Report for Lab on WiFi

Wi-Failing
Ciriaco Alifano, Gabriele Camisa, Simone Giammusso, Matteo Rizzo

1 INTRODUCTION

The goal of this experiment was to test the goodput in three different network scenarios: Ethernet, Wi-Fi, and Wi-Fi with an antenna placed in saltwater. The main goal was to predict the expected results beforehand by calculating the maximum theoretical goodput (TMG) (3.1.1) and then compare these predictions with the actual test results.

To perform the tests and analyze the results, we used a Python script (C), configured to run 10 iterations of each `iperf3` test, each one of 10 seconds each, first in normal mode (client sending data to the server) and then in reverse mode (client receiving data from the server). The script also captured the goodput on the receiver side, which is the key metric of interest in this study, and computed aggregated statistics over the 10 iterations, such as Average and Standard Deviation.

During each test, the script also captured packets traveling through the client's interface, generating 10 PCAP files per test. Throughout the report, we present various graphs, some of which are based on a single representative PCAP file; these were selected either because they reflect the general trend of the results or because they highlight a particularly relevant aspect of the experiment.

2 TOOLS AND DEVICES

- Hardware
  - Alice (Laptop #1)
    - USB Wi-Fi Adapter RTL8812AU [9] (2 TX x 2 RX | 802.11ac | 1200 Mbit/s | 2.4/5 GHz | 80 MHz)
    - TP-Link UE306 Ethernet USB 3.0 adapter [10] + Cat 8 Ethernet Cable
    - USB 3.0 | OS: Ubuntu 22.04 LTS
  - Bob (Laptop #2)
    - Intel Wireless-AC 9560 [6] (2 TX x 2 RX | 802.11ac | 1.73 Gbit/s | 2.4/5 GHz | 160 MHz)
    - Realtek 10/100/1000 GbE NIC + Cat 5e Ethernet Cable
    - USB 3.0 | OS: Ubuntu 24.04 LTS
  - Iliadbox Wi-Fi 6 Home Router [5] (2 TX x 2 RX @2.4 GHz | 4 TX x 4 RX @5 GHz | 802.11ax | 2x Ethernet 1 Gbit/s)
- Software
  - `iperf3` (v3.9) | Python (v3.10.12) | `tcpdump` (v4.99.1) + `libpcap` (v1.10.1, `TPACKET_V3`) | Wireshark (v4.4.3)

3 FORMULAS AND THEORY

3.1 Capacity, Throughput and Goodput definitions

- Capacity (C): The theoretical maximum data rate a network link can support, typically measured in Mbit/s or Gbit/s.
- Throughput (T): The total amount of data (including payload, retransmissions, and protocol overhead) successfully transmitted over a network in a given period.
- Goodput (G): The portion of throughput that considers only useful application data, excluding retransmissions and protocol overhead, transmitted in the period.

3.1.1 Theoretical Maximum Goodput (TMG). Even under ideal conditions (no packet losses, no idle time, no MAC layer fragmentation), the maximum goodput reachable is by definition lower than the maximum throughput, since it's affected by the unavoidable protocol overheads. In fact, the relation $G < T < C$ always holds.

In addition, the goodput you would measure in a real situation could differ from TMG, because of various reasons (collisions, idle times, driver bugs, environmental conditions etc.). This creates the need for a formula for Theoretical Maximum Goodput (TMG), which reflects the upper bound for the measurable goodput in a given context: $\text{TMG} = \eta_{\text{protocols}} \cdot C$.

