Analyzing Goodput in Different Network Scenarios: Ethernet vs Wi-Fi

Conforto Francesco Paolo, D'Amico Dennis, Drago Giovanni, Lo Tufo Stefano

1 GOALS OF THE LABORATORY

This study focuses on evaluating the performance characteristics of different network scenarios, ranging from wired (Ethernet) to wireless (WiFi) connections, through the application of TCP and UDP protocols. The experiment aims to study how these protocols respond to different network scenarios. By systematically analyzing their behavior, we seek to quantify their efficiency, reliability, and overall effectiveness in data transmission.
For each test, we will compute the theoretical Goodput, which quantifies the amount of useful application-layer data successfully transmitted over the network, excluding interference/congestion, retransmissions, and control messages. This theoretical value will be compared against the measured Goodput obtained from practical experiments to assess discrepancies between ideal and real-world performance.

2 THEORY & EXPECTED RESULT

In network communication, understanding the efficiency and effectiveness of data transmission is essential for optimizing performance.
Two critical metrics in this context are Efficiency and Goodput, together, these metrics provide a comprehensive view of both the actual data delivery and the overall performance of the network. This section explores these two concepts, providing their definitions and formulas.

2.1 Efficiency

Transmission efficiency (η) is a measure of the proportion of useful application-layer data successfully transmitted relative to the total amount of data sent. It quantifies how efficiently the available bandwidth is utilized for delivering actual payload data.
Follows the formula used to calculate the efficiency:

$$\eta = \frac{Data_{\text{L7}}}{Data_{\text{L7}} + Overhead_{\text{L4}} + Overhead_{\text{L3}} + Overhead_{\text{L2}}} \quad (1)$$

- $Data_{\text{L7}}$ refers to the application layer data (Layer 7 of the OSI model). It is the useful data that is passed from the application (such as a file, a message, or any other form of application data) that needs to be transmitted across the network.
- $Overhead_{\text{L4}}$ refers to the transport layer overhead (Layer 4 in the OSI model), which includes the headers and control information for protocols like TCP or UDP.
- $Overhead_{\text{L3}}$ refers to the network layer overhead (Layer 3, OSI model) provided by the IP header. This header, which includes source/destination IPs, TTL, protocol, and checksum, is essential for routing packets.
- $Overhead_{\text{L2}}$ refers to the Data Link Layer overhead (Layer 2 in the OSI model), which includes the frame header and trailer for both Ethernet and WiFi technologies.

2.2 Goodput

Goodput is a key performance metric that measures the rate of useful data delivered to the application layer, excluding retransmissions, overhead, and control messages. Follows the formula to calculate the Goodput:

$$G = \frac{D}{T} \quad (2)$$

- D is the total amount of successfully received application-layer data (bits).
- T is the total transmission time (seconds).

A different way to express Goodput takes into account the efficiency (η) of the transmission and the channel capacity (C). This leads to the formula:

$$G = \eta \cdot C \quad (3)$$

- η is the efficiency of the transmission
- C is the channel capacity (i.e., the maximum possible transmission rate, typically given in bits per second).

2.3 Expected Results

Theoretical Goodput (**T.G.**) differs from practical Goodput due to various real-world factors that impact network performance. In theory, Goodput is calculated based on ideal conditions, in practice, issues such as network congestion, packet loss, retransmissions, and network interference can all reduce the actual Goodput. These factors introduce delays, reduce efficiency, and result in a lower practical Goodput than the theoretical value.

3 TOOLS

In this section we briefly see the commands used to perform the tests.

3.1 Iperf3

Iperf3 is a tool that, by generating traffic, estimates some key metrics to calculate the real average Goodput. By default, it records data every second on ten samples, then calculates the average Bitrate for both the sender and the receiver. The average Bitrate of the receiver is used as the real Goodput, as defined in the formula in the previous section.

3.2 The commands

$iperf3 - c < server_ip > -J - t\ 10$

With this command we can test the TCP connection between two computers for the duration of 10 seconds and output the result in JSON format.

