-based mode. In this mode, devices communicate through a central wireless access point (AP), which manages traffic and ensures connectivity.
However, WiFi links have their limitations, including signal range, interference, and bandwidth constraints, which can impact overall network performance. By using tools like Wireshark, we aim to capture and analyze network traffic to gain insights into these limitations and improve our understanding of WiFi performance.

2 LAB SETUP AND SCENARIOS

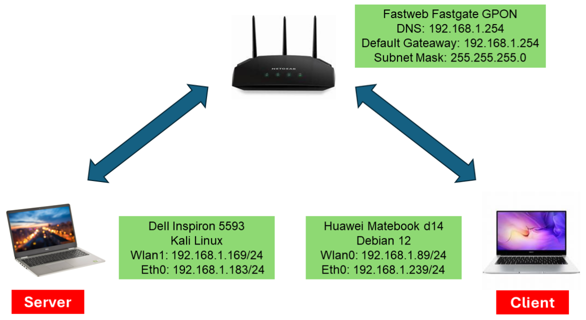


Figure 1: Setup

The Figure 1 shows our setup. The server uses a Qualcomm QCA9377 wireless network chipset. On Linux, support for this chipset is generally included in the kernel. However, in our case, it was necessary to install the firmware package for Qualcomm Atheros, running the following commands in the terminal:

```
sudo apt update
sudo apt ./firmware-atheros_20230625-2_all.deb
```

The ath10k driver installed, however, always reports a TX rate of 6 Mbit/s (even if the actual one is different). As we found out by typing in the terminal:

```
iw dev wlan0 link
```

In searching for information online, this is due to limitations,in terms of privacy, of the driver. Therefore, the solution was to use a TP-Link Archer T3U Plus USB network adapter on the server. The client is equipped with a Realtek RTL8822CE Wi-Fi network card. The channel used was the 104th, at a frequency of 5.52 GHz, the security protocol in place was WPA2 and the WiFi protocol version is 802.11ac.

Capacity (Mbit/s)	WiFi	Ethernet
Client	430	1000, Full duplex
Server	700	100, Full duplex

Table 1: Table Capacity

We have considered various scenarios, connecting the stations to the access points via Ethernet or Wifi, and analyzing individual flows between client and server, both ways, to extract performance using TCP or UDP at the transport layer:

- **Both WiFi**
- **Mixed mode**: Server is Ethernet, client is WiFi
- **Both ethernet**

3 TOOLS AND THEORY

In this section, goodput prediction and iperf3 parameters with tuning will be reported. Firstly, we need to specify the efficiency parameters used in the different experiments:

- In both WiFi scenarios, the bottleneck is represented by the client with a 430 Mbit/s limit. Given the complexity of the calculation, we assume that the efficiency parameters in WiFi conditions are $\mu_{TCP} = 0.5$ and $\mu_{UDP} = 0.55$.
  $G_{TCP} = \frac{1}{2}\mu_{TCP}C_{430} = 108.3\text{Mbit/s}$
  $G_{UDP} = \mu_{UDP}C_{430} = 118.2\text{Mbit/s}$
- In Mixed mode bottleneck will be the server with ethernet connection at 100 Mbit/s and the efficiency parameters $\mu_{TCP(Eth)} = 0.942$ and $\mu_{UDP(eth)} = 0.96$
  $G_{TCP} = \mu_{TCP(Eth)}C_{100} = 94.2\text{Mbit/s}$
  $G_{UDP} = \mu_{UDP(Eth)}C_{100} = 95.7\text{Mbit/s}$
- In both ethernet mode, like the previous one, bottleneck is represented by server and the paramters will be the same as mixed mode.
  $G_{TCP} = \mu_{TCP(Eth)}C_{100} = 94.2\text{Mbit/s}$
  $G_{UDP} = \mu_{UDP(Eth)}C_{100} = 95.7\text{Mbit/s}$

It's important to highlight, for future explanations, that the TCP measurements was collected in a different day then UDP.