η (the "efficiency" of one or more given protocols) represents the impact of the total overhead across those protocol layers. Considering several layers, the total efficiency is the product of the individual η of the various layers. Additionally, it can be measured in two main ways:

- Data-based approach: $\eta = \frac{\text{App Payload (B)}}{\text{App Payload (B) + Overheads (B)}}$
- Time-based approach: $\eta = \frac{\text{App Payload TX time}}{\text{Total TX time}}$

3.2 Useful computations

- We are considering the Ethernet header as part of the overheads even in Wi-Fi communications, since 802.11 networks encapsulate Ethernet frames within 802.11 frames before transmission, to ensure compatibility with wired.
- It's also important to note that the overhead of TCP is not of fixed size, due to possible TCP options being exchanged. In fact, in our experiments, we noted a total size of 32 B (20 B of header + 12 of options).
- $\text{MSS}_{\text{TCP}} = \text{MTU}_{\text{Eth}} - \text{Header}_{\text{IP}} - \text{Header}_{\text{TCP}} = (1500 - 20 - 32)\ \text{B} = 1448\ \text{B}$
- η_{TCP} is the efficiency of the layers above 802.11, up to TCP, in terms of data bytes exchanged over total bytes exchanged to send a MSS segment.
  - $\eta_{\text{TCPoFD}} = \frac{\text{MSS}_{\text{TCP}}}{\text{MSS}_{\text{TCP}}+\text{Header}_{\text{TCP}}+\text{Header}_{\text{IP}}+\text{Overhead}_{\text{Eth}}} = \frac{1448\ \text{B}}{1448+32+20+38\ \text{B}} = 94\%$
  - $\eta_{\text{TCPoHD}} = \frac{\text{MSS}_{\text{TCP}}}{\text{PktLen}_{\text{Data}}+\text{PktLen}_{\text{ACK}}} = \frac{1448\ \text{B}}{(1448+32+20+38)+(32+20+38)\ \text{B}} = 89\%$
- $\eta_{802.11}$ is the efficiency of the layers up to 802.11. When considering the goodput at the application layer over a Wi-Fi channel, we will assume $\eta_{802.11} \approx 80\%$ without performing detailed computations, since precisely determining this efficiency is complicated by the peculiar mechanisms of channel bonding, MU-MIMO and frame aggregation in the modern standards.

4 SCENARIO 1: BOTH HOSTS ON ETHERNET

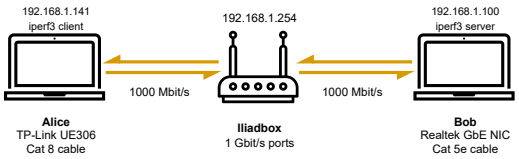

Figure 1: Ethernet scenario

4.1 Scenario description

For our first scenario, we set up the simplest network configuration: two hosts connected via Ethernet cables to the IliadBox router, which acted as a switch. Although this setup is straightforward, it is rarely used in everyday domestic environments. However, it remains widely prevalent in enterprise and corporate settings.

In this scenario, we had: Alice (`iperf3` client), IP: 192.168.1.141; Bob (`iperf3` server) with IP: 192.168.1.100.

4.2 Computing the TMG

In our setup, every device, adapter and port supports a maximum capacity of 1 Gbit/s. Therefore, we can assume the theoretical network capacity as $C = 1$ Gbit/s.

As discussed previously, when using an Ethernet connection in full-duplex mode, we need to account for the efficiency introduced by the various layers, namely Ethernet, IP and the transport layer protocol. We employed TCP, therefore, based on the previous theoretical computations (3.2), the TMG can be estimated as:

$$\text{TMG} = \eta_{\text{TCPoFD}} \cdot C = 0.94 \cdot 1 \text{ Gbit/s} = 940 \text{ Mbit/s}$$

This result reflects the expected goodput under ideal conditions, assuming no packet loss, latency, or other external disturbances.

4.3 Test results evaluation

Table 1: TCP Goodput from `iperf3` test (Mbit/s)

Reverse flag	Avg	Median	Min	Max	Std Dev
False	937.6	938.0	937.0	938.0	0.49
True	897.0	897.0	897.0	897.0	0.0

Table 1 summarizes the `iperf3` test results for standard and reverse transmission modes.

In standard mode (reverse = False), the average goodput is 937.6 Mbit/s, with a median of 938.0 Mbit/s. The minimum (937.0 Mbit/s) and maximum (938.0 Mbit/s) values are close, resulting in a low standard deviation (0.49 Mbit/s), indicating stable performance. In reverse mode (reverse = True), the goodput remains constant at 897.0 Mbit/s, with zero variance, showing slightly lower yet stable performance. This difference may stem from transmission directionality or hardware handling.

