Computer Networks II

Report
Project 2 : SpeedTest with Unix Sockets

Due Date: May 27th, 2025

Submitted by:
Gravalos Georgios - Angelos, 2021030001
Kerimi Rafaela - Aikaterina, 2021030007
Stamou Spyridon, 2021030090
Apostolopoulos - Papidas Dimitris, 2021030176

Technical University of Crete
Department of Electrical and Computer Engineering

Contents

Introduction

The goal of this project is the implementation of a SpeedTest application using TCP Unix Sockets in C. While typical SpeedTest applications measure throughput and latency at both Downlink and Uplink directions, our SpeedTest App will measure only Downlink TCP throughput, using a client/server architecture, running over Unix Sockets.
The system consists of two C files, **server.c and client.c**.

- **server.c**: Accepts TCP connections and calculates per-interval and average throughput over a fixed duration.
- **client.c**: Connects to the server and sends dummy data continuously at maximum speed. Also calculates per-interval and average throughput from its own side.

The system is designed to simulate a real-world speed test between a client and an access point/server, and to allow precise throughput logging every 2 seconds, as required for network performance analysis. To verify our app's throughput estimation, we compared its performance with **iperf** tool.
Using the Wi-Fi Doctor implemented in Project 1, we also analyzed the performance of the wireless link, taking into consideration metrics, such as Throughput as estimated by the Wi-Fi Doctor, Data Rate, Frame Loss, RSSI and Rate Gap.

Finally, we revised Wi-Fi Doctor throughput estimation by weighing in one more factor: **channel utilization**. With this addition, our estimation approaches real-life throughput values in a more successful way.

Server and Client Implementation with Unix Sockets

The two core files of the SpeedTest using Unix Sockets are `client.c` and `server.c`. As you can tell, the entire SpeedTest is based on a client/server architecture, and is designed to measure downlink throughput at the server.
The server always runs at a configured port, actively listening for other clients.
The client connects to the server's port/IP, and generates buffer traffic for 30sec.
For the duration of the SpeedTest, both the client and server print out the throughput (Mbps) in 2-second windows, and the average throughput (Mbps) after the 30 seconds. Throughput is **data sent rate** for client, **data received rate** for the server.

We chose our **port** to be **14444.** We chose a **buffer size of 64KB** (65536), given that this is the max size of a TCP segment's payload. Likewise, we defined the interval to be 2 seconds, whereas the duration was set to 30 seconds. The above are true for both server and client and have been defined as constants on top of the file.

`client.c`

This file is the sender part of the SpeedTest.
We begin with using `getaddrinfo()` to resolve the server's IP to a usable `sockaddr` struct. The address is strictly IPv4 (`AF_INET`) and the connection type is strictly TCP `SOCK_STREAM`.
Using `socket()` and `connect()`, the client attempts to create a TCP socket, using the address info, and connect to the server using the resolved address and port.
Once the connection succeds, we initialize the desired-size buffer with the character 'A', which is the data that we will be using for the SpeedTest.
We start the time counter, and enter a `while(1)` loop, which:
- Calculates how much time has passed
- Starting at time 0sec, it sends data for the first time. Then, at the next allowed interval (2sec), it calculates the sent throughput, waits for the next allowed interval (4sec) to send again, and so on. The send throughput is calculated and printed in Mbps using the formula

$$\text{Throughput (Mbps)} = \frac{\text{interval\_bytes\_sent} \times 8}{10^6 \times \text{interval\_seconds}}$$

Each time the client uses `send()`, the variable `total_bytes_sent` is incremented by `bytes_sent`.
Once the 30 seconds have passed, the program exits the `while(1)` loop (using an `if` that checks if $elapsedTime > 30sec$) and calculates the average throughput, using the formula

$$\text{AverageThroughput (Mbps)} = \frac{\text{total\_bytes\_sent} \times 8}{10^6 \times \text{interval\_seconds}}$$

and closes the socket.

It's worth noting that on the client side, if there is delay or congestion, for any reason, and the AP throttles the client's sending rate, the throughput will appear to be 0Mbps. That will not necessarily be true for the server, which will still be receiving data at a slower rate, whereas the client will be forced by the AP to stop overwhelming the server.

`server.c`

The `server.c` file implements the receiving end of the SpeedTest. It

In the **server_setup()** function, we create the TCP socket with the desired configurations. It calls the `socket()` system call, which creates an IPv4 TCP socket (`AF_INET`, `SOCK_STREAM`). The use of `setsockopt()` avoids the error message "address already in use" and allows address reuse. We bind the socket to the IP and port with `bind()`, and mark it ready to accept connections using `listen()`.

The utility function **get_real_time()** uses `gettimeofday()`, returns the current time with precision in seconds, as a double. This is done to accurately measure throughput intervals.

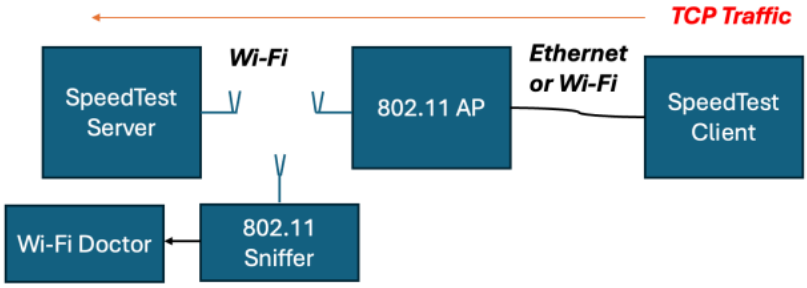
In **handle_client()**, we handle the 30-second data reception from clients. We receive data in 64KB chunks using `recv()`, track total bytes received during the session, and interval bytes for the 2-sec windows with the help of the utility function mentioned above.
The interval throughput and average throughput are both calculated using the same formulas we presented in `client.c`.

In the **main()**, we call `server_setup()` and inside a `while(1)` loop, we wait for incoming client connections using `accept()`. upon successful connection, we call `handle_client()` to perform the throughput measurements. After the test, we close the connection with the client and keep the server waiting for new connections.

The socket is configured with `TCP_BACKLOG` value of 1 in order to handle one client at a time.

SpeedTest App Evaluation

We tested our SpeedTest application using the topology below.



We connected the server to our home network through our home AP, while the client was connected via Ethernet.
A third device, set in Monitor mode using Wireless Diagnostics, operating as a 802.11 frame sniffer in the same channel as the AP, was used to capture the communication in **5 different scenarios**:
1) SpeedTest server **close to AP** (high RSSI), **5GHz**.
2) SpeedTest server **far from AP** (low RSSI), **5GHz**.
3) SpeedTest server **close to AP** (high RSSI), **2.4GHz**.
4) SpeedTest server **far from AP** (low RSSI), **2.4GHz**.
5) Mobility test, where the server is moving at pedestrian speed **far from AP**, **5GHz**.
To achieve high RSSI, all we needed to do was place the server and sniffer right next to the AP. To achieve low RSSI, it took serious thinking and craftsmanship to come up with the idea to cover the AP with tinfoil, then throw two pairs of jeans on top, and hide the server and sniffer in the closet in the next room and shut the door.

For all the tests:
- **Server MAC** → `dc:45:46:54:0d:e9`
- **Client MAC** → `3c:ab:72:13:26:55`
- **AP MAC (5GHz)** → `04:71:53:5e:f2:bb`
- **AP MAC (2.4GHz)** → `04:71:53:5e:f2:b6`

Therefore, to **filter the communication frames between AP and server, originating from the client**, we apply the following filters:
- **5GHz** `.pcap` **files** : `wlan.sa == 3c:ab:72:13:26:55 and wlan.ta == 04:71:53:5e:f2:bb and wlan.da == dc:45:46:54:0d:e9 and radiotap.dbm_antsignal != 0`
- **2.4GHz** `.pcap` **files** : `wlan.sa == 3c:ab:72:13:26:55 and wlan.ta == 04:71:53:5e:f2:b6 and wlan.da == dc:45:46:54:0d:e9 and radiotap.dbm_antsignal != 0`

The iperf tool

We use the iperf, an open-source network perfomance measurement tool, to compare the perfomance of our SpeedTest.
After each SpeedTest scenario, we conducted the iperf tests, matching the exact conditions (for good/bad RSSI), to get the appropriate results.
On one laptop, we run the `iperf3 -s` command to start the server, which listens for upcoming connections.

On the other laptop, we run the `iperf3 -c <server_ip> -t 30 -i 2` , which configure the iperf to run for a duration of 30 seconds, with 2 intervals to measure throughput.

High RSSI, 5GHz

SpeedTest Analysis

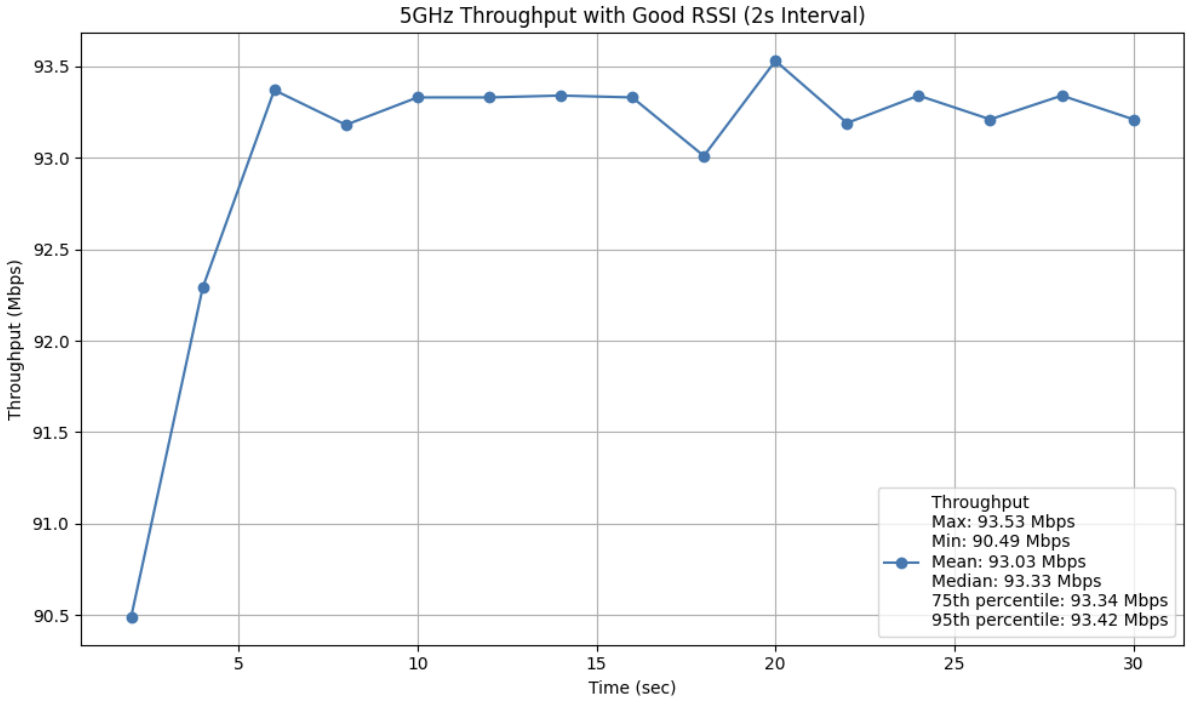
The results are:

```
Speed test started for this client...
[2.0 sec] Throughput: 90.49 Mbps
[4.0 sec] Throughput: 92.29 Mbps
[6.0 sec] Throughput: 93.37 Mbps
[8.0 sec] Throughput: 93.18 Mbps
[10.0 sec] Throughput: 93.33 Mbps
[12.0 sec] Throughput: 93.33 Mbps
[14.0 sec] Throughput: 93.34 Mbps
[16.0 sec] Throughput: 93.33 Mbps
[18.0 sec] Throughput: 93.01 Mbps
[20.0 sec] Throughput: 93.53 Mbps
[22.0 sec] Throughput: 93.19 Mbps
[24.0 sec] Throughput: 93.34 Mbps
[26.0 sec] Throughput: 93.21 Mbps
[28.0 sec] Throughput: 93.34 Mbps
[30.0 sec] Throughput: 93.21 Mbps
Total throughput over 30 sec: 93.03 Mbps
```