Regardig iperf3 tool with UDP, it was crucial to exploit various parameters like -b option to set bitrate and -l for the lenght of the packet. In particurarly, on the WiFi-to-WiFi connection, we initially decided to perform a 'parameter tuning' on the bitrate value, followed by adjustments to the packet length. We conducted 10 iperf tests for each of the following bitrate values (Mbit/s): 0 (no limit), 50, 70, and 100, obtaining the results outlined below (Table 2).

b = 0	SENDER	RECEIVER	LOSS %
AVG	108.75	76.03	34.5
STD	3.69	13.72	4.43
b = 100	SENDER	RECEIVER	LOSS %
AVG	99.95	93.1	6.08
STD	0.1	2.95	2.68
b = 50	SENDER	RECEIVER	LOSS %
AVG	50	49	1.09
STD	0	0.22	0.9
b = 70	SENDER	RECEIVER	LOSS %
AVG	70	68.15	2.25
STD	0	0.71	0.6

Table 2: UDP Test Results (b = 0/50/70/100)

Based on these tests, we selected a target bitrate value of 70 Mbit/s, as it provided an optimal balance between bitrate and packet loss. We finally decided to set the packet length to 1472 bytes because it's the Maximum Segment Size (MSS) for UDP packets. In UDP, fragmentation doesn't occur at Layer 4; rather, it's handled at the IP level. Consequently, if a packet's length exceeds the MSS, it could potentially lead to a slower connection, as we saw by putting the l parameter to 1473. This is because if a fragment of the message is lost, the entire message must be resent.

b = 70 / L = 1472	SENDER	RECEIVER	LOSS %
AVG	70	68.52	1.28
STD	0	0.21	0.15

Table 3: UDP Test Results (b = 70 / L = 1472)

Another important aspect to highlight is that the *t_start* value, as calculated by Wireshark on the sender side, refers to the time when the packet is captured by Wireshark. This means that if the buffer on Layer 2 is full, packets will be sent at different times. Consequently, we will always use the measurements taken on the receiver side, as they are more consistent.

4 RESULTS

4.1 WiFi-WiFi

4.1.1 ***TCP.*** During the WiFi-to-WiFi TCP configuration test, we observed a goodput value significantly lower than the theoretical one, particularly in case C, where we found a value approximately 80% lower (Figure 16). After analyzing the flows executed with iperf3 using Wireshark, we noticed a high number of "TCP WINDOW FULL" packets, which could indicate network congestion. This hypothesis is supported by the graph in Figure 2, where it is evident that the MTU (Maximum Transmission Unit) of the packets frequently changes, often reaching much smaller values compared to the standard of 1500 bytes. Frequent changes to a smaller MTU can lead to increased overhead and reduced efficiency, contributing to the observed lower goodput.

In case D (reverse), we also observed a goodput lower than the theoretical one, with high variance in the measurements. By analyzing the individual iperf3 tests, we noticed that one test performed excellently, while the other nine did not. Specifically, we compared this high-performing flow with another from the nine, and observed the following:

From the sequence number per packet graph, as seen in the graph in Figure 3, the flow is almost linear without interruptions. In contrast, the graph in Figure 4 shows frequent interruptions where the client is waiting for ACKs.