Both configurations performed efficiently, with results close to the theoretical maximum goodput (950 Mbit/s).

By analyzing one of the PCAP files captured by the Python script, we can observe the overall behavior of the Ethernet scenario.

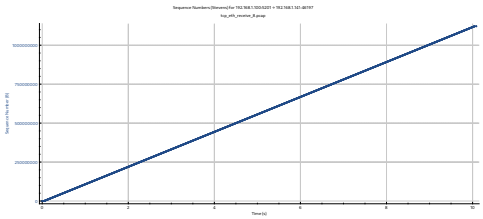

Figure 2: Sequence Numbers (Stevens), reverse flag set to True

Looking at the graph that represents the sequence numbers of packets sent from the server (192.168.1.100) to the client (192.168.1.141) - which are the packets received when the reverse flag is enabled in `iperf3` - we notice that the graph forms a straight line. This indicates that the transmission occurred smoothly, without any retransmissions, and that the throughput remained consistent throughout the test. We also plotted goodput computed by Wireshark.

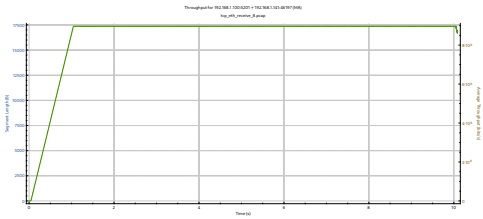

Figure 3: Goodput (MA) on the receiver side, reverse flag set to True

It is important to note that we selected the "Goodput" option when generating the graph instead of "Throughput."

However, we are unsure if Wireshark correctly differentiates between the two because both graphs appear identical.

5 SCENARIO 2: BOTH HOSTS ON WI-FI

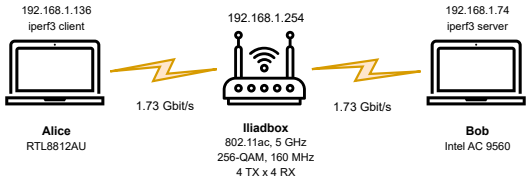

Figure 4: Wi-Fi scenario

5.1 Scenario description

For our second test, we benchmarked our LAN in the scenario of two hosts connected via Wi-Fi to the AP, far more common in the daily use of our home networks.

Using the same laptops as in the Ethernet test, now with Wi-Fi adapters, we configured the AP (same router as before) to enforce 802.11ac via the admin panel, from which we also confirmed it operated with 160 MHz channel bandwidth (eight 20 MHz channels combined) [12] and 256-QAM modulation (combining amplitude and phase shifts to encode 8 bits in a symbol) [13].

In this scenario, we had: Alice (`iperf3` client), IP: 192.168.1.136; Bob (`iperf3` server) with IP: 192.168.1.74.

5.2 Computing the TMG

Based on our information and on the router specifications (in particular, the number of antennas, which is 4 TX x 4 RX for the 5 GHz band), it has been easy to know about the capacity C of the physical link in such a configuration [11]. We have in fact a nominal capacity $C = 1.73$ Gbit/s for both physical links, since both the clients are equipped with 2 TX x 2 RX antennas and support MU-MIMO.

These numbers are to some degree confirmed by the statistics computed by the OS of our devices. The TMG for the communication has been estimated using the equation:

$$\begin{aligned} \text{TMG} &= \eta_{\text{TCPoHD}} \cdot \eta_{802.11} \cdot (0.5 \cdot C) \\ &= 0.89 \cdot 0.80 \cdot (0.5 \cdot 1.73 \text{ GBit/s}) = 616 \text{ Mbit/s} \end{aligned}$$

which takes into account the fact that every frame has to be transmitted twice (hence the factor 0.5), once from the sender to the AP and then from the AP to the recipient. From the equations, TMG is 616 Mbit/s. We expect this to be valid for both directions ($A \to B$ and $B \to A$).