Wireless Security Report, Torino, 2025
2025. ACM ISBN .
https://doi.org/

$iperf3 - c < server_ip > -J - t\ 10 - u - b < value > -l\ 1472$

With this command we can test the UDP connection between two computers for the duration of 10 seconds and also it permits to choose the bitrate value thanks to the -b flag. The -l flag permits to set the fragmentation size.

3.3 Wireshark

Wireshark is a widely used network protocol analyzer that captures and inspects data packets traveling through a network. In performance testing, it monitors protocol behavior under various conditions and loads.
Additionally, Wireshark enables the creation of custom graphs based on specific network parameters. For our tests, we used it to plot sequence numbers, throughput, and bitrate over time.

4 SCENARIOS AND RESULTS

This section provides a detailed analysis of three scenarios and the corresponding results. To perform the test we used the Script 1 attached in the appendix 1

4.1 Ethernet to Ethernet

For the first case of study we connected two Windows computers through an Ethernet cable with capacity **1 Gbps** as shown in the Figure 1. From this configuration, we expect a goodput of **949 Mbps** for TCP and **957 Mbps** for UDP as calculated in the formula (3), assuming that the efficiency changes according to the protocol. In the TCP case, the efficiency (η) is equal to 0,949 while in UDP is 0,957.

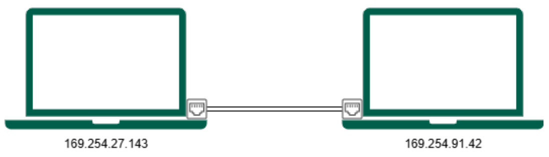

Figure 1: Ethernet to Ethernet configuration

Follows the table with the results obtained from the Iperf3 tests:

Protocol	T.G.	AVG	Max	Min	Dev Std
TCP	949	947.10	948.00	944.00	1.29
UDP t=5	957	926.00	934.00	915.00	9.13
UDP t=10	957	876.43	910.00	855.00	16.8

Table 1: Results of throughtput measurements in Mbps (ETH to ETH)

In the TCP scenario, the measured performance aligns closely with theoretical expectations, with an average bitrate of 947.10 Mbps compared to the theoretical 949 Mbps. In the UDP tests, two distinct behaviors were observed: one using test with a duration of 5 seconds and another one with a duration of 10 seconds. In the first test case, the transmission rate averaged around 954 Mbps; however, a packet loss of approximately 2.45% reduced the effective Goodput, likely due to receiver buffer overflow. In the second test case, instead, a notable issue occurred: after about seven seconds, the transmission rate dropped sharply from roughly 954 Mbps to between 750 and 800 Mbps, suggesting potential factors such as buffer overflows or misconfigurations.
This strange behavior is not present when a Linux and a Windows machine are connected to each other, where the average slightly exceeds 950 Mbps, always remaining below the theoretical Goodput.
In the following section, graphs will be presented to visually illustrate these differences and provide a clearer understanding of the behavior observed during the tests.

4.2 Wi-Fi to Wi-Fi

For the second case of study, we connected two Windows computers, as shown in Figure 2, through a Wi-Fi connection with a capacity of **400 Mbps**.

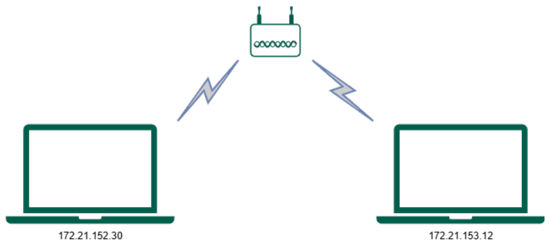

Figure 2: Wi-Fi to Wi-Fi configuration

From this configuration, we expect a goodput calculated as follows:

$$\text{Goodput} = 400 \times \eta_{\text{Wi-Fi}} \times \eta_{\text{TCP/IP or UDP/IP}} \times 0.5 \tag{4}$$