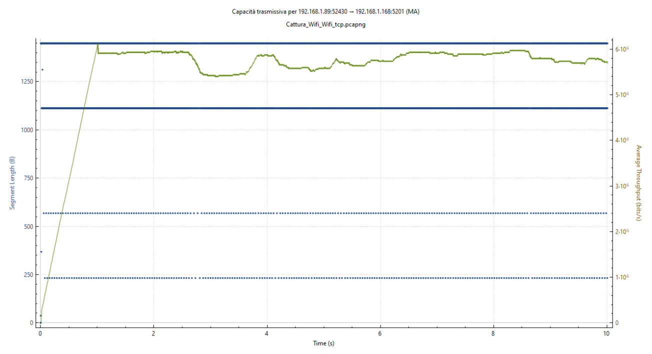


Figure 2: Goodput TCP

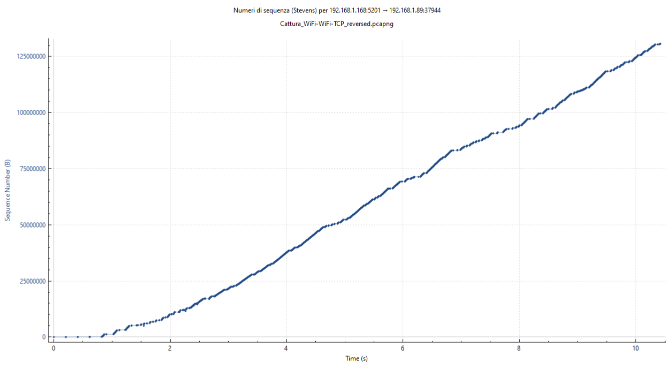


Figure 3: Sequence number per second TCP

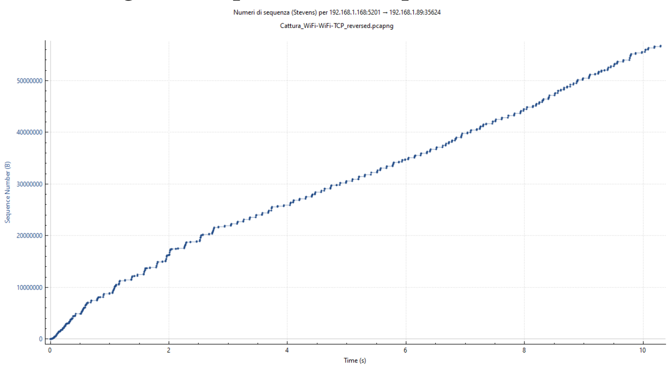


Figure 4: Sequence number per second TCP reverse

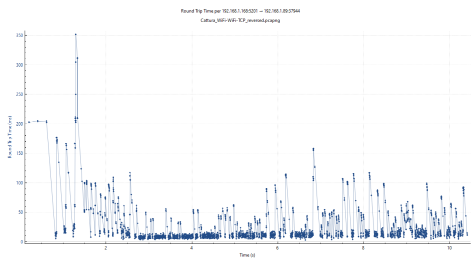


Figure 5: RTT, TCP

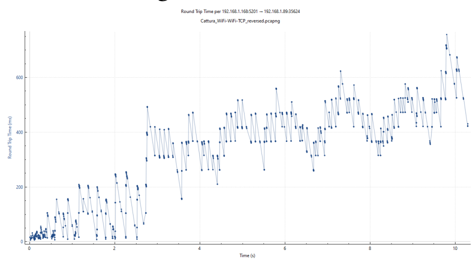


Figure 6: RTT, TCP reverse

From the round trip time graph, we see that in the best iperf3 test, the values are close to 0 ms for most of the time. Conversely, in the graph in Figure 6, the average round trip time is about 400 ms.

These analyses suggest that in the best iperf3 test, the network conditions were optimal. However, in the other tests, network congestion negatively affected the goodput (like in the case C, Figure 16).

4.1.2 ***UDP****.* In the iperf3 tests using the UDP protocol, we obtained results close to our expectations in both cases C and D (Figure 17). These results can be attributed to the following factors:

- The network was not congested, unlike in the TCP case.
- The selection of a bandwidth that optimized the packet rate relative to packet loss.

It is important to note that the theoretical goodput (without setting the -b parameter in iperf3) would have been 118.25 Mbit/s (Client's bottleneck, Table 2), which would not have led to an optimal result, due to a packet loss average of 34,5% , very large than the one with the optimal parameters (Table 3).

4.2 WiFi-Ethernet

In this scenario, the client was connected via Wi-Fi to the AP, while the server was using Ethernet.

4.2.1 ***TCP****.* The test results were quite consistent with the theory, except for case E of TCP tests (Figure 16), therefore, we investigated further using Wireshark. The graphs of goodput and throughput coincided and the value of the segment length remained fairly constant to 1448 bytes.

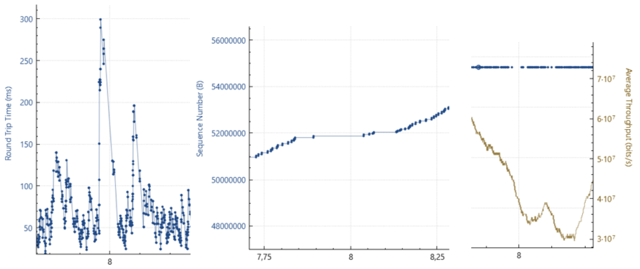


Figure 7: Mixed mode, TCP

We found that in several flows there were time intervals where the goodput value dropped below the theoretical value, due to ACKs arriving late or not at the same rate at which the client transmits, as shown in the graphs in Figure 7, the round trip time exhibits spikes, and there are stalls in the sequence number graph.
This problem may stem from packets being queued in router buffers, leading to notable delays or network congestion. We find it unlikely that the delayed ACKs are a result of collisions, especially given that the RTS/CTS mechanism was enabled.
As demonstrated by the 7 Mbit/s standard deviation, the goodput varied between each run.