5.3 Test results evaluation

Table 2: TCP Goodput from `iperf3` test (Mbps)

Reverse flag	Avg	Median	Min	Max	Std Dev
False	192.0	193.5	177.0	197.0	5.42
True	147.3	147.0	143.0	150.0	2.00

The test results clash with our expectations; in fact, we have a third of the TMG in the direction $A \to B$ and even a lower goodput for the direction $B \to A$. We dived deeper in the specifications of all the devices, discovering that the RTL8812AU chipset is only capable of exploiting channel bandwidths up to 80 MHz. Our estimation needs now to take into account the disparity in the speeds of the two links, with the $A \leftrightarrow AP$ link being the bottleneck. In particular, the overall capacity is not anymore $0.5 \cdot C$, since the two channels have different transmission speeds:

- $C_{A \leftrightarrow AP} = C/2 = 867$ Mbit/s
- $C_{B \leftrightarrow AP} = C = 1.73$ Gbit/s

Let C' be the new overall capacity. Then:

$$\begin{cases} C' & = \frac{\text{Data}}{\text{TX Time}} = \frac{\text{Data}}{\text{TX Time } (A \leftrightarrow AP) + \text{TX Time } (AP \leftrightarrow B)} \\ & = \frac{\text{Data}}{\frac{\text{Data}}{C_{A \leftrightarrow AP}} + \frac{\text{Data}}{C_{B \leftrightarrow AP}}} \\ & = \frac{1}{1/C_{A \leftrightarrow AP} + 1/C_{B \leftrightarrow AP}} = \frac{1}{3} \cdot C = 578 \text{ Mbit/s} \\ \text{TMG} & = \eta_{\text{TCPoHD}} \cdot \eta_{802.11} \cdot C' \\ & = 0.89 \cdot 0.80 \cdot 578 \text{ Mbit/s} = 411 \text{ Mbit/s} \end{cases}$$

The new equations estimate a TMG of 411 Mbit/s, but discrepancies from the results persist, likely due to signal attenuation, interference from neighbour networks, and unknown device limitations.

Similar to Ethernet, we also have to tackle the loss of goodput (-23%) when communicating in the $B \to A$ direction with respect to the $A \to B$ direction. We are unsure on what caused the discrepancy, which is nonetheless consistent across several tests. While the reciprocity principle [3] suggests symmetrical TX/RX performance, real antennas include additional components that may introduce asymmetries. We hypothesize that 802.11ac optimizes the downlink, as switching to 802.11n @ 2.4 GHz reversed the effect. No other configuration adjustments (client-server swap, device variation, UDP use) made the trend change.

As seen in the Ethernet scenario, we now examine the sequence number and goodput graphs for this case as well. In this PCAP instance, we observe that no retransmissions occurred.

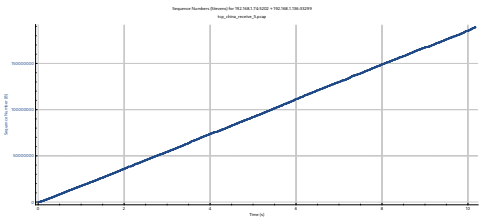

Figure 5: Sequence Numbers (Stevens), reverse flag set to True

However, unlike the Ethernet scenario, the goodput shows more fluctuation over time (it is no longer a flat, horizontal line but displays small oscillations), indicating the presence of a less stable wireless channel.

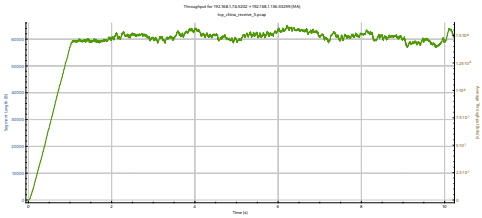

Figure 6: Goodput (MA) on the receiver side, reverse flag set to True

The goodput is also significantly lower than in the Ethernet case. The average goodput calculated from the different `iperf3` tests is 147.3 Mb/s, which closely matches the value observed on Wireshark (above 150 Mb/s, as shown in the graph). The slight discrepancy can be attributed to the seventh test, which performed worse than the others (with retransmissions observed), causing a drop in the overall average goodput.