- **400 Mbps Factor**: This value represents the theoretical maximum capacity of the Wi-Fi link used in our experiment.
- $\eta_{\textbf{Wi-Fi}}$: We denote by Wi-Fi Efficiency the efficiency of data transmission at the MAC level of the **802.11ac** protocol, which in our case is equal to **0.8**.
- $\eta_{\textbf{TCP/IP or UDP/IP}}$: - In the case of **TCP/IP**, the Transport and Network Protocol Efficiency is calculated based on the formula (1):

$$\eta_{\text{TCP/IP}} = \frac{1460}{1460 + 20 + 20} = 0.9733 \tag{5}$$

  In the case of **UDP/IP**, the efficiency is slightly higher and is calculated following the expression (1) as:

$$\eta_{\text{UDP/IP}} = \frac{1460}{1460 + 8 + 20} = 0.9813 \tag{6}$$

- **0.5 Factor**: Each packet must be transmitted twice, once from the sender to the Access Point, and again from the Access Point to the receiver

Based on these values, we estimate a goodput of **155.73 Mbps** for TCP and **157.01 Mbps** for UDP.
Follows the table with the results obtained from the Iperf3 tests:

Protocol	T.G.	AVG	Max	Min	Dev Std
TCP	155.73	123.57	129.57	116.74	4.40
UDP	157.01	141.38	150.16	127.89	7.84

Table 2: Results of throughtput measurements in Mbps (Wi-Fi to Wi-Fi)

In both TCP and UDP scenarios, the measured goodput is slightly lower than the theoretical values. For TCP, the average measured bitrate was 123.57 Mbps versus a theoretical 155.73 Mbps. In the UDP scenario, the measured goodput was 141.38 Mbps compared to 157.01 Mbps theoretically. This reduction is likely due to packet loss from transient network conditions, such as congestion or interference, since the tests were performed on the university network.

4.3 Wi-Fi to Ethernet

For the third case of study we used two Windows computers. The sender was connected with the AP through an ethernet cable with capacity of **1 Gbps** while the receiver was connected through Wi-Fi with the AP as shown in the Figure 3.
From this configuration, we expect a goodput calculated as follows:

$$\text{Goodput} = 866.7 \times \eta_{\text{Wi-Fi}} \times \eta_{\text{TCP/IP or UDP/IP}} \qquad (7)$$

In this configuration, we consider the Wi-Fi Efficiency and the Transport and Network Protocol Efficiency to be the same as in the Wi-Fi to Wi-Fi configuration (η_{WiFi}=0.8, $\eta_{\text{TCP/IP or UDP/IP}}$=0.9733 or 0.9813). This choice was made because, given the network topology, the bottleneck is located at the device connected via Wi-Fi. From this configuration, we expect a goodput of **674,87** for TCP and **680.42** for UDP.

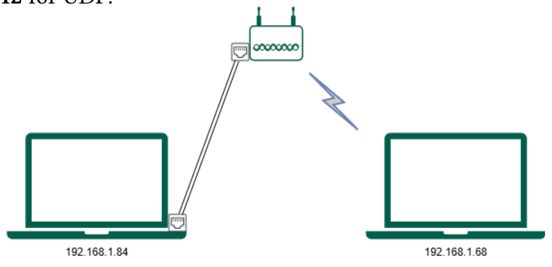

Figure 3: Wi-Fi to Ethernet configuration

Follows the table with the results obtained from the Iperf3 tests:

Protocol	T.G.	AVG	Max	Min	Dev Std
TCP	674.87	604.00	633.00	584.00	16.6
UDP	680.42	653.30	670.00	629.00	14.4

Table 3: Results of throughtput measurements in Mbps (Wi-Fi to Ethernet)

As shown, the measured TCP Goodput is consistently lower than the Theoretical Goodput (**T.G.**), likely due to the limited capacity of the wireless channel, leading to frequent retransmissions and packet loss. This congestion could explain the gap between the measured values and the theoretical prediction.
In the UDP scenario, the results are more aligned with the Theoretical Goodput, likely due to the absence of retransmissions, allowing for a much higher transmission speed. However, this comes at the cost of reliability, as an average packet loss of 30% was observed. This loss occurs because packets are transmitted at the maximum capacity of the Ethernet link before transitioning to a wireless channel, which, being inherently less efficient, cannot sustain the same performance. caption

5 GRAPHIC COMPARISON

This section compares the most significant metrics obtained using Wireshark. The graphs show two different parameters: Throughput and Bitrate. All other related graphs are provided in the Appendix (A.2, A.3).