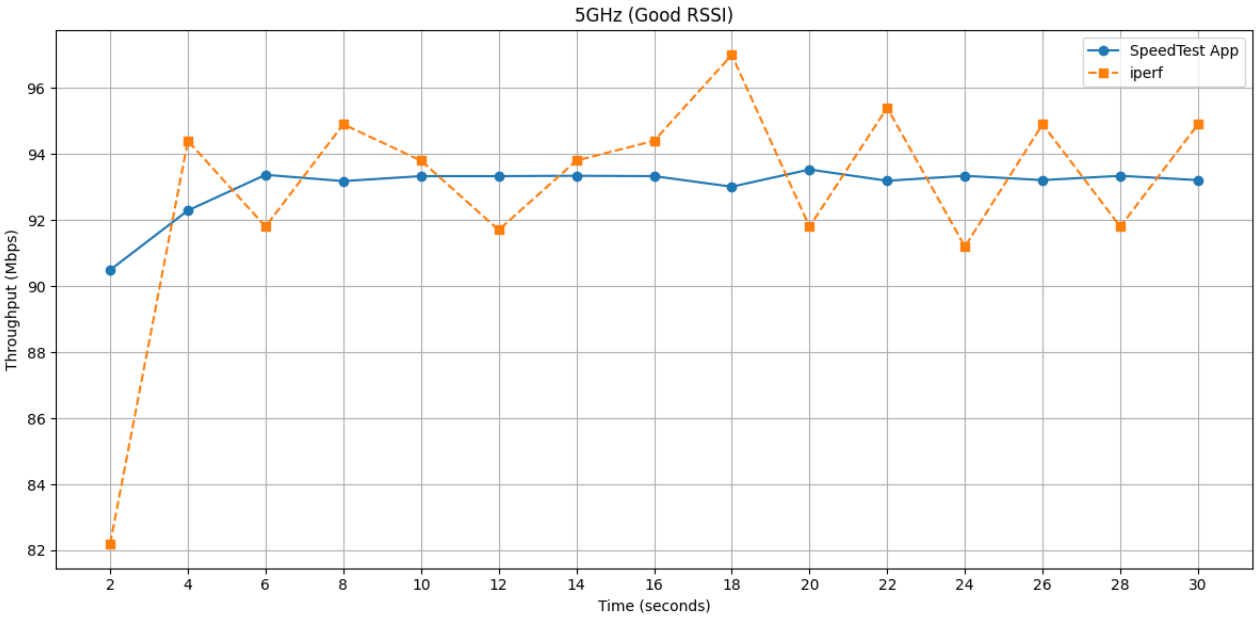
```
Connecting to host 192.168.1.5, port 5201
[  5] local 192.168.1.8 port 42390 connected to 192.168.1.5 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-2.00   sec  19.6 MBytes  82.2 Mbits/sec   74    175 KBytes
[  5]   2.00-4.00   sec  22.5 MBytes  94.4 Mbits/sec    0    215 KBytes
[  5]   4.00-6.00   sec  21.9 MBytes  91.8 Mbits/sec    0    215 KBytes
[  5]   6.00-8.00   sec  22.6 MBytes  94.9 Mbits/sec    0    272 KBytes
[  5]   8.00-10.00  sec  22.4 MBytes  93.8 Mbits/sec    0    272 KBytes
[  5]  10.00-12.00  sec  21.9 MBytes  91.7 Mbits/sec    0    272 KBytes
[  5]  12.00-14.00  sec  22.4 MBytes  93.8 Mbits/sec    0    272 KBytes
[  5]  14.00-16.00  sec  22.5 MBytes  94.4 Mbits/sec    0    272 KBytes
[  5]  16.00-18.00  sec  23.1 MBytes  97.0 Mbits/sec    0    413 KBytes
[  5]  18.00-20.00  sec  21.9 MBytes  91.8 Mbits/sec    0    413 KBytes
[  5]  20.00-22.00  sec  22.8 MBytes  95.4 Mbits/sec    0    413 KBytes
[  5]  22.00-24.00  sec  21.8 MBytes  91.2 Mbits/sec    0    413 KBytes
[  5]  24.00-26.00  sec  22.6 MBytes  94.9 Mbits/sec    0    413 KBytes
[  5]  26.00-28.00  sec  21.9 MBytes  91.8 Mbits/sec    0    413 KBytes
[  5]  28.00-30.00  sec  22.6 MBytes  94.9 Mbits/sec    0    413 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-30.00  sec   332 MBytes  92.9 Mbits/sec   74             sender
[  5]   0.00-30.01  sec   330 MBytes  92.3 Mbits/sec                  receiver

iperf Done.
```

Results of our SpeedTest (left), results of iperf (right)

Using the `numpy` library, we generated the timeseries for the throughput, along with the throughput distribution (max, min, mean, median, 75th percentile, 95th percentile).



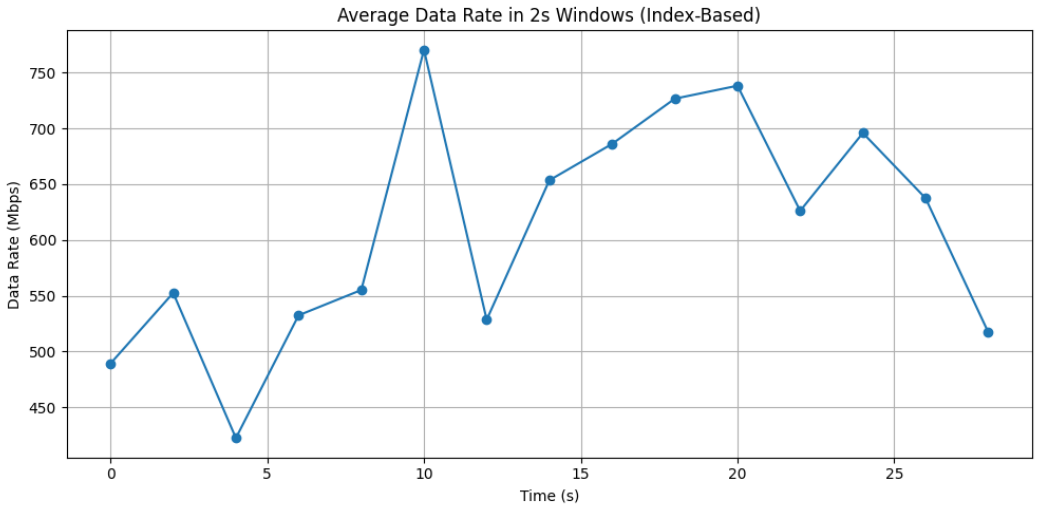
A timeseries comparison between our SpeedTest implementation and iperf is shown below :



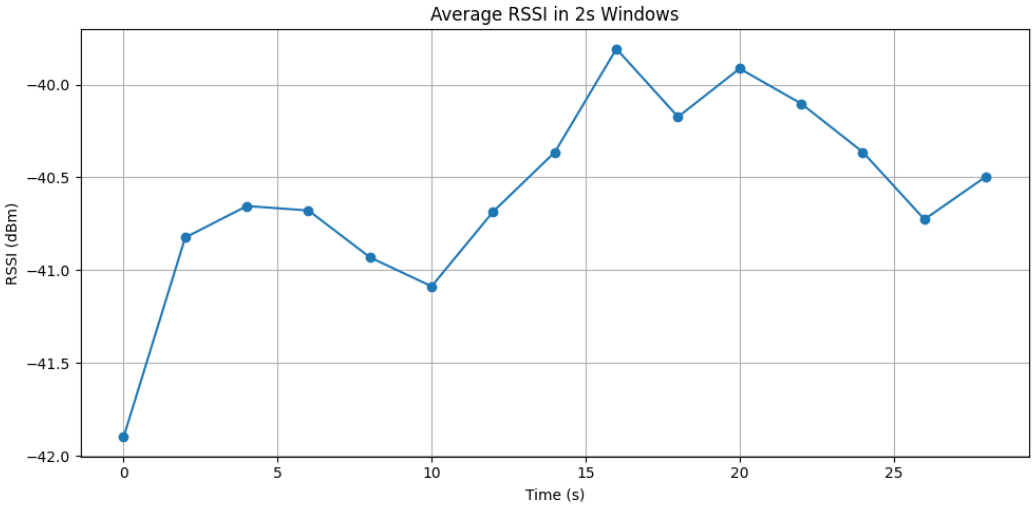
We observe that while both time series have approximately the same mean throughput, iperf exhibits greater variance compared to SpeedTest. This indicates that our SpeedTest implementation provides more stable throughput measurements. The fluctuations observed in iperf, around ±3%, fall within a typical and acceptable margin of error.

WiFi Doctor Analysis

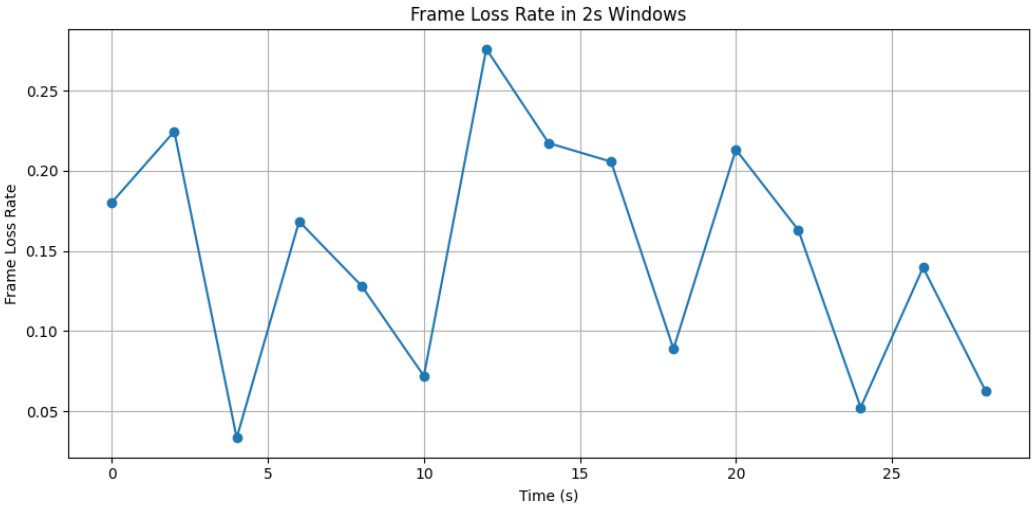
Since our original WiFi Doctor was significantly more complex than needed for this project, and was built entirely on MCS needs for project 1's pcap files (only 802.11n frames), we made a simplified version that is compatible with all 802.11 protocols. It only extracts RSSI, data rate, retry rates, and we used these to calculate frame loss and Rate Gap.



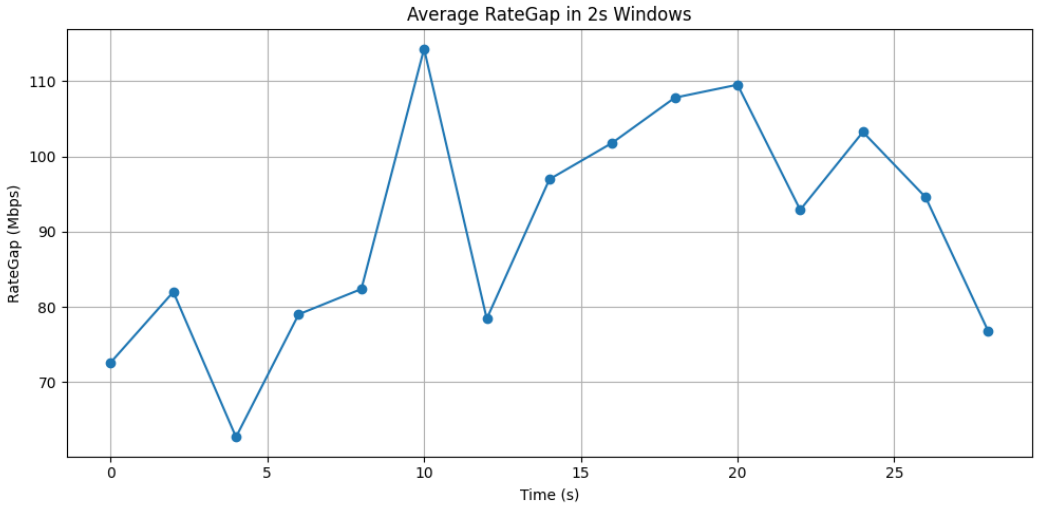
We can observe that results are greatly exaggerated compared to the actual SpeedTest. Firstly, the data rate that we see on Wireshark stems from the 802.11 metrics, such as MCS, channel width, GI, spatial streams. These do not represent the actual data rate, only a theoretical maximum.
Even though our home line is capable of sending up to 300Mbps (MCS does not take that hardware detail into account), the ethernet cable connecting the client to the AP negotiates only 100Mbps due to the CAT5 limitations of the cable.



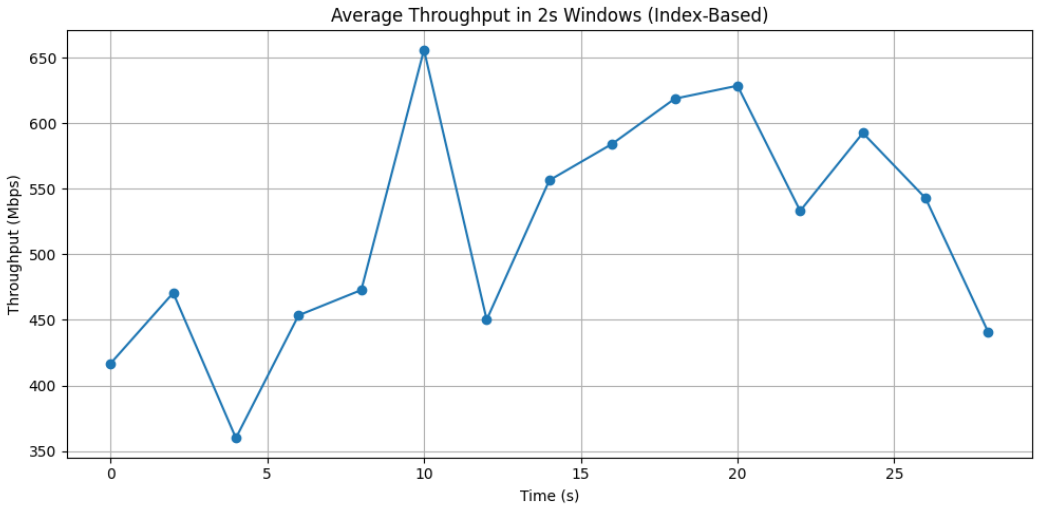
RSSI values indicate that we have succeeded in our objective. The signal strength is great, which is what lets the MCS index to be such that allows for theoretical data rates that are not even reachable in a realistic SpeedTest scenario.