6 SCENARIO 3: BOTH HOSTS ON WI-FI, ONE SUBMERGED IN SEAWATER

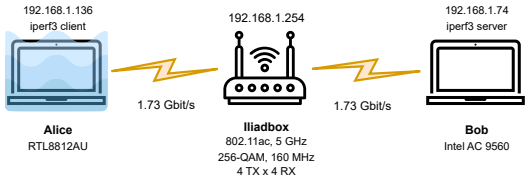

Figure 7: Seawater scenario

6.1 Scenario description

For our third test, we explore how Wi-Fi connections work underwater. This may happen in scenarios such as sensor analysis for water quality and oil spill detection, or submarine communications and mine detection and neutralization. We designed an experiment in which one Wi-Fi-connected host was submerged in seawater

while the other host was not. To replicate this, we enclosed the directional antenna in a waterproof silicone material. Subsequently, we submerged the antenna in 5 cm of water with a salt concentration of 35%.

Our test setup consisted of the same two hosts with identical configurations in the previous scenario (5). The only variable introduced was the addition of water for the host equipped with the directional antenna.

6.2 Consideration on the TMG

The salinity of water significantly impacts electromagnetic (EM) wave propagation, leading to higher attenuation compared to free-space conditions. This is due to changes in the water's permittivity, which is influenced by the dipole moment of water molecules, and the increased conductivity resulting from the ionization of added salt. Pure water has low conductivity because its molecules are not naturally polarized, but the introduction of salt enhances conductivity, further increasing the EM wave attenuation.

Based on these factors, we expect a substantial reduction in throughput in saline water, primarily due to the high attenuation of EM waves. The increased conductivity and dielectric properties of seawater are likely to degrade signal quality and lower the data transfer rate. [2]

In the previous experiment conducted in air, we observed a goodput of 192 Mbit/s under conventional conditions (5). However, when submerged in seawater, we anticipate that signal attenuation, potential multipath effects and wave absorption will significantly reduce goodput.

6.3 Test results evaluation

Table 3: TCP Goodput from `iperf3` test (Mbps)

Reverse flag	Avg	Median	Min	Max	Std Dev
False	3.499	3.81	1.13	5.47	1.49
True	9.914	9.985	5.86	14.6	2.72

During the experiments we noted, observing the router's admin panel, how Alice's network card was hopping from one configuration to another. In particular, it autonomously downgraded its connection from the 5 GHz band to a slower standard in the 2.4 GHz band. In general, the results of this experiment align with our expectations regarding the impact of seawater on Wi-Fi performance. As discussed previously, the presence of saltwater significantly attenuates electromagnetic waves (EM), which in turn degrades the throughput and reliability of wireless networks.

This scenario also reveals differences in signal performance between transmission and reception modes. Among the various reasons for this, we could consider the high unrealiability of the channel as the main cause. In fact, other technologies are often used for underwater communication. Acoustic waves, similar to sonar, are the most common method, as they travel efficiently through water and are used in submarines and underwater research (A) [2].

Moving on to examine the sequence number graph for this scenario, we can immediately notice that the slope of the curve is less steep compared to previous scenarios. This suggests a lower overall throughput.

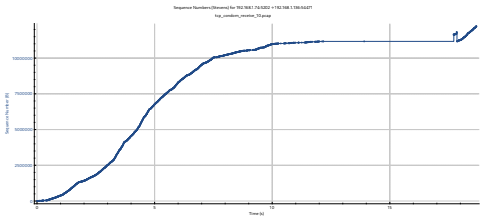

Figure 8: Sequence Numbers (Stevens), reverse flag set to True

Additionally, the transmission lasted longer than the 10 seconds we set in the test parameters. This is evident from the graph, where towards the end of the transmission, the slope flattens (forming a horizontal line), indicating that no data was being received at that point. This is due to the need for retransmissions, as shown by the drop in the sequence number at one point. These retransmissions and the presence of duplicate ACKs are confirmed by using specific Wireshark filters and observing the TCP errors graph (B).