5.1 Throughput

This section presents throughput graphs for three network scenarios: Ethernet-to-Ethernet (4), Wi-Fi-to-Wi-Fi (6), and Ethernet-to-Wi-Fi (5).

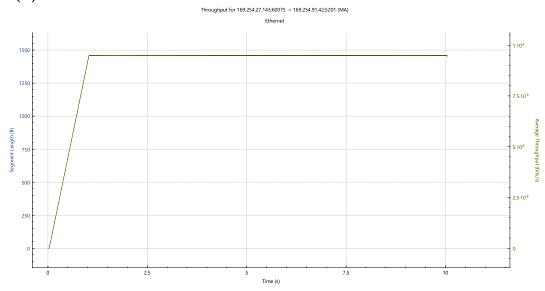

Figure 4: Ethernet to Ethernet throughput for TCP

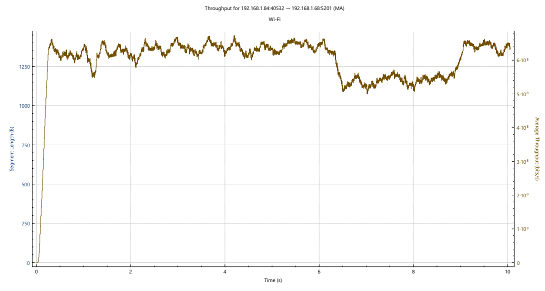

Figure 5: Ethernet to Wi-Fi throughput for TCP

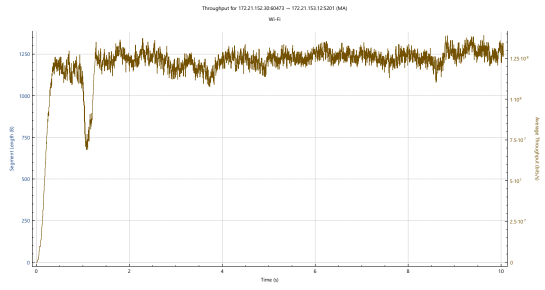

Figure 6: Wi-Fi to Wi-Fi throughput for TCP

- **Ethernet-to-Ethernet:** Throughput is generally high and stable due to the wired connection, which offers high capacity and minimal interference. (4)
- **Ethernet-to-Wi-Fi:** Throughput is noticeably lower compared to the wired scenario. This reduction is likely due to differences in channel capacity and efficiency, as well as interference and congestion that may cause TCP retransmissions. (5)
- **Wi-Fi-to-Wi-Fi:** Throughput measurements tend to be closer to the theoretical values, despite the inherent variability of wireless channels. (6)

5.2 Bitrate

This section presents bitrate graph comparisons for TCP and UDP in three network scenarios.

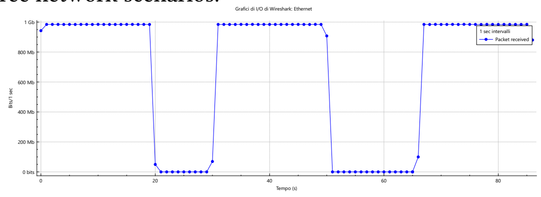

Figure 7: Ethernet to Ethernet bits per sec for TCP

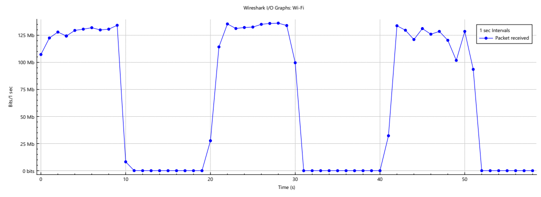

Figure 8: Wi-Fi to Wi-Fi bits per sec for TCP

The performance analysis of Ethernet-Ethernet (7) and Wi-Fi-Wi-Fi (8) in TCP transmission, reveals significant differences in stability and bitrate fluctuations.
Ethernet shows a stable trend with minimal variability, thanks to the reliable wired connection, low latency, and absence of interference. This results in consistent throughput.
In contrast, Wi-Fi experiences noticeable bitrate peaks and drops due to factors like wireless interference, congestion, and signal attenuation, leading to less stable throughput.
These differences confirm that Ethernet offers more predictable performance, while Wi-Fi, although more flexible, is more sensitive to fluctuations that can impact sensitive applications.