Given our exceptional RSSI in this scenario, the sender (AP) chooses higher MCS indexes, using modulation schemes such as 64-QAM and 256QAM, wider bandwidths (80MHz) and possibly short GI. These configurations may be faster, but are more susceptible to higher BER. This leads to an increased number of losses, which as we will see affects both throughput and Rate Gap. However, this does not deal significant enough damage to throughput and rate gap to consider it harmful.



Given that in our scenarios, we consider **Rate Gap = Data Rate − Throughput**, we can observe that the plot of Rate Gap is identical to the one of Data Rate, but in a smaller number scale, of course.



Again, given that in our scenarios, we consider **Rate Gap = Data Rate − Throughput**, the plot of the throughput is again identical to the other two plots, but in a different scale.
Since **Throughput = (1-FrameLoss)·DataRate**, this is only a theoretical max throughput, based on the frameloss and the theoretical max data rate, and not a realistic depiction of the SpeedTest.

Low RSSI, 5GHz

SpeedTest Analysis

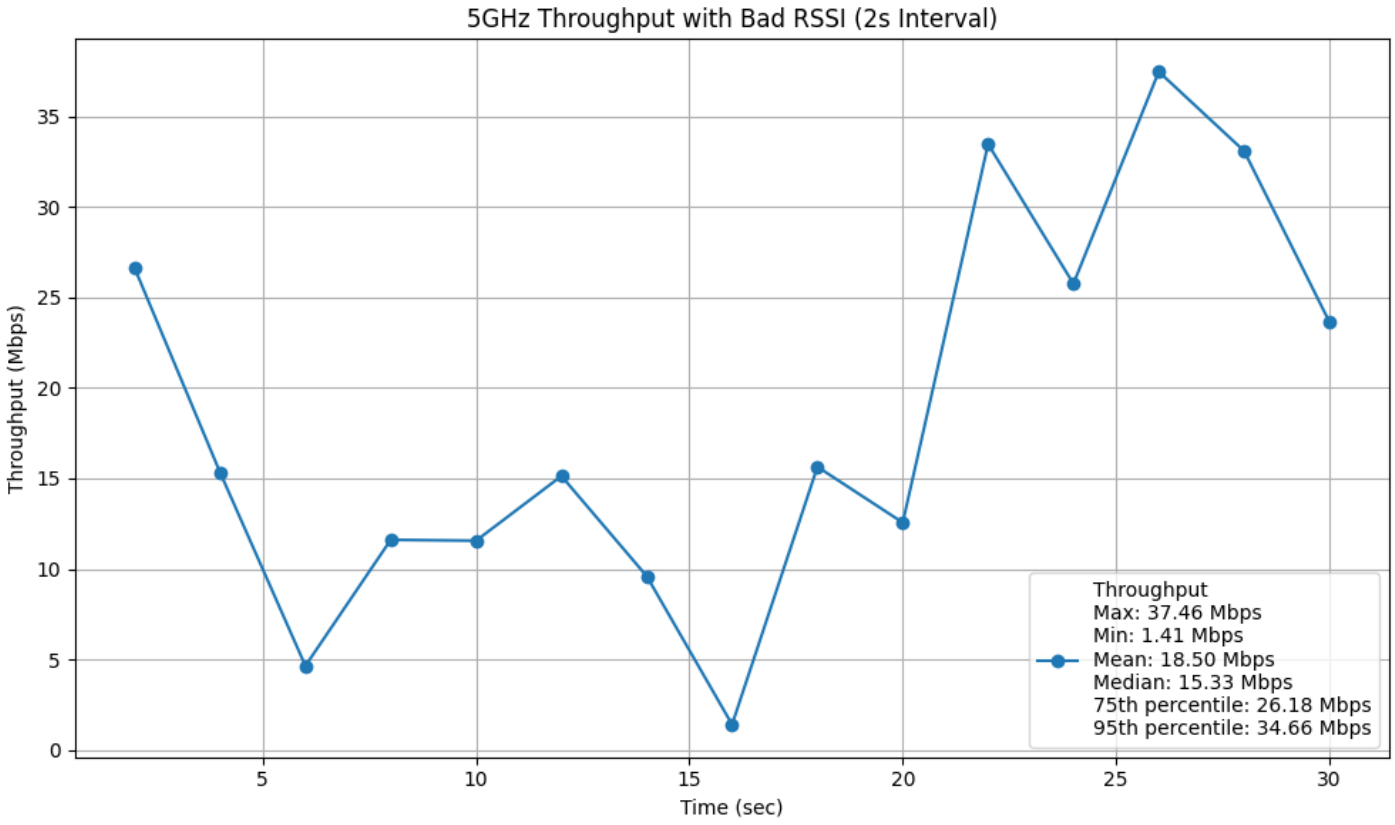
The results are:

```
Speed test started for this client...
[2.0 sec] Throughput: 26.60 Mbps
[4.0 sec] Throughput: 15.33 Mbps
[6.0 sec] Throughput: 4.65 Mbps
[8.0 sec] Throughput: 11.61 Mbps
[10.0 sec] Throughput: 11.56 Mbps
[12.0 sec] Throughput: 15.13 Mbps
[14.0 sec] Throughput: 9.60 Mbps
[16.0 sec] Throughput: 1.41 Mbps
[18.0 sec] Throughput: 15.64 Mbps
[20.0 sec] Throughput: 12.56 Mbps
[22.0 sec] Throughput: 33.46 Mbps
[24.0 sec] Throughput: 25.75 Mbps
[26.0 sec] Throughput: 37.46 Mbps
[28.0 sec] Throughput: 33.08 Mbps
[30.0 sec] Throughput: 23.64 Mbps
Total throughput over 30 sec: 18.50 Mbps
```

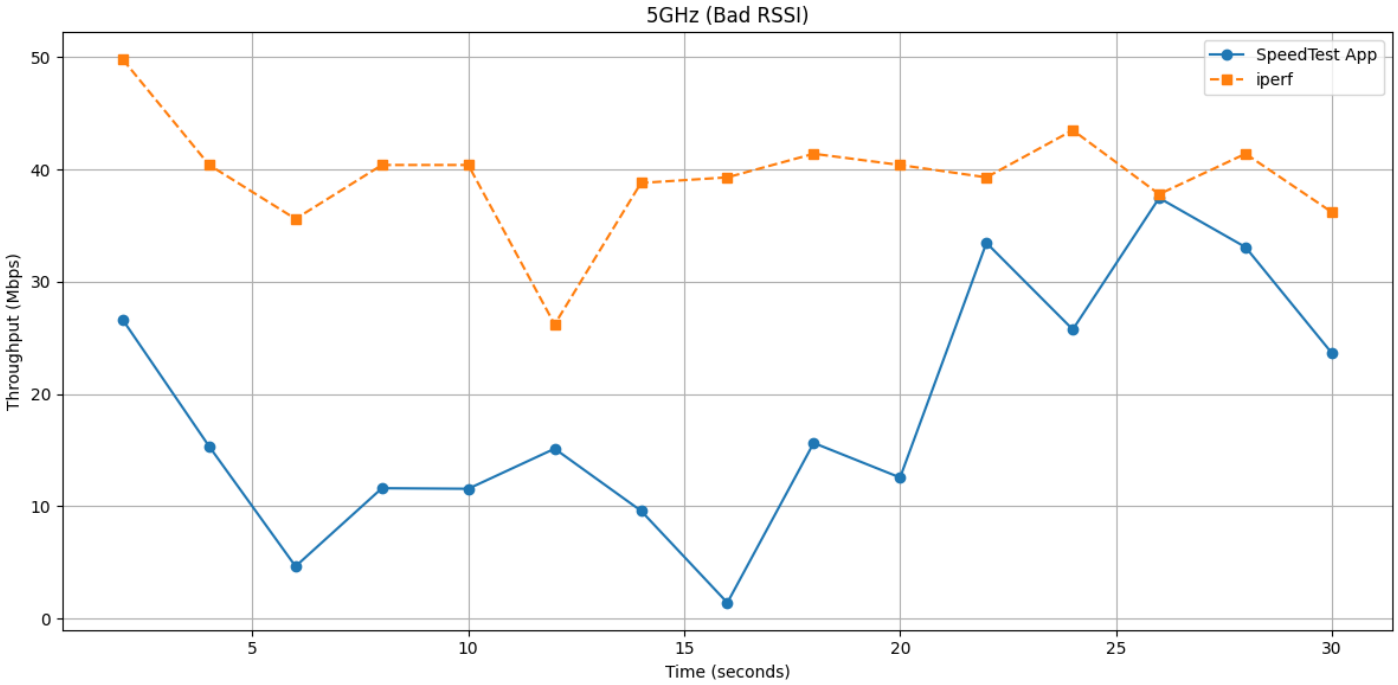
```
Connecting to host 192.168.1.5, port 5201
[  5] local 192.168.1.8 port 39882 connected to 192.168.1.5 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-2.00   sec  11.9 MBytes  49.8 Mbits/sec    0    592 KBytes
[  5]   2.00-4.00   sec  9.62 MBytes  40.4 Mbits/sec    0   1.07 MBytes
[  5]   4.00-6.00   sec  8.50 MBytes  35.6 Mbits/sec    0   1.47 MBytes
[  5]   6.00-8.00   sec  9.62 MBytes  40.4 Mbits/sec    0   1.92 MBytes
[  5]   8.00-10.00  sec  9.62 MBytes  40.4 Mbits/sec    0   2.12 MBytes
[  5]  10.00-12.00  sec  6.25 MBytes  26.2 Mbits/sec  238   1.48 MBytes
[  5]  12.00-14.00  sec  9.25 MBytes  38.8 Mbits/sec    0   1.77 MBytes
[  5]  14.00-16.00  sec  9.38 MBytes  39.3 Mbits/sec    0   1.95 MBytes
[  5]  16.00-18.00  sec  9.88 MBytes  41.4 Mbits/sec    0   2.03 MBytes
[  5]  18.00-20.00  sec  9.62 MBytes  40.4 Mbits/sec    0   2.03 MBytes
[  5]  20.00-22.00  sec  9.38 MBytes  39.3 Mbits/sec    0   2.03 MBytes
[  5]  22.00-24.00  sec  10.4 MBytes  43.5 Mbits/sec    0   2.03 MBytes
[  5]  24.00-26.00  sec  9.00 MBytes  37.8 Mbits/sec    0   2.03 MBytes
[  5]  26.00-28.00  sec  9.88 MBytes  41.4 Mbits/sec    0   2.03 MBytes
[  5]  28.00-30.00  sec  8.62 MBytes  36.2 Mbits/sec    0   2.03 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-30.00  sec   141 MBytes  39.4 Mbits/sec  238             sender
[  5]   0.00-30.39  sec   140 MBytes  38.6 Mbits/sec                  receiver

iperf Done.
```

Results of our SpeedTest (left), results of iperf (right)

A timeseries for the throughput, along with its statistics (max, min, mean, median, 75th percentile, 95th percentile).