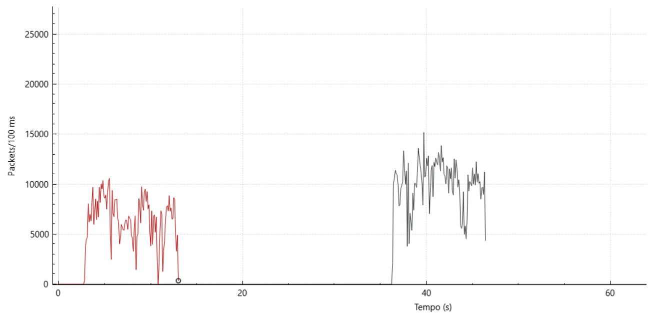


Figure 8: I/O Graph, Flow Differences, Packets/100ms

In Figure 8, the difference in packets sent every 100 ms by the client between the initial flow and the flow that achieved maximum goodput is evident.
Subsequently, we changed the direction of the TCP flows and observed improvements in performance computations, as evidenced also by the Figure 9. This illustrative flow shows noticeable differences compared to the previous direction: the average RTT is lower, the Stevens curve is linear, and the goodput averages 95 Mbit/s.

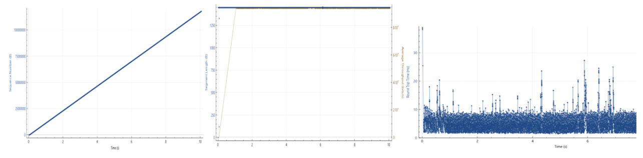
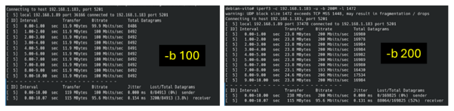


Figure 9: Mixed mode, TCP reversed

The WiFi channel is not symmetrical, the AP might perceive a higher capacity or better conditions for transmission compared to the client, due to various factors, such as differences in hardware capabilities, signal strength or interference. This asymmetry resulted in better performance with reversed TCP.

4.2.2 ***UDP.*** Different target bitrates led to varying percentages of packet loss, as shown in the Figure 10. The client's network card can reach 200 Mbit/s, however, the AP-server link has a capacity of 100 Mbit/s, the excess packets cannot be accommodated and are consequently dropped. We decided to use -b 100 considering the Table 1.



```
Connecting to host 192.168.1.183, port 5201
[  5] local 192.168.1.89 port 36166 connected to 192.168.1.183 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec  11.9 MBytes  99.9 Mbits/sec  8486
[  5]   1.00-2.00   sec  11.9 MBytes   100 Mbits/sec  8492
[  5]   2.00-3.00   sec  11.9 MBytes   100 Mbits/sec  8492
[  5]   3.00-4.00   sec  11.9 MBytes   100 Mbits/sec  8492
[  5]   4.00-5.00   sec  11.9 MBytes   100 Mbits/sec  8491
[  5]   5.00-6.00   sec  11.9 MBytes   100 Mbits/sec  8492
[  5]   6.00-7.00   sec  11.9 MBytes   100 Mbits/sec  8492
[  5]   7.00-8.00   sec  11.9 MBytes   100 Mbits/sec  8492
[  5]   8.00-9.00   sec  11.9 MBytes   100 Mbits/sec  8492
[  5]   9.00-10.00  sec  11.9 MBytes   100 Mbits/sec  8492
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.00  sec   119 MBytes   100 Mbits/sec  0.000 ms  0/84913 (0%)  sender
[  5]   0.00-10.07  sec   115 MBytes  95.6 Mbits/sec  0.154 ms  3200/84913 (3.8%)  receiver
```

```
debian-vito# iperf3 -c 192.168.1.183 -u -b 200M -l 1472
warning: UDP block size 1472 exceeds TCP MSS 1448, may result in fragmentation / drops
Connecting to host 192.168.1.183, port 5201
[  5] local 192.168.1.89 port 37478 connected to 192.168.1.183 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec  23.8 MBytes   200 Mbits/sec  16980
[  5]   1.00-2.00   sec  23.8 MBytes   200 Mbits/sec  16978
[  5]   2.00-3.00   sec  23.8 MBytes   200 Mbits/sec  16984
[  5]   3.00-4.00   sec  23.8 MBytes   200 Mbits/sec  16984
[  5]   4.00-5.00   sec  23.8 MBytes   200 Mbits/sec  16982
[  5]   5.00-6.00   sec  23.8 MBytes   200 Mbits/sec  16989
[  5]   6.00-7.00   sec  23.8 MBytes   200 Mbits/sec  16980
[  5]   7.00-8.00   sec  23.1 MBytes   193 Mbits/sec  16430
[  5]   8.00-9.00   sec  24.6 MBytes   206 Mbits/sec  17534
[  5]   9.00-10.00  sec  23.8 MBytes   200 Mbits/sec  16984
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.00  sec   238 MBytes   200 Mbits/sec  0.000 ms  0/169825 (0%)  sender
[  5]   0.00-10.07  sec   115 MBytes  95.6 Mbits/sec  0.131 ms  88064/169825 (52%)  receiver
```