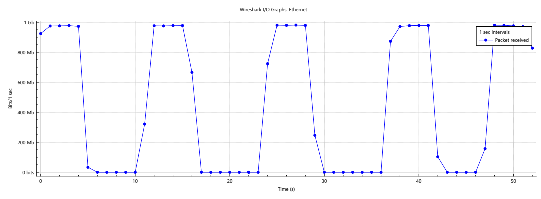

Figure 9: Ethernet to Ethernet bits per sec for UDP, 5 seconds case Windows-Windows

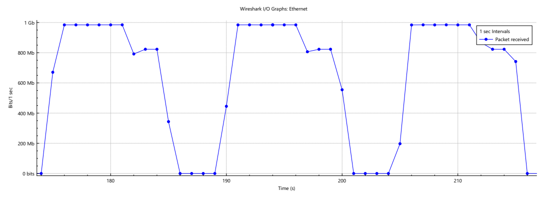

Figure 10: Ethernet to Ethernet bits per sec for UDP, 10 seconds case Windows-Windows

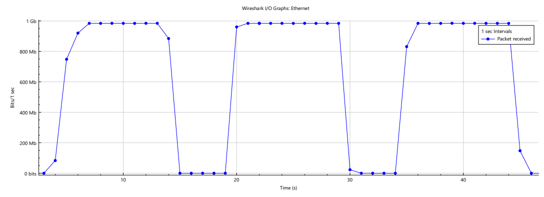

Figure 11: Ethernet to Ethernet bits per sec for UDP, 10 seconds case Ubuntu-Windows

The UDP performance was analyzed in three scenarios with different OS configurations and detection intervals, revealing key differences in stability and throughput.

- **Scenario 1**: Two Windows systems with a 5-second test show stable throughput. (9)
- **Scenario 2**: Two Windows systems with a 10-second test experience a drop to 750-800 Mbps, likely due to buffering or congestion management, possibly influenced by the use of the APIPA protocol for IP assignment. (10)
- **Scenario 3**: A Linux and a Windows system maintain throughput close to the expected goodput even with a 10-second test. In this case, IPs are assigned statically, leading to more efficient UDP handling on Linux. (11)

These results highlight that OS configuration, measurement interval, and APIPA usage in Windows-only setups significantly impact UDP performance. Linux systems mitigate bitrate drops seen in Windows-only configurations, likely due to differences in packet handling.

6 CONCLUSION

From the experiments that we have conducted we obtained interesting results to be analyzed:

- For Ethernet to Ethernet we had a stable bitrate for TCP while for UDP we experienced a significant drop of the trasmission rate in the ten seconds test.
- For Wi-Fi to Wi-Fi, we experienced a result more aligned with the theoretical one. The little discrepancy is probably caused by the network congestion o interferences, since the tests where performed into the university network.
- For the Ethernet to WiFi configuration, there is a significant discrepancy in TCP between the theoretical Goodput and the measured value. In contrast, UDP measurements are much closer to the theoretical value.

Overall, while some results aligned well with theoretical expectations, others deviated due to various influencing factors. These findings highlight the impact of network conditions, protocol characteristics, and transmission environments on performance, underscoring the need for further investigations to refine theoretical models and optimize network performance in heterogeneous environments.