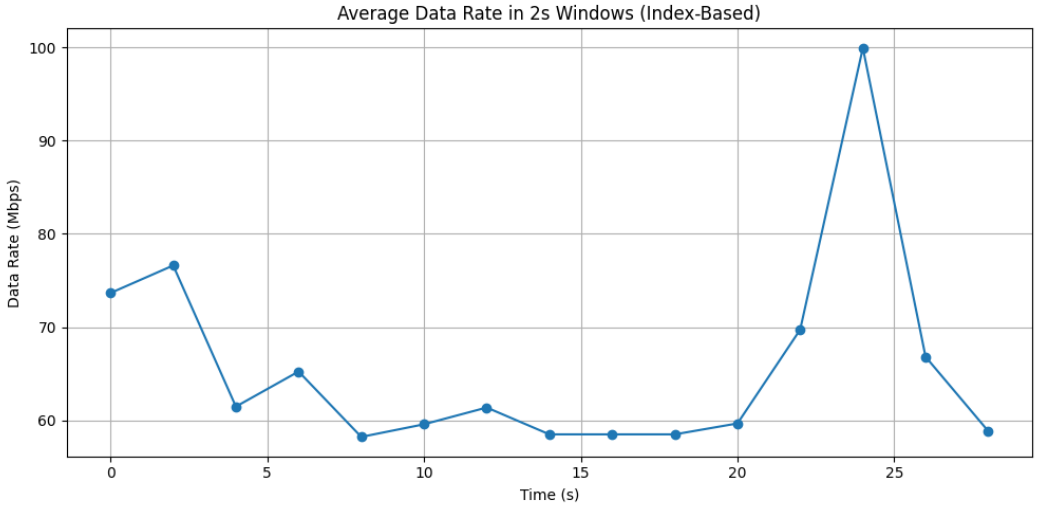


The timeseries of the SpeedTest - iperf comparison is :

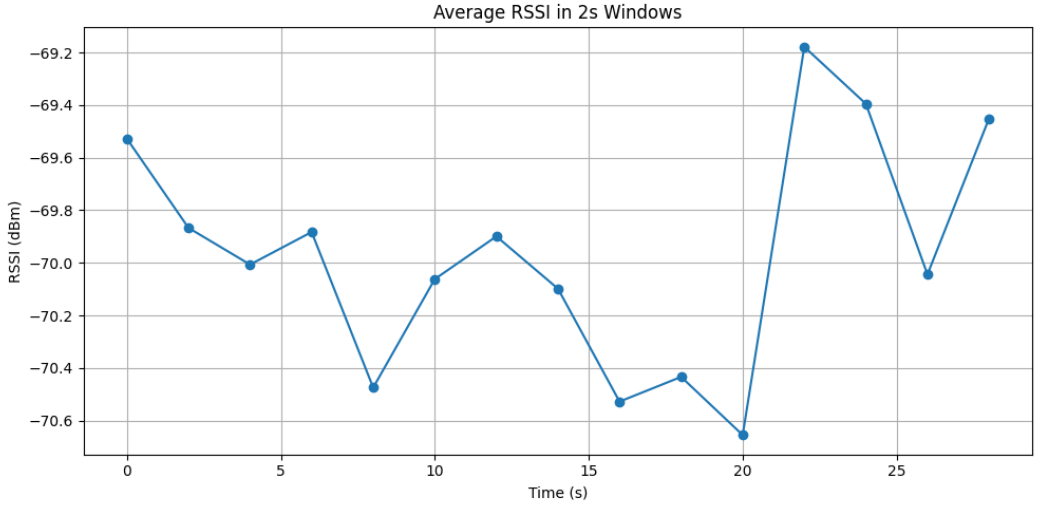


We observe that the two time series exhibit similar behavior, however iperf consistently shows a positive offset. This is expected, as iperf is a highly optimized tool, whereas our SpeedTest implementation lacks TCP-level optimizations (such as buffer sizing and pipelining) and incurs additional overhead due to the use of C code with Unix sockets.

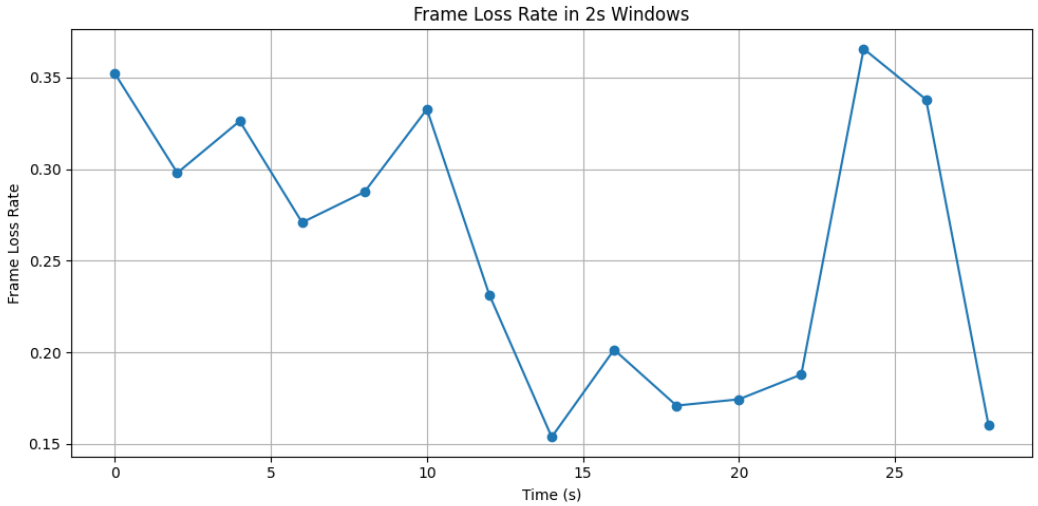
WiFi Doctor Analysis



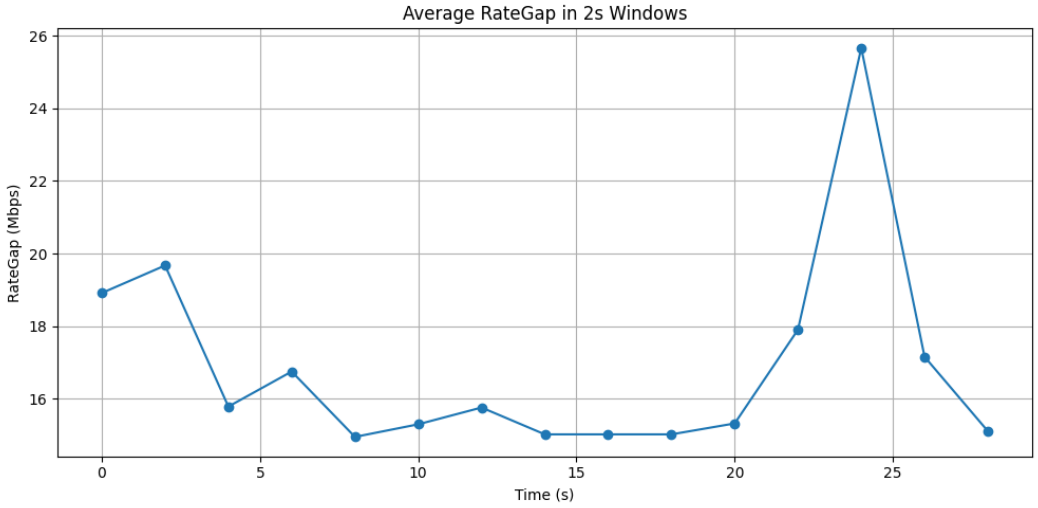
We observe that the data rate time series behaves similarly to our SpeedTest results, but in a more exaggerated manner. This is reasonable, as the data rate reflects only the theoretical maximum, as previously mentioned.



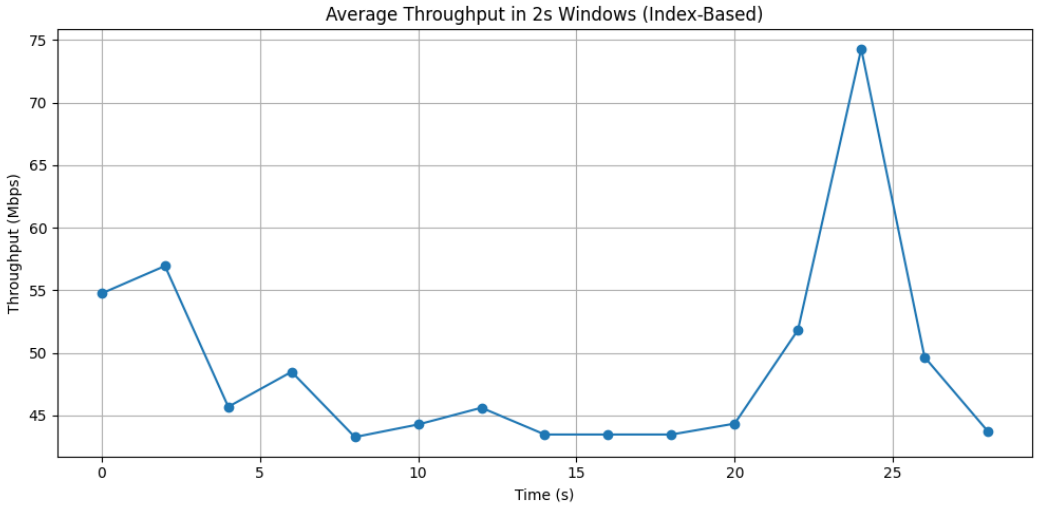
The low RSSI values indicate that we have successfully met our objective, suggesting that the experiment results are reliable.



In the low RSSI scenario the sender is forced to select lower MCS indexes, narrower bandwidths and longer guard intervals to maintain link reliability. Despite these conservative settings, which aim to improve robustness, we still observe a relatively high frame loss rate throughout the experiment—often exceeding 25%. This indicates that even under fallback configurations, the 5GHz band struggles with reliability at low RSSI levels. These losses are expected to significantly impact both throughput and rate gap, highlighting the sensitivity of 5GHz to signal attenuation without proper channel or topology planning.



We can observe that the plot of Rate Gap is identical to the one of Data Rate, but in a smaller number scale for the reasons previously mentioned. The large value of the Rate Gap for our given data rate can be justified due to the high frame loss rate.



Again, given that in our scenarios, we consider **Rate Gap = Data Rate − Throughput**, the plot of the throughput is again identical to the other two plots, but in a different scale.
Since **Throughput = (1-FrameLoss)·DataRate**, this is only a theoretical max throughput, based on the frameloss and the theoretical max data rate, and not a realistic depiction of the SpeedTest. It is evident that the throughput is noticeably lower, as expected due to the frame loss.

High RSSI, 2.4GHz

SpeedTest Analysis

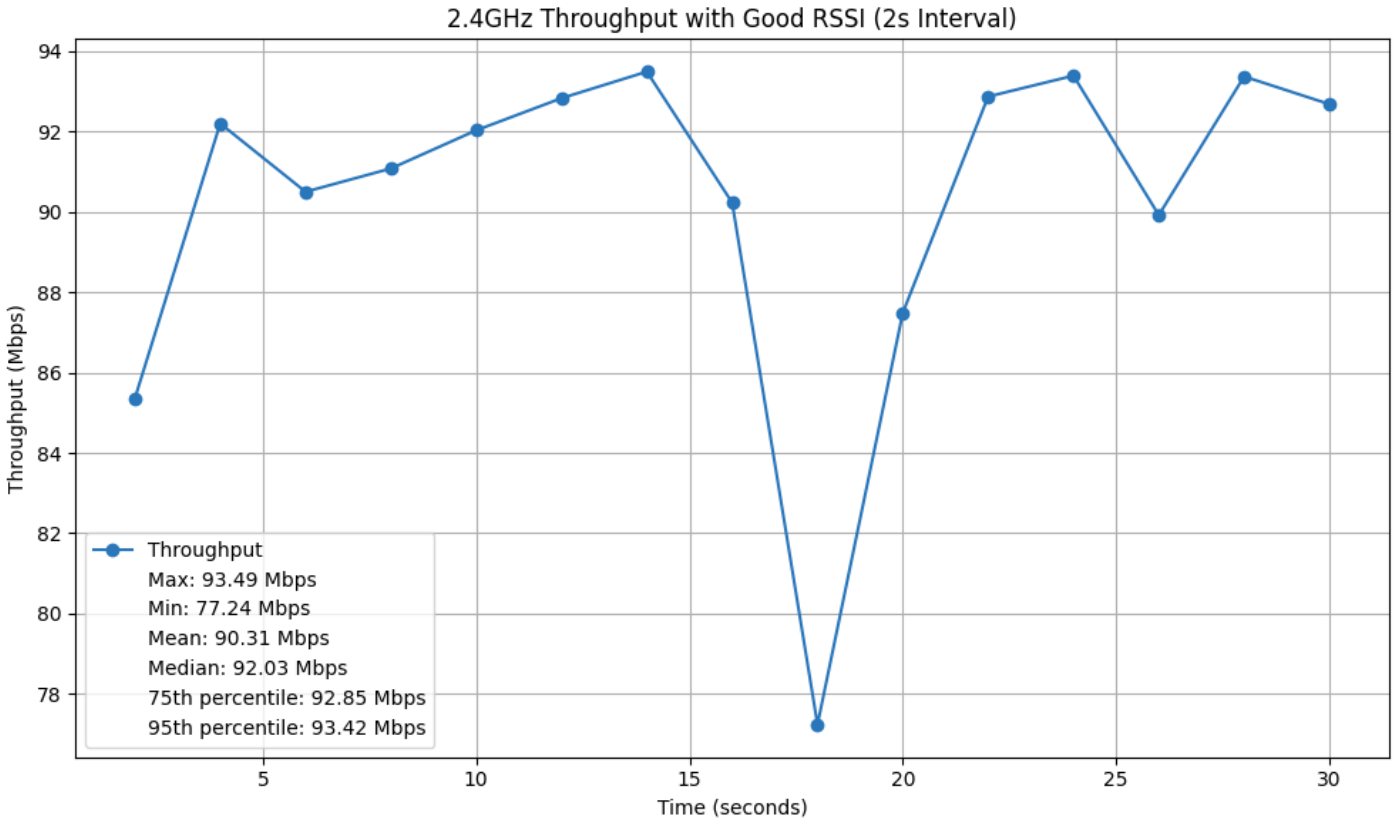
The results are:

```
Speed test started for this client...
[2.0 sec] Throughput: 85.36 Mbps
[4.0 sec] Throughput: 92.20 Mbps
[6.0 sec] Throughput: 90.50 Mbps
[8.0 sec] Throughput: 91.08 Mbps
[10.0 sec] Throughput: 92.03 Mbps
[12.0 sec] Throughput: 92.83 Mbps
[14.0 sec] Throughput: 93.49 Mbps
[16.0 sec] Throughput: 90.23 Mbps
[18.0 sec] Throughput: 77.24 Mbps
[20.0 sec] Throughput: 87.49 Mbps
[22.0 sec] Throughput: 92.87 Mbps
[24.0 sec] Throughput: 93.39 Mbps
[26.0 sec] Throughput: 89.92 Mbps
[28.0 sec] Throughput: 93.37 Mbps
[30.0 sec] Throughput: 92.68 Mbps
Total throughput over 30 sec: 90.31 Mbps
```