Figure 10: Set target bitrate

In Wireshark, we observed that our choice of parameters is optimal. In both transmission directions, we have low packet loss and a nearly constant goodput (Figure 11), close to the theoretical value (Figure 17).

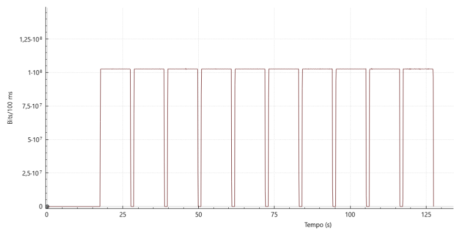


Figure 11: I/O Graph, Bits/100ms

4.3 Ethernet-Ethernet

4.3.1 ***TCP.*** In this scenario, we connect the devices to the AP using two ethernet links CAT.5e which support 1000Mbit/s.In contrast to Wifi-Wifi mode, the client and server rely on a full duplex communication but with different supported bandwidth, due to different ethernet interfaces. As a consequence the server will be the bottleneck (in terms of bit rate).

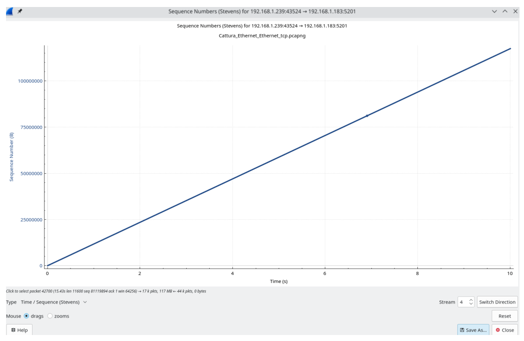


Figure 12: Eth-Eth TCP session

Even though Stevens graph (Figure 12) shows a linear behaviour that infer no retransmission in the measurements, the results on performance are quite different. In fact, when the client sends data RTT is higher (around 2.5 ms on Figure 13) compared to RTT when the servers sends (near to 0 ms). The reason is that when the client sends data at a higher rate, it can cause congestion on the router, resulting in a queuing of numerous packets, potentially leading to delays, increased Round-Trip Time (RTT), and frequent fluctuations.

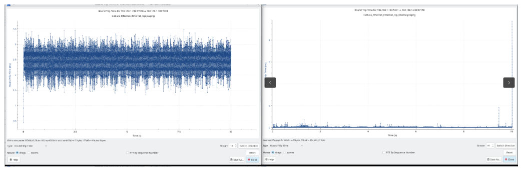


Figure 13: (Sx)C->S, (Dx)S->C

In Figure 14 we can observe a considerable difference in the length of the packets. Specifically, during the Client-to-Server connection, the packets exceed the standard MTU size due to Offloading implementation. It's a technique that exploits specialized hardware to do fragmentation. This can reduce CPU load, just because the fragmentation at TCP level is done by CPU, and improve network throughput. The settings of the interface are shown on Figure 15.

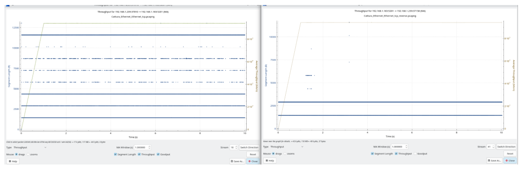
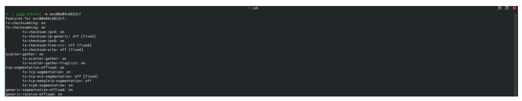


Figure 14: (Sx)C->S,(Dx)S->C



```
→  ~ sudo ethtool -k enx00e04c6813cf
Features for enx00e04c6813cf:
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: on
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: on
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: on
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp-ecn-segmentation: off [fixed]
        tx-tcp-mangleid-segmentation: off
        tx-tcp6-segmentation: on
generic-segmentation-offload: on
generic-receive-offload: on
```

Figure 15: Offloading on

4.3.2 ***UDP.*** Similar to our approach in Ethernet-Wifi mode, we optimized parameters to achieve peak performance with a bandwidth setting of -b 100, reflecting the bottleneck conditions. It's important to highlight that in Ethernet-Ethernet mode, the performance is symmetrical.