The second graph, which shows the outstanding bytes (bytes that have been sent but have not yet been acknowledged), further confirms that retransmissions occurred later in the transmission. This behavior can be better understood when examining the receiving window graph. The recWin shows how much data the receiver can accept at a given time. The recWin graph fluctuates continuously, suggesting that the system was adjusting dynamically due to network congestion or other factors.

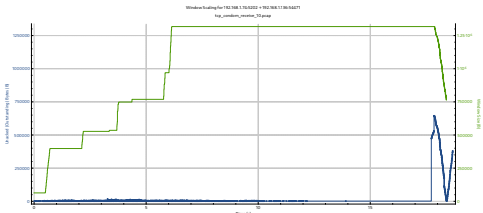

Figure 9: OutBytes and recWin size on the client, reverse flag set to True

Unlike other scenarios, where the recWin remains stable, the continuous change in this case indicates that the submerged receiver dynamically adjusted its window size, likely due to fluctuating network conditions during the test. This adjustment was a preventive measure to avoid overloading the receiver. In fact, if the receiver experiences high traffic or delays, it may reduce the window size to prevent overloading, causing fluctuations in the recWin even without the need for retransmission.

In conclusion, the experiment validated our initial predictions regarding the catastrophic impact of seawater on EM wave propagation.

7 CONCLUSION

Despite our computations being somewhat approximative, we can still extract valuable insights. Wired networks offer stable transmission, allowing accurate goodput estimations due to controlled variables. In contrast, wireless communication introduces complexity, making precise theoretical predictions challenging, because of the wide range of technologies of 2025. Finally, empirical evaluation remains essential, particularly in highly variable environments like saline water.

REFERENCES

[1] Ghida Jubran Alqahtani and Fatma Bouabdallah. 2023. Routing protocols based on node selection for freely floating underwater wireless sensor networks: a survey. *EURASIP Journal on Wireless Communications and Networking* 2023, 1, 117. https://doi.org/10.1186/s13638-023-02324-6

[2] Giovanni Arrabito. 2018. *Underwater communication system design using electromagnetic waves.* Master's Thesis. Politecnico di Torino. https://webthesis.biblio.polito.it/7527/1/tesi.pdf

[3] Constantine A. Balanis. 2005. *Antenna Theory: Analysis and Design* (3rd ed.). Wiley.

[4] Salvador Climent, Antonio Sanchez, Juan Vicente Capella, Nirvana Meratnia, and Juan Jose Serrano. 2014. Underwater Acoustic Wireless Sensor Networks: Advances and Future Trends in Physical, MAC and Routing Layers. *Sensors* 14, 1, 795–833. https://doi.org/10.3390/s140100795

[5] Iliad SA. Iliadbox Wi-Fi 6 Technical Specifications. https://assistenza.iliad.it/privati/fibra/supporto/iliadbox-guida-e-caratteristiche-tecniche/. [Online; accessed 30-March-2025].

[6] Intel Corp. Intel Wireless-AC 9560 Technical Specifications. https://www.intel.com/content/www/us/en/products/sku/99446/intel-wirelessac-9560/specifications.html. [Online; accessed 30-March-2025].

[7] Ning Li, José-Fernán Martínez, Juan Manuel Meneses Chaus, and Martina Eckert. 2016. A Survey on Underwater Acoustic Sensor Network Routing Protocols. *Sensors* 16, 3. https://doi.org/10.3390/s16030414

[8] Qian Lu, Feng Liu, Ying Zhang, and Shengming Jiang. 2017. Routing Protocols for Underwater Acoustic Sensor Networks: A Survey from an Application Perspective. In *Underwater Acoustics*, Andrzej Zak (Ed.). IntechOpen, Rijeka, Chapter 2. https://doi.org/10.5772/intechopen.68900

[9] Realtek Semiconductors Corp. Realtek RTL8812AU Technical Specifications. https://www.elinfor.com/pdf/RealtekMicroelectronics/RTL8812AU-CG-RealtekMicroelectronics.pdf. [Online; accessed 30-March-2025].