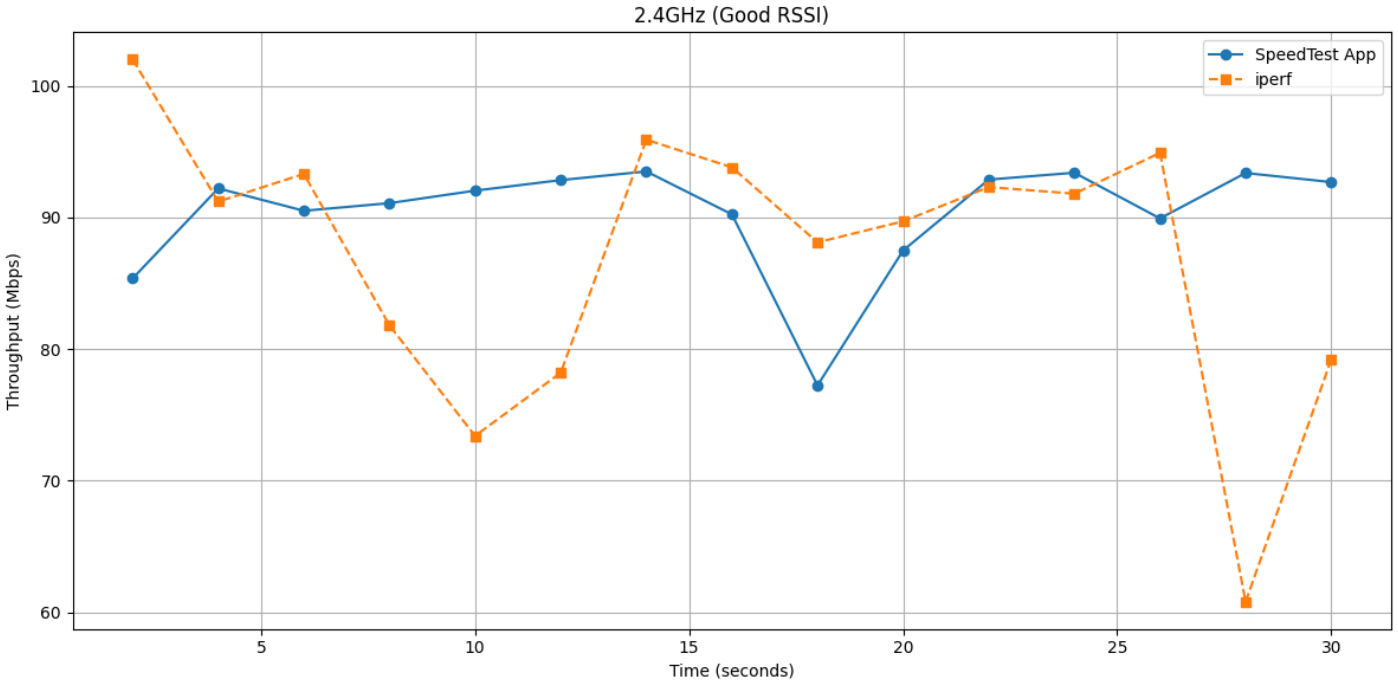
```
Connecting to host 192.168.1.5, port 5201
[  5] local 192.168.1.8 port 59600 connected to 192.168.1.5 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-2.00   sec  24.2 MBytes   102 Mbits/sec    0    782 KBytes
[  5]   2.00-4.00   sec  21.8 MBytes  91.2 Mbits/sec    0    918 KBytes
[  5]   4.00-6.00   sec  22.2 MBytes  93.3 Mbits/sec    0   1.04 MBytes
[  5]   6.00-8.00   sec  19.5 MBytes  81.8 Mbits/sec    4    863 KBytes
[  5]   8.00-10.00  sec  17.5 MBytes  73.4 Mbits/sec    0    979 KBytes
[  5]  10.00-12.00  sec  18.6 MBytes  78.2 Mbits/sec    0   1.02 MBytes
[  5]  12.00-14.00  sec  22.9 MBytes  95.9 Mbits/sec    0   1.02 MBytes
[  5]  14.00-16.00  sec  22.4 MBytes  93.8 Mbits/sec    0   1.04 MBytes
[  5]  16.00-18.00  sec  21.0 MBytes  88.1 Mbits/sec    0   1.05 MBytes
[  5]  18.00-20.00  sec  21.4 MBytes  89.7 Mbits/sec    5    775 KBytes
[  5]  20.00-22.00  sec  22.0 MBytes  92.3 Mbits/sec    0    967 KBytes
[  5]  22.00-24.00  sec  21.9 MBytes  91.8 Mbits/sec    0   1.02 MBytes
[  5]  24.00-26.00  sec  22.6 MBytes  94.9 Mbits/sec    4    769 KBytes
[  5]  26.00-28.00  sec  14.5 MBytes  60.8 Mbits/sec   58    652 KBytes
[  5]  28.00-30.00  sec  18.9 MBytes  79.2 Mbits/sec    0    926 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-30.00  sec   311 MBytes  87.1 Mbits/sec   71             sender
[  5]   0.00-30.02  sec   308 MBytes  86.2 Mbits/sec                  receiver

iperf Done.
```

Results of our SpeedTest (left), results of iperf (right)

A timeseries for the throughput, along with its statistics (max, min, mean, median, 75th percentile, 95th percentile).

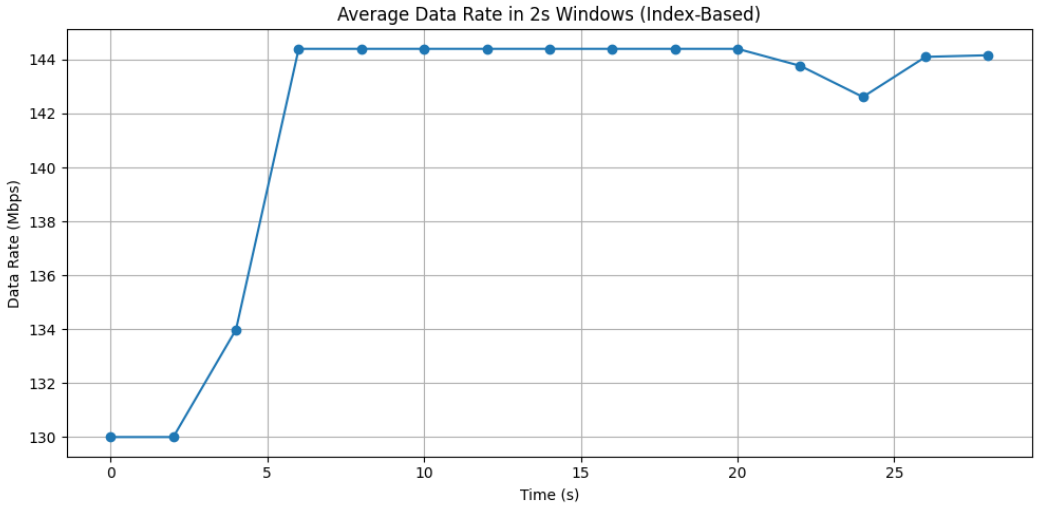


The timeseries of the SpeedTest - iperf comparison is :

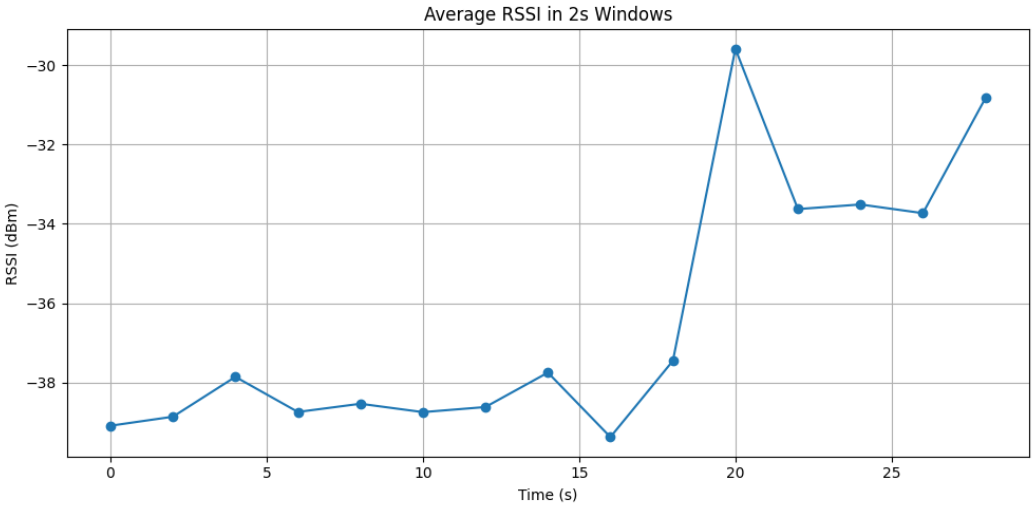


Once again, we observe that both time series have a similar average throughput, but iperf demonstrates greater variability. Its higher peaks and lower troughs indicate a more erratic behavior.

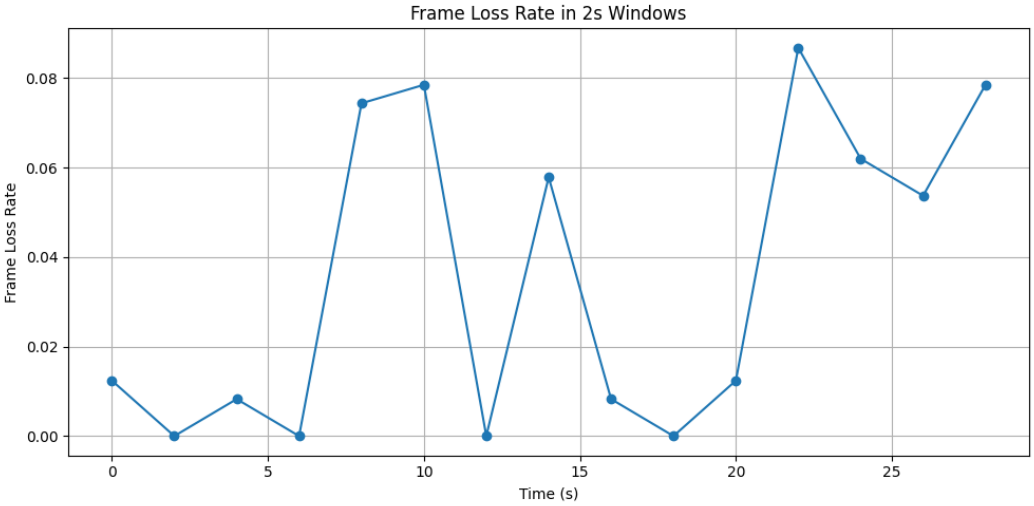
WiFi Doctor Analysis



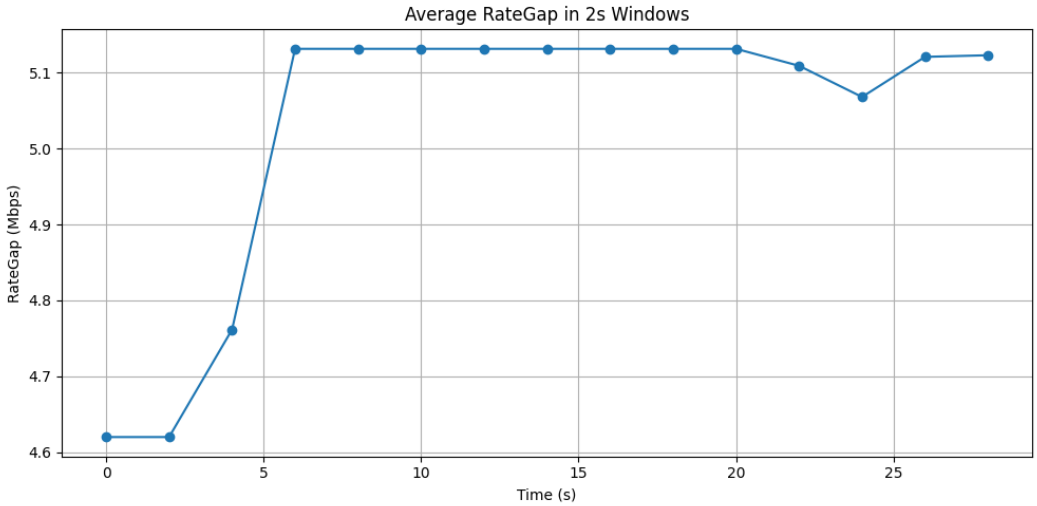
We can observe that results are greatly exaggerated compared to the actual SpeedTest. Data rate do not represent the actual value, only a theoretical maximum.