Upon analyzing packets in Wireshark, we found no evidence of issues or packet loss during the measurement period.

5 CONCLUSIONS

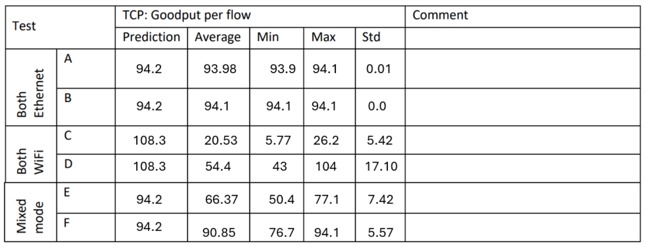
In scenarios where an Ethernet link was used, we measured values close to the theoretical maximums. However, in the WiFi-to-WiFi scenario, we observed that network congestion and asymmetry of WiFi channels significantly impacted the goodput. This highlights the challenges in achieving optimal performance in wireless networks compared to wired connections.

6 APPENDIX

```
#!/bin/bash
#TCP SESSION
output_file="Cattura_Ethernet_Wifi_tcp.txt"
num_iterations=10
test_duration=10
for ((i=1; i<=$num_iterations; i++))
do
    echo "Esecuzione del test $i di $num_iterations..."
    # greppando solo i Mbit/s
    # usa -R option per eseguire il test inverso
    iperf_output=$(iperf3 -c 192.168.1.183 -t $test_duration
         | grep -i "bits/sec" | awk '{print $7}')
    echo "$iperf_output" >> $output_file
    sleep 1
done
echo "Test completati. I risultati sono stati scritti in
     $output_file"
```

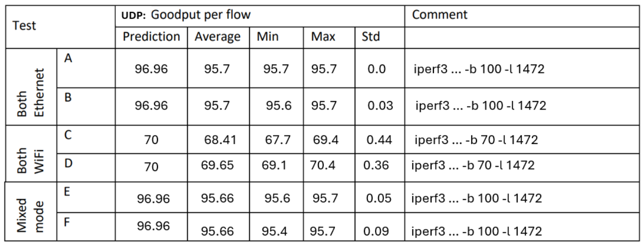
```
#!/bin/bash
# UDP SESSION
output_file="Cattura_Wifi_Ethernet_UDP100M_reverse.txt"
num_iterations=10
test_duration=10

for ((i=1; i<=$num_iterations; i++))
do
    echo "Esecuzione del test $i di $num_iterations..."
    # usa -b per settare il bitrate e -l per settare la
         lunghezza del pacchetto
    iperf_output=$(iperf3 -c 192.168.1.183 -b 100M -u -l
         1472 -R -t $test_duration | grep -i "bits/sec" |
         awk '{print $7}')
    echo "$iperf_output" >> $output_file
    sleep 1
done
echo "Test completati. I risultati sono stati scritti in
     $output_file"
```



Test		TCP: Goodput per flow					Comment
		Prediction	Average	Min	Max	Std	
Both Ethernet	A	94.2	93.98	93.9	94.1	0.01	
	B	94.2	94.1	94.1	94.1	0.0	
Both WiFi	C	108.3	20.53	5.77	26.2	5.42	
	D	108.3	54.4	43	104	17.10	
Mixed mode	E	94.2	66.37	50.4	77.1	7.42	
	F	94.2	90.85	76.7	94.1	5.57	

Figure 16



Test		UDP: Goodput per flow					Comment
		Prediction	Average	Min	Max	Std	
Both Ethernet	A	96.96	95.7	95.7	95.7	0.0	iperf3 ... -b 100 -l 1472
	B	96.96	95.7	95.6	95.7	0.03	iperf3 ... -b 100 -l 1472
Both WiFi	C	70	68.41	67.7	69.4	0.44	iperf3 ... -b 70 -l 1472
	D	70	69.65	69.1	70.4	0.36	iperf3 ... -b 70 -l 1472
Mixed mode	E	96.96	95.66	95.6	95.7	0.05	iperf3 ... -b 100 -l 1472
	F	96.96	95.66	95.4	95.7	0.09	iperf3 ... -b 100 -l 1472

Figure 17