A APPENDIX

A.1 Code

Follows the Python code used to perform the tests.

```python
import argparse
import subprocess
import json
import statistics
from tabulate import tabulate
from time import sleep
import sys

# Funzione per determinare il comando da utilizzare (.\iperf3.exe oppure iperf3)
def get_iperf_command():
    # Prova prima con .\iperf3.exe (Windows)
    try:
        subprocess.check_output([".\\iperf3.exe", "--version"], text=True, stderr=subprocess.STDOUT)
        return ".\\iperf3.exe"
    except Exception:
        # Se fallisce, usa iperf3
        return "iperf3"

def run_iperf_test(server_ip, protocol, bandwidth, iperf_cmd, num_runs=10):
    results = []

    for i in range(num_runs):
        # Comando base
        cmd = [iperf_cmd, '-c', server_ip, '-J', '-t', '10']  # test di 10 secondi

        if protocol == 'udp':
            cmd.extend(['-u', '-b', bandwidth, '-l', '1472'])  # impostazioni UDP
            sleep(2)  # Attesa di 2 secondi per evitare conflitti tra test

        log_line = f"\n[Esecuzione {i+1}/{num_runs}] {' '.join(cmd)}"
        print(log_line)
        output_lines.append(log_line)

        try:
            output = subprocess.check_output(cmd, text=True, stderr=subprocess.STDOUT)

            # Trova l'inizio del JSON nell'output
            json_start = output.find('{')
            if json_start != -1:
                output = output[json_start:]

            try:
                data = json.loads(output)

                # Estrai throughput medio dal ricevitore
                if protocol == 'tcp':
                    thr = data['end']['sum_received']['bits_per_second'] / 1e6  # Mbps
                elif protocol == 'udp':
                    thr = data['end']['sum']['bits_per_second'] / 1e6  # Mbps

                results.append(thr)
            except json.JSONDecodeError:
                error_msg = "    Errore decodifica JSON. Output ricevuto:\n" + output
                print(error_msg)
                output_lines.append(error_msg)

        except subprocess.CalledProcessError as e:
            error_msg = f"    Errore durante iperf3: {e.output}"
            print(error_msg)
            output_lines.append(error_msg)

    return results

def calculate_stats(data):
    if not data:
        return {'mean': 0, 'max': 0, 'min': 0, 'stddev': 0}

    return {
        'mean': statistics.mean(data),
        'max': max(data),
        'min': min(data),
        'stddev': statistics.stdev(data) if len(data) > 1 else 0
    }

if __name__ == "__main__":
    output_lines = []  # lista per salvare l'output da scrivere nel file

    parser = argparse.ArgumentParser(description='Esegui test iperf3 TCP/UDP')
    parser.add_argument('server_ip', help='IP del server iperf3')
    parser.add_argument('--bandwidth', default='400M', help='Larghezza di banda per il test UDP (es. 400M)')
    parser.add_argument('--protocol', choices=['tcp', 'udp', 'both'], default='both',
                        help="Protocollo per il test: 'tcp', 'udp', o 'both' (default   'both')")
    args = parser.parse_args()

    iperf_cmd = get_iperf_command()
    print(f"Utilizzo comando iperf3: {iperf_cmd}")
    output_lines.append(f"Utilizzo comando iperf3: {iperf_cmd}\n")

    results_summary = []

    # Test TCP
    if args.protocol in ['tcp', 'both']:
        print("    Esecuzione test TCP...")
        output_lines.append("    Esecuzione test TCP...\n")
        tcp_results = run_iperf_test(args.server_ip, 'tcp', args.bandwidth, iperf_cmd)
        stats_tcp = calculate_stats(tcp_results)
        results_summary.append(["TCP", f"{stats_tcp['mean']:.2f}", f"{stats_tcp['max']:.2f}",
                                f"{stats_tcp['min']:.2f}", f"{stats_tcp['stddev']:.2f}"])

    # Test UDP
    if args.protocol in ['udp', 'both']:
        print("\ n   Esecuzione test UDP...")
        output_lines.append("\ n   ␣Esecuzione␣test␣UDP
            ...\n")
        udp_results = run_iperf_test(args.server_ip, 'udp
            ', args.bandwidth, iperf_cmd)
        stats_udp = calculate_stats(udp_results)
        results_summary.append(["UDP", f"{stats_udp['mean
            ']:.2f}", f"{stats_udp['max']:.2f}",
                                f"{stats_udp['min']:.2f}"
                                    , f"{stats_udp['
                                    stddev']:.2f}"])

    # Creazione della tabella con i risultati
    table_header = ["Protocollo", "Media␣(Mbps)", "Max␣(
        Mbps)", "Min␣(Mbps)", "Dev␣Std"]
    table = tabulate(results_summary, headers=
        table_header, tablefmt="grid")

    results_output = "\ n   ␣Risultati:\n" + table
    print(results_output)
    output_lines.append(results_output)

    # Salvataggio dell'output in output.txt
    try:
        with open("output.txt", "w", encoding="utf-8") as
             f:
            f.write("\n".join(output_lines))
        print("\ n  ␣Output␣salvato␣in␣output.txt")
    except Exception as e:
        print(f"   ␣Errore␣durante␣il␣salvataggio␣dell'
            output:␣{e}")
```