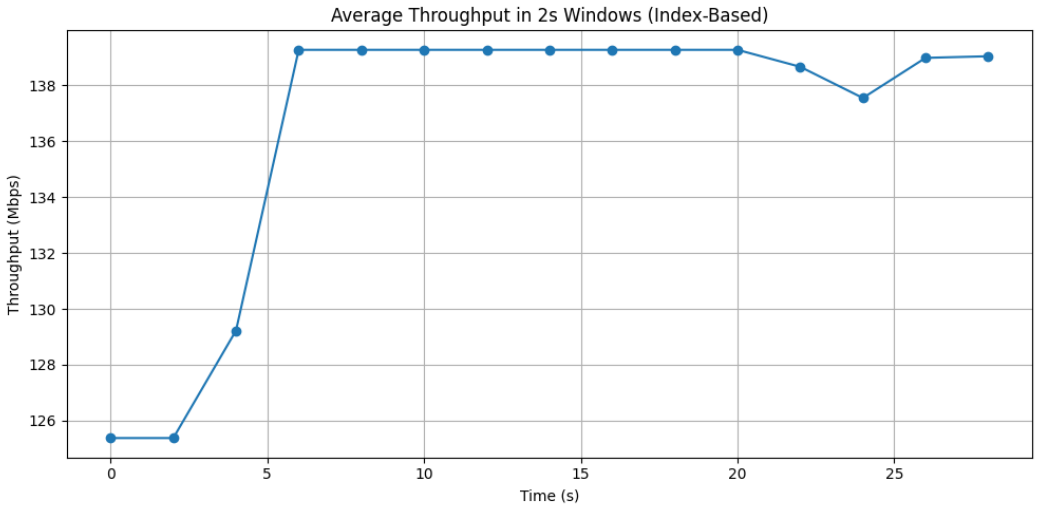
The great RSSI values indicate that we have successfully met our objective, suggesting that the experiment results are reliable. The stability of the RSSI is also the one responsible for the stability of the theoretical maximum data rate we observe above.



In the high RSSI scenario the sender utilizes higher MCS indexes and shorter guard intervals to maximize throughput. The bandwidth remains at 20MHz throughout the test. Despite these favorable conditions, the frame loss rate still exhibits non-negligible spikes—most notably exceeding 8%. These transient bursts of frame loss suggest that while signal strength is sufficient, other factors such as channel interference, collisions, or hardware buffering issues may intermittently impact link reliability.



As expected, the plot of Rate Gap is identical to the one of Data Rate, but in a smaller number scale for the reasons previously mentioned.



Again, given that in our scenarios, we consider **Rate Gap = Data Rate − Throughput**, the plot of the throughput is again identical to the other two plots, but in a different scale.
Since **Throughput = (1-FrameLoss)·DataRate**, this is only a theoretical max throughput, based on the frameloss and the theoretical max data rate, and not a realistic depiction of the SpeedTest. The fact that its value is so close to the data rate is due to the low frame loss rate.

Low RSSI, 2.4GHz

SpeedTest Analysis

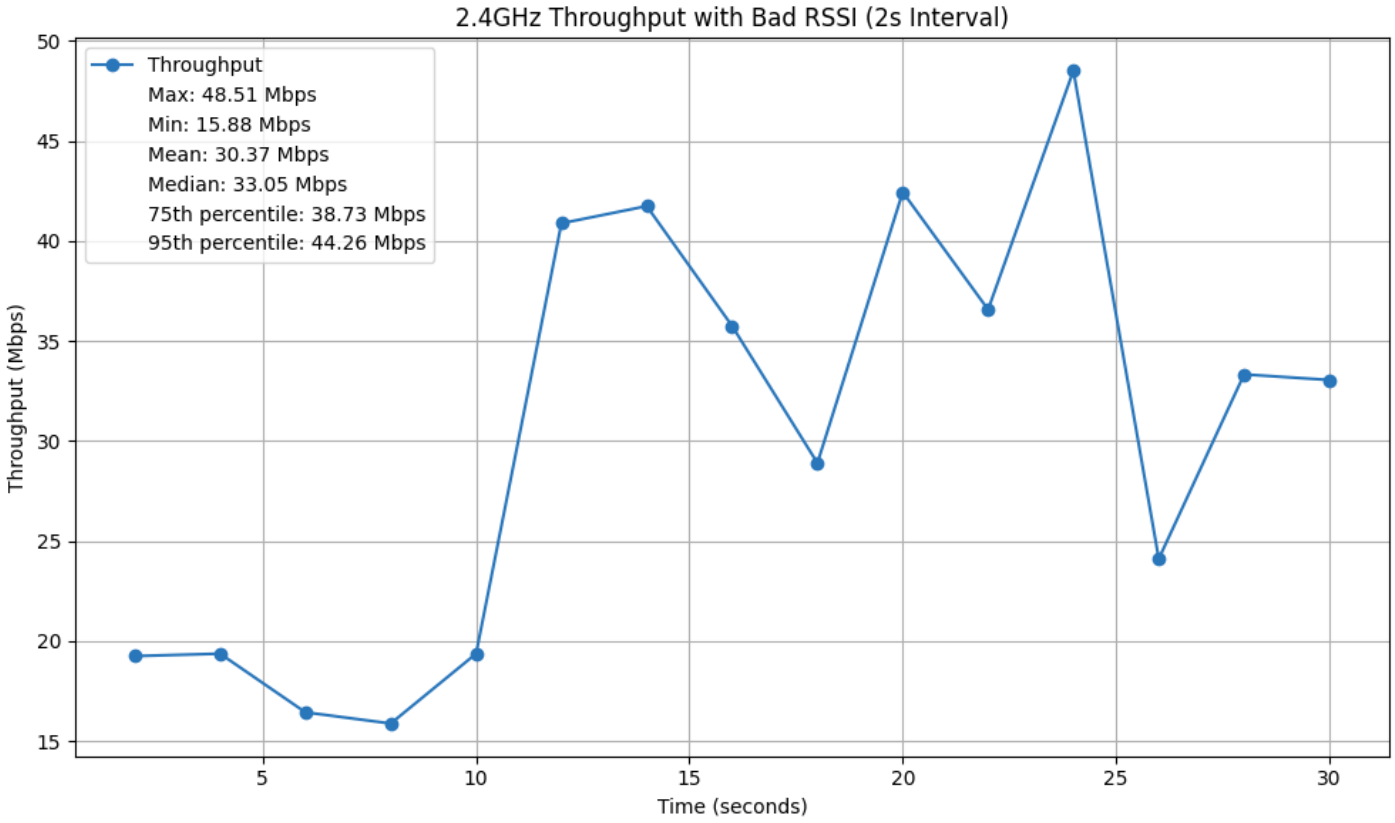
The results are:

```
Speed test started for this client...
[2.0 sec] Throughput: 19.25 Mbps
[4.0 sec] Throughput: 19.37 Mbps
[6.0 sec] Throughput: 16.44 Mbps
[8.0 sec] Throughput: 15.88 Mbps
[10.0 sec] Throughput: 19.37 Mbps
[12.0 sec] Throughput: 40.88 Mbps
[14.0 sec] Throughput: 41.74 Mbps
[16.0 sec] Throughput: 35.75 Mbps
[18.0 sec] Throughput: 28.90 Mbps
[20.0 sec] Throughput: 42.44 Mbps
[22.0 sec] Throughput: 36.57 Mbps
[24.0 sec] Throughput: 48.51 Mbps
[26.0 sec] Throughput: 24.11 Mbps
[28.0 sec] Throughput: 33.33 Mbps
[30.0 sec] Throughput: 33.05 Mbps
Total throughput over 30 sec: 30.37 Mbps
```

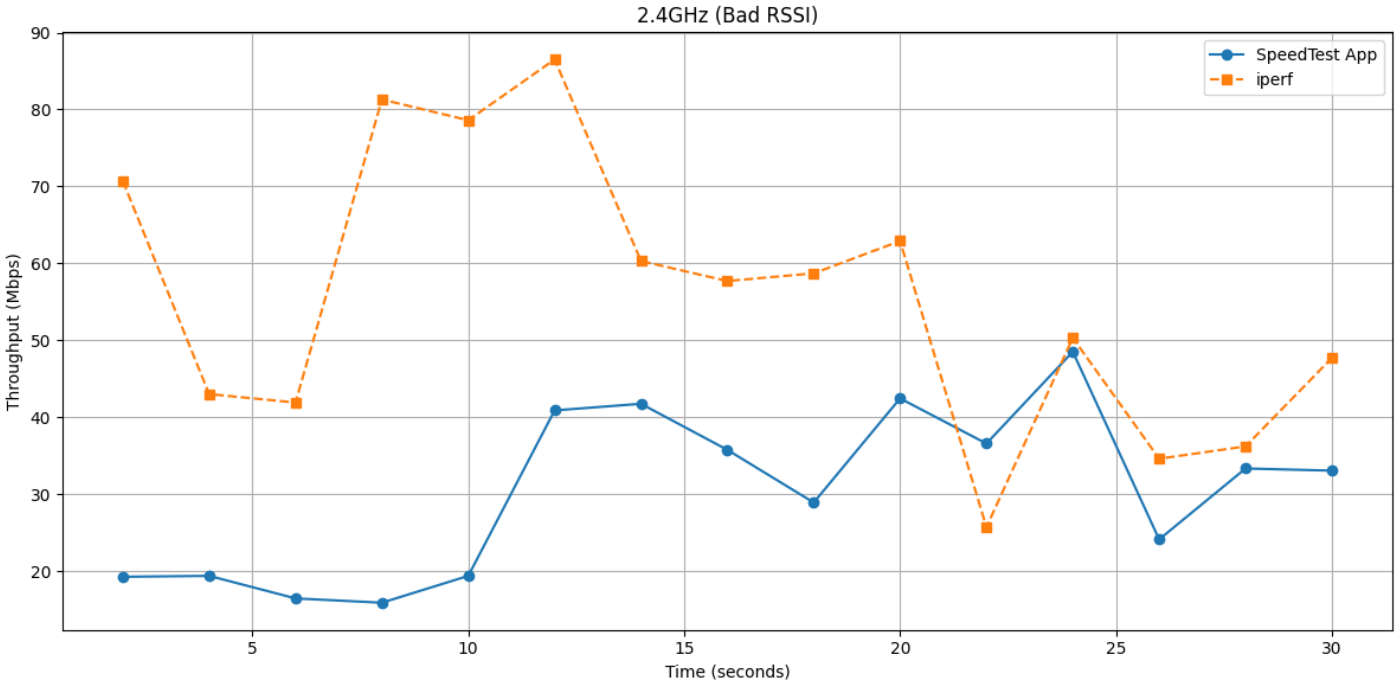
```
Connecting to host 192.168.1.5, port 5201
[  5] local 192.168.1.8 port 32784 connected to 192.168.1.5 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-2.00   sec  16.9 MBytes  70.7 Mbits/sec    0    786 KBytes
[  5]   2.00-4.00   sec  10.2 MBytes  43.0 Mbits/sec    0   1.23 MBytes
[  5]   4.00-6.00   sec  10.0 MBytes  41.9 Mbits/sec    0   1.63 MBytes
[  5]   6.00-8.00   sec  19.4 MBytes  81.3 Mbits/sec    0   2.06 MBytes
[  5]   8.00-10.00  sec  18.8 MBytes  78.6 Mbits/sec    4   1.44 MBytes
[  5]  10.00-12.00  sec  20.6 MBytes  86.5 Mbits/sec    0   1.77 MBytes
[  5]  12.00-14.00  sec  14.4 MBytes  60.3 Mbits/sec    0   1.94 MBytes
[  5]  14.00-16.00  sec  13.8 MBytes  57.7 Mbits/sec    0   2.01 MBytes
[  5]  16.00-18.00  sec  14.0 MBytes  58.7 Mbits/sec    0   2.01 MBytes
[  5]  18.00-20.00  sec  15.0 MBytes  62.9 Mbits/sec    0   2.01 MBytes
[  5]  20.00-22.00  sec  6.12 MBytes  25.7 Mbits/sec  1630    300 KBytes
[  5]  22.00-24.00  sec  12.0 MBytes  50.3 Mbits/sec  380   1.23 MBytes
[  5]  24.00-26.00  sec  8.25 MBytes  34.6 Mbits/sec    0   1.70 MBytes
[  5]  26.00-28.00  sec  8.62 MBytes  36.2 Mbits/sec    0   2.08 MBytes
[  5]  28.00-30.00  sec  11.4 MBytes  47.7 Mbits/sec    0   2.08 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-30.00  sec   199 MBytes  55.7 Mbits/sec  2014             sender
[  5]   0.00-30.22  sec   197 MBytes  54.7 Mbits/sec                   receiver

iperf Done.
```

Results of our SpeedTest (left), results of iperf (right)

A timeseries for the throughput, along with its statistics (max, min, mean, median, 75th percentile, 95th percentile).