[10] TP-Link. TP-Link UE306 Technical Specifications. https://www.tp-link.com/us/home-networking/usb-converter/ue306/. [Online; accessed 30-March-2025].

[11] Wikipedia contributors. 2025. IEEE 802.11ac-2013 — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=IEEE_802.11ac-2013&oldid=1282616403. [Online; accessed 30-March-2025].

[12] Wikipedia contributors. 2025. Link aggregation / Channel bonding — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Link_aggregation. [Online; accessed 30-March-2025].

[13] Wikipedia contributors. 2025. Quadratic Amplitude Modulation — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Quadrature_amplitude_modulation. [Online; accessed 30-March-2025].

A AN ALTERNATIVE WAY FOR UNDERWATER TRANSMISSIONS

Underwater Wireless Sensor Networks (UWSNs) are crucial for applications such as environmental monitoring, surveillance, and marine resource management. Unlike terrestrial networks, UWSNs rely on acoustic communication due to the high attenuation of electromagnetic waves in water. However, acoustic communications pose significant challenges, including limited bandwidth, high propagation delays, and high error rates [4].

The design of efficient routing protocols for UWSNs is complex due to node mobility caused by ocean currents, the variability of the acoustic channel, and the limited energy resources of sensor nodes. Routing protocols must ensure reliable and efficient data transmission while adapting to the dynamic underwater environment [7].

Several approaches have been proposed to address these challenges. For example, cross-layer design protocols integrate information from different layers of the OSI model to optimize overall network performance. Additionally, intelligent algorithms, such as reinforcement learning, have been explored to enhance the adaptability of routing protocols to the dynamic conditions of UWSNs [8].

Moreover, a recent study on routing protocols for UWSNs classifies these protocols based on node mobility management, distinguishing between those that rely on continuous updates of localization information and those that predict future node positions. This classification provides a better understanding of the strategies adopted to address the challenges posed by mobility in the underwater environment. [1]

In summary, the development of routing protocols for UWSNs requires careful consideration of the unique characteristics of underwater acoustic communications and environmental challenges to ensure effective and sustainable communication in marine applications.

B TCP ERRORS GRAPH IN THE SEAWATER SCENARIO

The same observations we made about retransmissions and duplicate ACKs can be made when looking at the packet error + throughput graph, which also shows a fluctuating trend, indicating the presence of an unreliable channel.

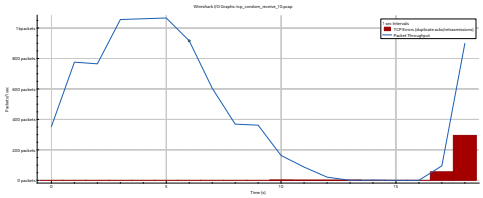

Figure 10: Packets/s (MA) on the receiver side + TCP errors, reverse flag set to True

C PYTHON CODE

To run the various `iperf3` tests and collect aggregated results, we crafted a Python script with the following capabilities:

- The script can autonomously run `iperf3` tests on a Linux machine.
- Every time a test, which is performed N times, finishes, the script computes and prints out a table containing minimum, maximum, mean, median and standard deviation over the N iterations.
- The various tests to run can be listed in a CSV file. Each CSV line specifies a name for the test, the number N of iterations and additional parameters for the `iperf3` command (the protocol to use, the server's IP address, the desired bitrate etc.).
- Each time a single iteration of a test is executed, a PCAP file containing every packet sent or received during that iteration is saved in a folder named after the test. For example, the test "realtek" with $N = 5$ will generate five PCAPs: `realtek_1.pcap` to `realtek_5.pcap`.

```
import subprocess
import time
import numpy as np
import csv

def run_iperf(name, server_ip, bind_ip, reverse, repetitions, pcap_fileNoExt, server_port, duration, udp, bitrate,
↪  app_buff_length):
    results = []

    try:
        for i in range(repetitions):
            # Start tcpdump to capture traffic
            tcpdump_process = subprocess.Popen(
                ["tcpdump", "-i", "any", "host", server_ip, "-w", pcap_fileNoExt+f"_{i+1}.pcap"],
                stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
            )

            time.sleep(2)  # Allow tcpdump to start

            try:
                command = ["iperf3", "-c", server_ip, "-B", bind_ip, "-p", str(server_port), "-t", str(duration)]
                if reverse:
                    command.append("-R")
                if udp:
                    command.append("-u")
                if bitrate:
                    command.extend(["-b", str(bitrate)])
                if app_buff_length:
                    command.extend(["-l", str(app_buff_length)])

                result = subprocess.run(
                    command,
                    capture_output=True, text=True, check=True
                )

                for line in result.stdout.split("\n"):
                    if "receiver" in line:
                        parts = list(filter(None, line.split()))
                        if len(parts) > 6:
                            results.append(float(parts[6]))
            except subprocess.CalledProcessError as e:
                print(f"Error running iperf3: {e}")

            time.sleep(1)
            # Stop tcpdump
            tcpdump_process.terminate()
            tcpdump_process.wait()
    except Exception as e:
```

```
        print(e)

    return name, results

def compute_stats(name, data):
    if not data:
        print(f"No data collected for test {name}.")
        return name, None, None, None, None, None

    avg = np.mean(data)
    median = np.median(data)
    min_val = np.min(data)
    max_val = np.max(data)
    std_dev = np.std(data)

    print(f"Test: {name}")
    print(f"avg= {avg:.3f}")
    print(f"median= {median:.3f}")
    print(f"min= {min_val:.3f}")
    print(f"max= {max_val:.3f}")
    print(f"std= {std_dev:.3f}")

    return name, avg, median, min_val, max_val, std_dev

def read_csv_parameters(csv_file):
    tests = []
    with open(csv_file, mode='r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip header
        for row in reader:
            name = row[0]
            server_ip = row[1]
            bind_ip = row[2]
            reverse = row[3].strip().lower() == 'true'
            repetitions = int(row[4])
            pcap_fileNoExt = row[5]
            server_port = int(row[6]) if len(row) > 6 else 5201  # Default port 5201
            duration = int(row[7]) if len(row) > 7 else 10  # Default 10s
            udp = row[8].strip().lower() == 'true' if len(row) > 8 else False
            bitrate = row[9] if len(row) > 9 and row[9] else None
            app_buff_length = row[10] if len(row) > 10 and row[10] else None

            tests.append((name, server_ip, bind_ip, reverse, repetitions, pcap_fileNoExt, server_port, duration, udp, bitrate,
            ↪  app_buff_length))
    return tests

def save_results_to_csv(results):
    for result in results:
        test_name = result[0]
        output_csv = f"{test_name}_results.csv"
        with open(output_csv, mode='w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(["Test Name", "Avg Throughput", "Median", "Min", "Max", "Std Dev"])
            writer.writerow(result)
        print(f"Results saved to {output_csv}")

if __name__ == "__main__":
    input_csv = "test_parameters.csv"  # Input CSV file

    test_cases = read_csv_parameters(input_csv)
    all_results = []
```

```
    for test in test_cases:
        print(f"Running test with parameters: {test}")
        name, throughput_data = run_iperf(*test)
        test_results = compute_stats(name, throughput_data)
        all_results.append(test_results)

    save_results_to_csv(all_results)
```

An example of the CSV file is as follows:

name	server_ip	bind_ip	reverse
TCP_send	192.168.1.74	192.168.1.136	False
TCP_receive	192.168.1.74	192.168.1.136	True

repetitions	pcap_file_noExt	server_port	duration
10	send	5202	10
10	receive	5202	10

udp	bitrate	app_buff_length