Listing 1: Python script for iperf3

A.2 Sequence Number

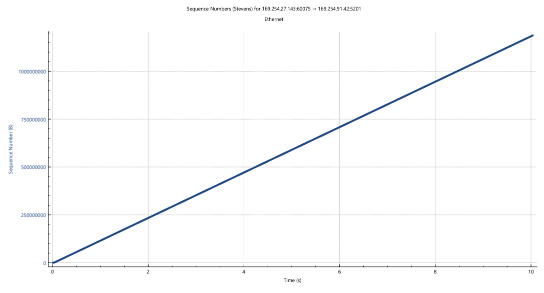

Figure 12: Ethernet to Ethernet sequence number for TCP

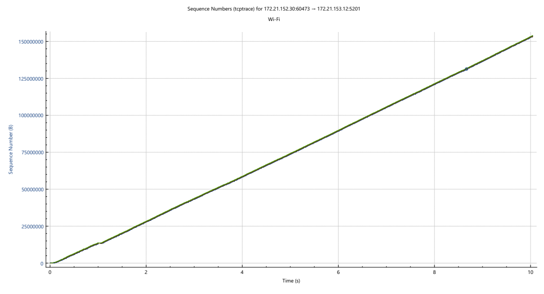

Figure 13: Wi-Fi to Wi-Fi sequence number for TCP

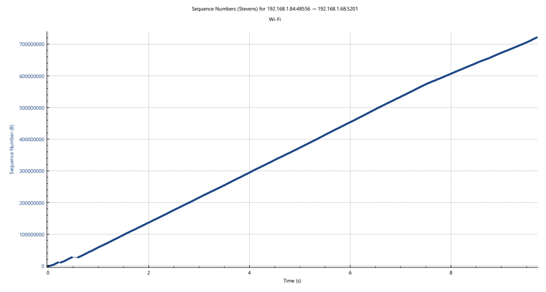

Figure 14: ETH to Wi-Fi sequence number for TCP

A.3 More graphs

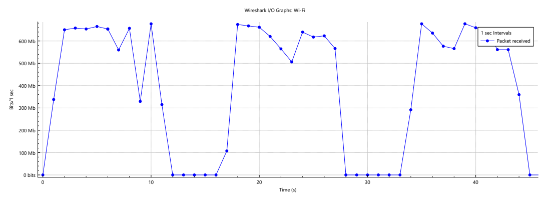

Figure 15: Eth to Wi-Fi bits per sec for TCP

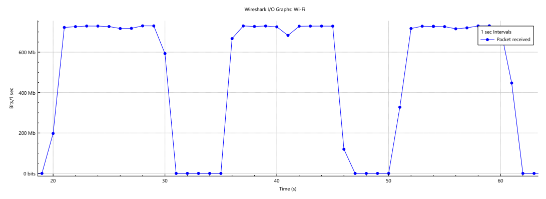

Figure 16: Eth to Wi-Fi bits per sec for UDP

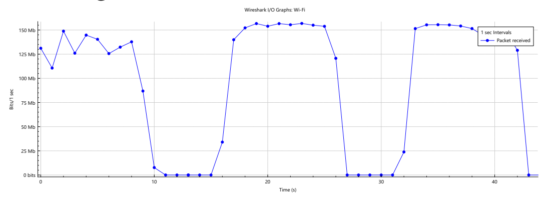

Figure 17: Wi-Fi to Wi-Fi bits per sec for UDP