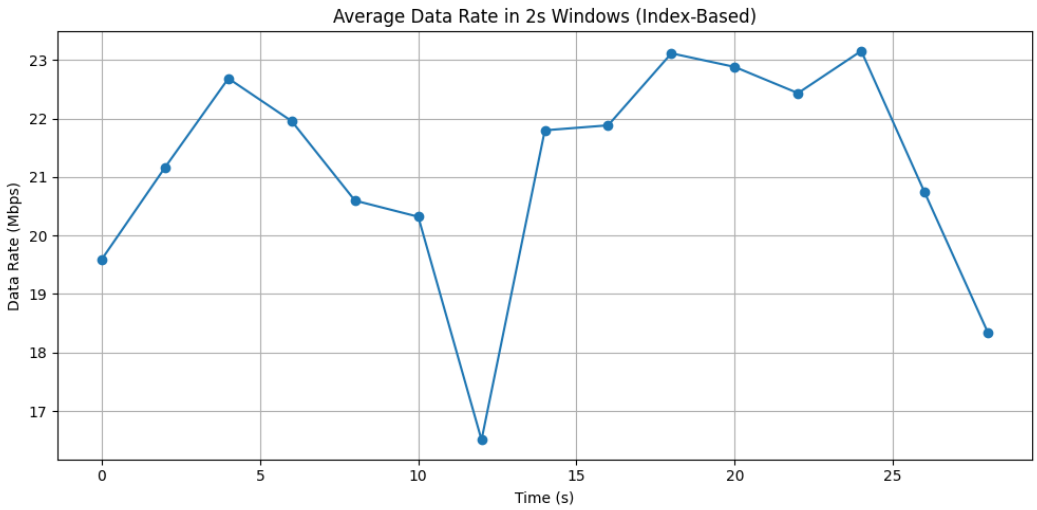


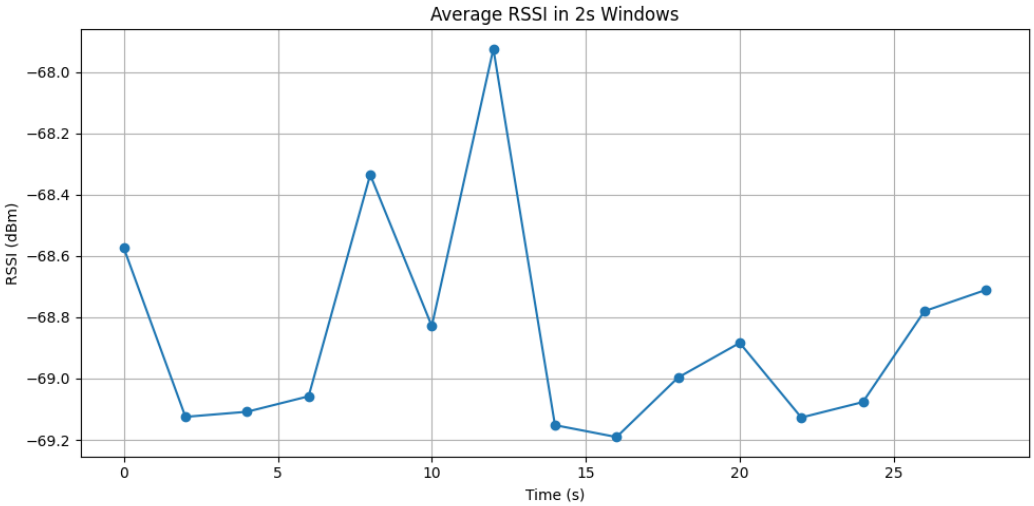
The timeseries of the SpeedTest - iperf comparison is :



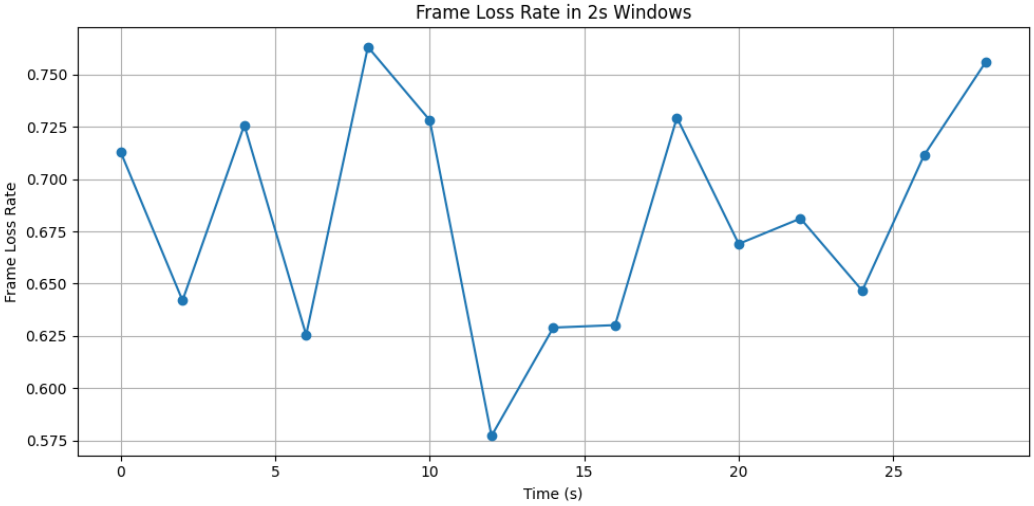
Just as in the second case, we observe that iperf performs better. We can now safely conclude that our SpeedTest implementation, which lacks optimizations, does not perform as well under low RSSI conditions.

WiFi Doctor Analysis

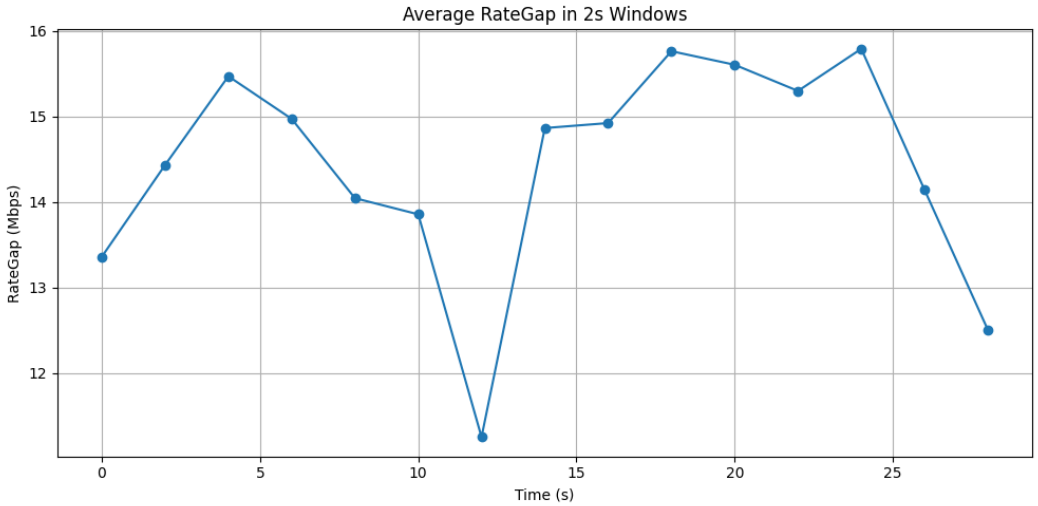




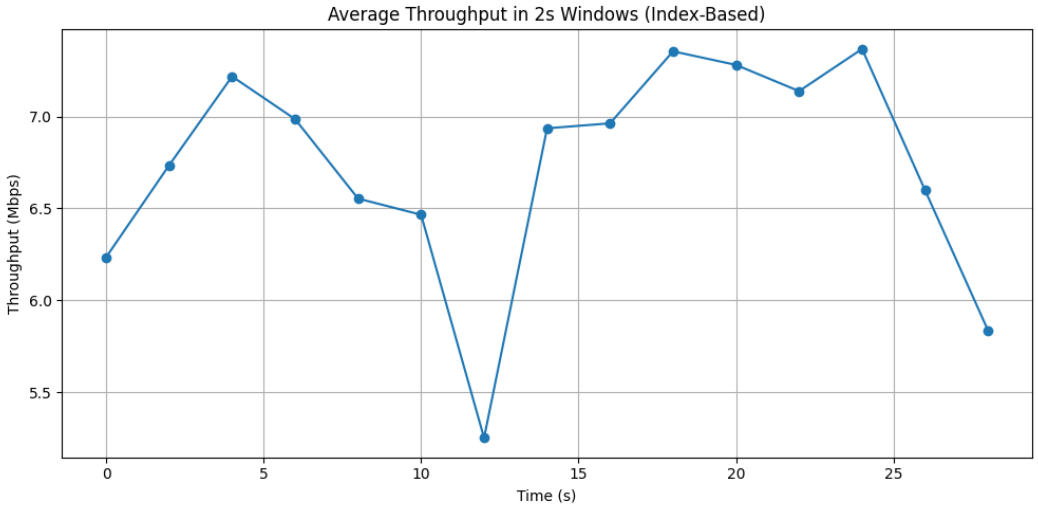
The low RSSI values indicate that we have successfully met our objective, suggesting that the experiment results are reliable.



In the low RSSI scenario the frame loss rate remains persistently high throughout the experiment—generally ranging between 60% and 75%, with multiple spikes approaching or exceeding 75%. This suggests that the client is experiencing significant link degradation, likely due to weak signal strength and possibly increased channel noise or interference. Low RSSI conditions severely impair reliability, forcing the sender to use more conservative PHY layer configurations (e.g., lower MCS indexes and narrower channel widths). However, even these adjustments fail to sufficiently mitigate losses, resulting in a consistently elevated frame loss rate.



Given that in our scenarios, we consider **Rate Gap = Data Rate – Throughput**, we can observe that the plot of Rate Gap is identical to the one of Data Rate, but in a smaller number scale, of course. Its large values are justified by the high frame loss rate.



Once again, given that in our scenarios, we consider **Rate Gap = Data Rate – Throughput**, the plot of the throughput is again identical to the other two plots, but in a different scale.
Since **Throughput = (1-FrameLoss)·DataRate**, this is only a theoretical max throughput, based on the frameloss and the theoretical max data rate, and not a realistic depiction of the SpeedTest.

Variable RSSI (Mobility) , 5GHz

For the mobility test, we started the SpeedTest, and after that the iperf test, next to the AP. In the 30-second duration, we kept walking with the server away from the AP inside the apartment building in order to get a observe a gradual decline of RSSI and data rate.

SpeedTest Analysis

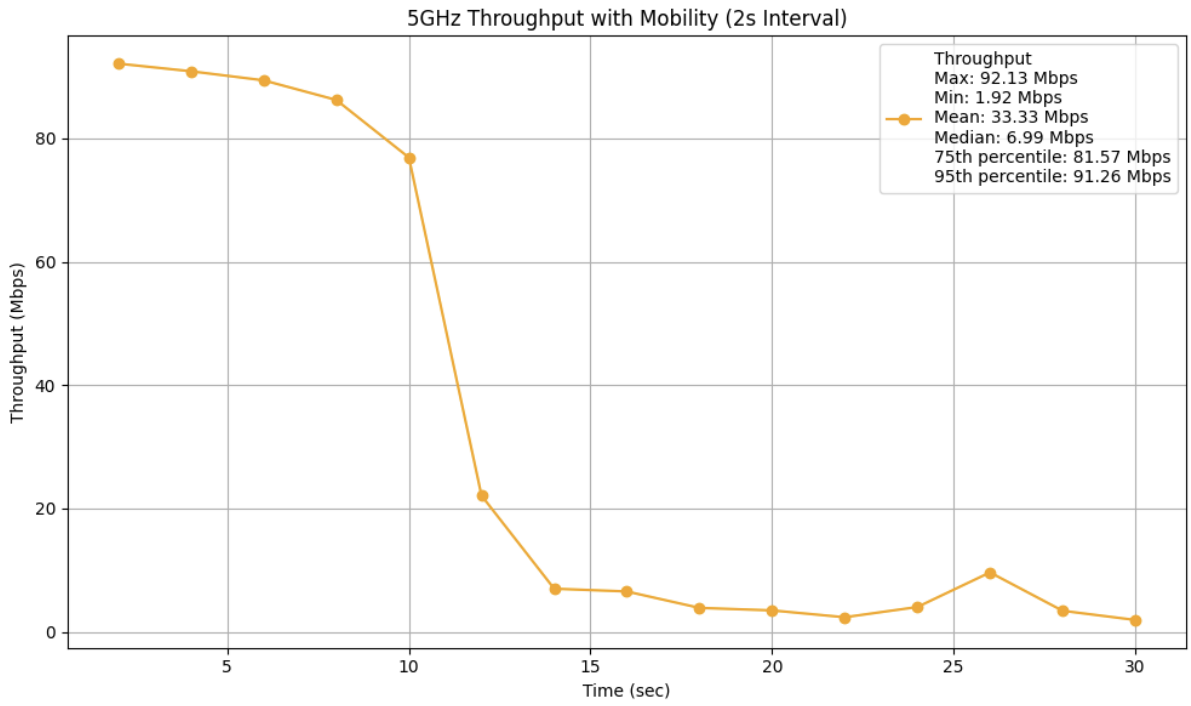
The results are:

```
Speed test started for this client...
[2.0 sec] Throughput: 92.13 Mbps
[4.0 sec] Throughput: 90.89 Mbps
[6.0 sec] Throughput: 89.42 Mbps
[8.0 sec] Throughput: 86.24 Mbps
[10.0 sec] Throughput: 76.90 Mbps
[12.0 sec] Throughput: 22.08 Mbps
[14.0 sec] Throughput: 6.99 Mbps
[16.0 sec] Throughput: 6.54 Mbps
[18.0 sec] Throughput: 3.88 Mbps
[20.0 sec] Throughput: 3.47 Mbps
[22.0 sec] Throughput: 2.36 Mbps
[24.0 sec] Throughput: 4.02 Mbps
[26.0 sec] Throughput: 9.63 Mbps
[28.0 sec] Throughput: 3.41 Mbps
[30.0 sec] Throughput: 1.92 Mbps
Total throughput over 30 sec: 33.33 Mbps
```

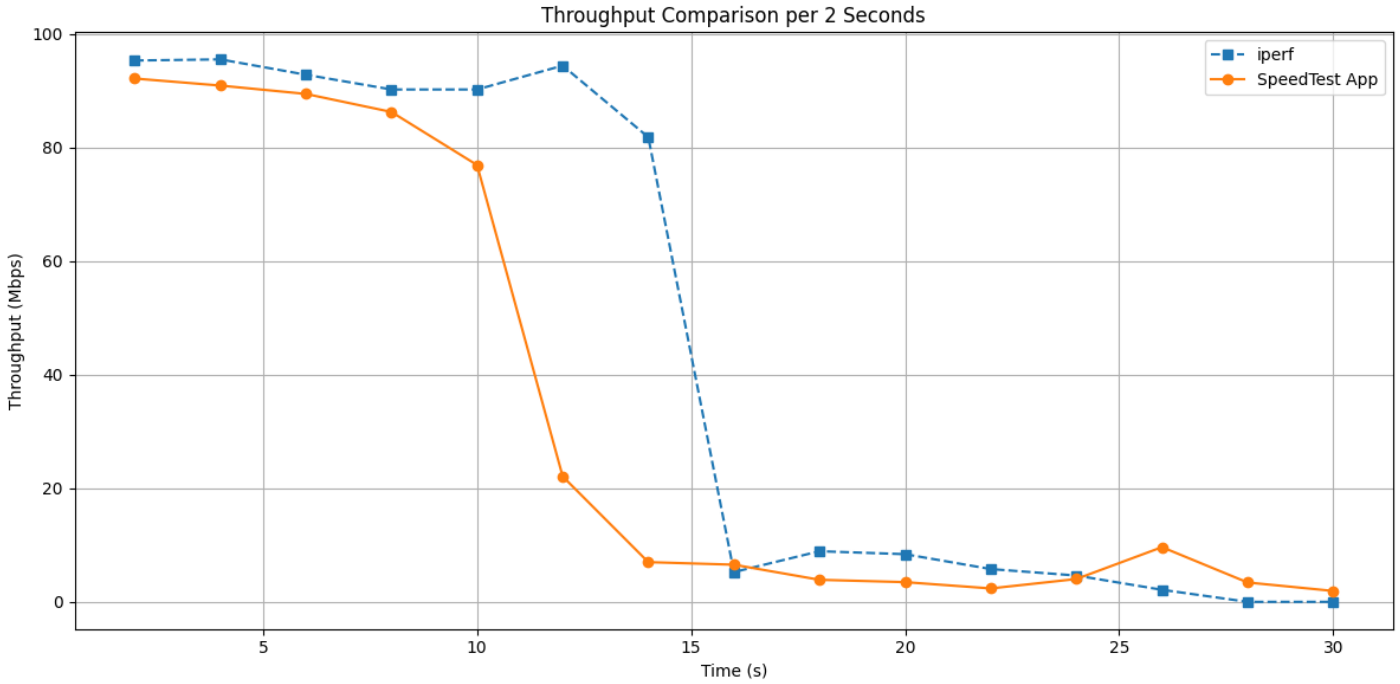
```
Connecting to host 192.168.1.5, port 5201
[  5] local 192.168.1.8 port 37420 connected to 192.168.1.5 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-2.00   sec  22.8 MBytes  95.3 Mbits/sec    0    205 KBytes
[  5]   2.00-4.00   sec  22.8 MBytes  95.5 Mbits/sec    0    266 KBytes
[  5]   4.00-6.00   sec  22.1 MBytes  92.8 Mbits/sec    0    410 KBytes
[  5]   6.00-8.00   sec  21.5 MBytes  90.2 Mbits/sec    0    611 KBytes
[  5]   8.00-10.00  sec  21.5 MBytes  90.2 Mbits/sec    0    611 KBytes
[  5]  10.00-12.00  sec  22.5 MBytes  94.4 Mbits/sec    0    925 KBytes
[  5]  12.00-14.00  sec  19.5 MBytes  81.8 Mbits/sec    0   1.41 MBytes
[  5]  14.00-16.00  sec  1.25 MBytes  5.24 Mbits/sec  269   1.30 MBytes
[  5]  16.00-18.00  sec  2.12 MBytes  8.91 Mbits/sec   30   1.30 MBytes
[  5]  18.00-20.00  sec  2.00 MBytes  8.39 Mbits/sec   42   86.3 KBytes
[  5]  20.00-22.00  sec  1.38 MBytes  5.77 Mbits/sec   65   94.7 KBytes
[  5]  22.00-24.00  sec  0.00 Bytes  0.00 bits/sec   14   74.9 KBytes
[  5]  24.00-26.00  sec  1.38 MBytes  5.76 Mbits/sec   10   43.8 KBytes
[  5]  26.00-28.00  sec  0.00 Bytes  0.00 bits/sec    1   42.4 KBytes
[  5]  28.00-30.00  sec  1.38 MBytes  5.77 Mbits/sec   13   49.5 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-30.00  sec   162 MBytes  45.3 Mbits/sec  444             sender
[  5]   0.00-30.06  sec   159 MBytes  44.4 Mbits/sec                  receiver

iperf Done.
```

Results of our SpeedTest (left), results of iperf (right)

A timeseries for the throughput, along with its statistics (max, min, mean, median, 75th percentile, 95th percentile).

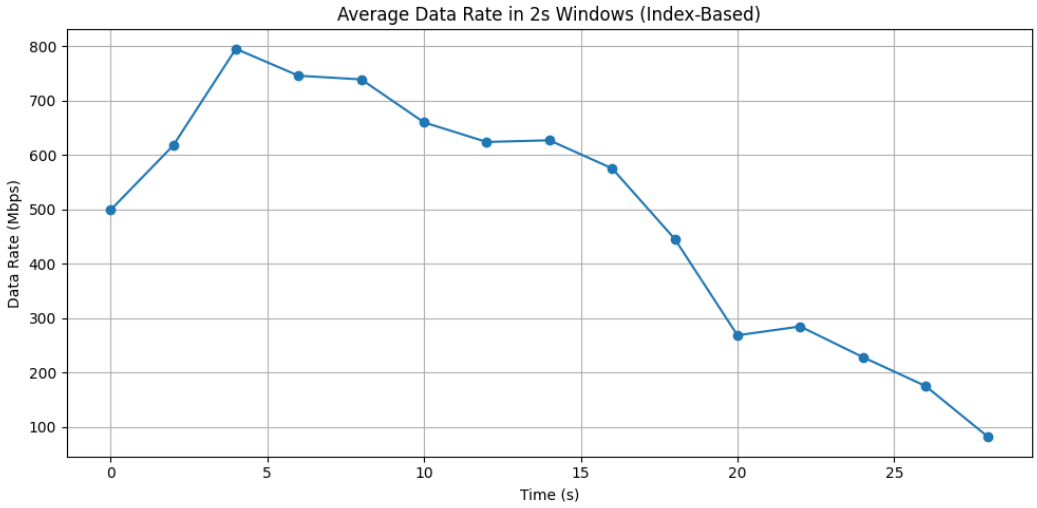


The timeseries of the SpeedTest - iperf comparison is :

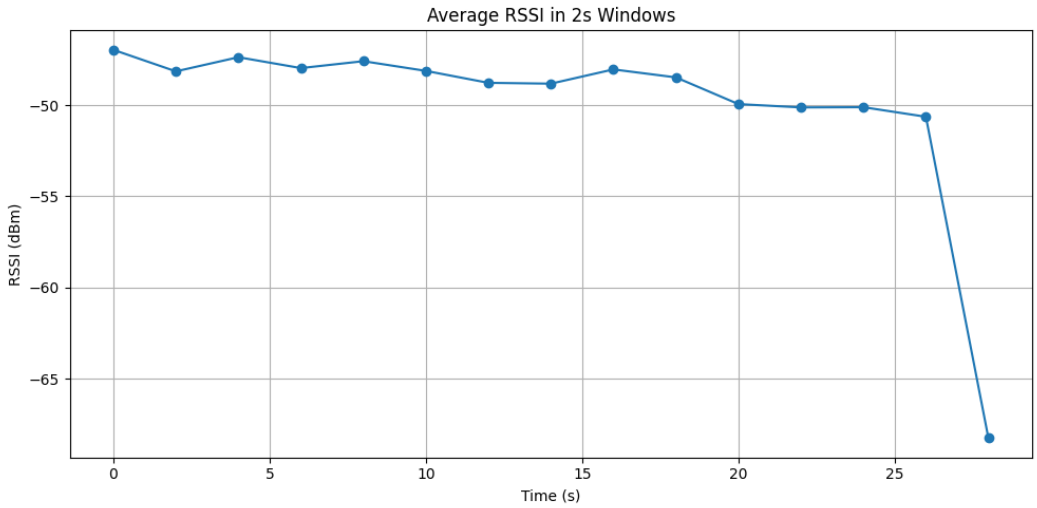


Both tools follow a similar pattern. Initially, they achieve high throughput (up to around 14 seconds) while the server is located close to the access point (AP). After the 14-second mark, we observe a sharp drop in throughput, in both tools, caused by the increasing distance from the AP, eventually resulting in a (near) zero data rate.

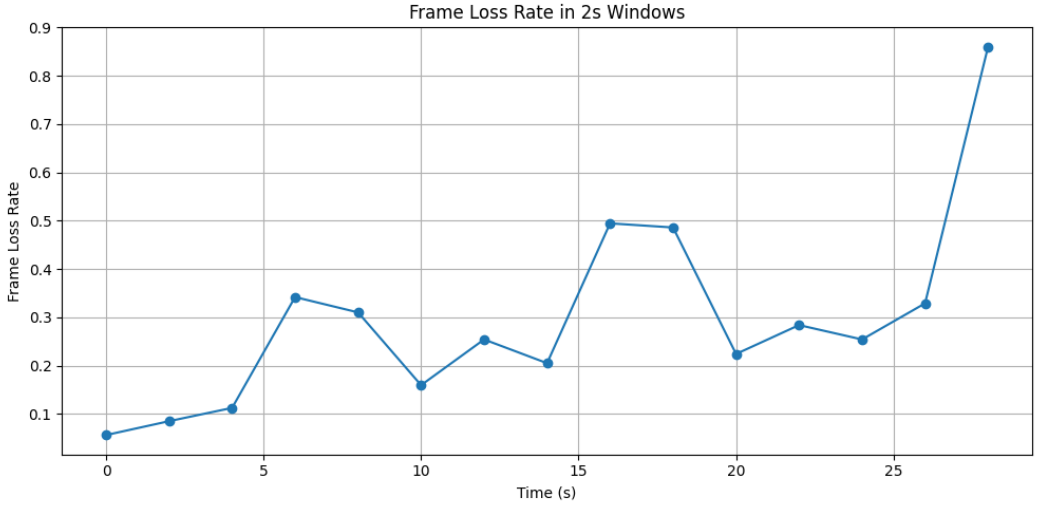
WiFi Doctor Analysis



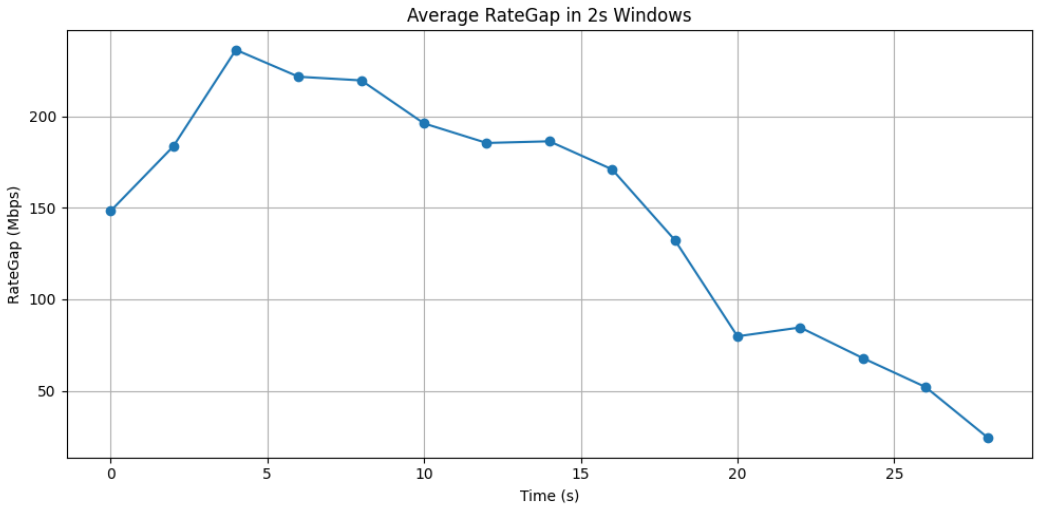
At the beginning of the experiment, when we are close to the AP, the performance resembles that of the first scenario. However, as time passes and the RSSI decreases, the performance starts to resemble the second scenario.



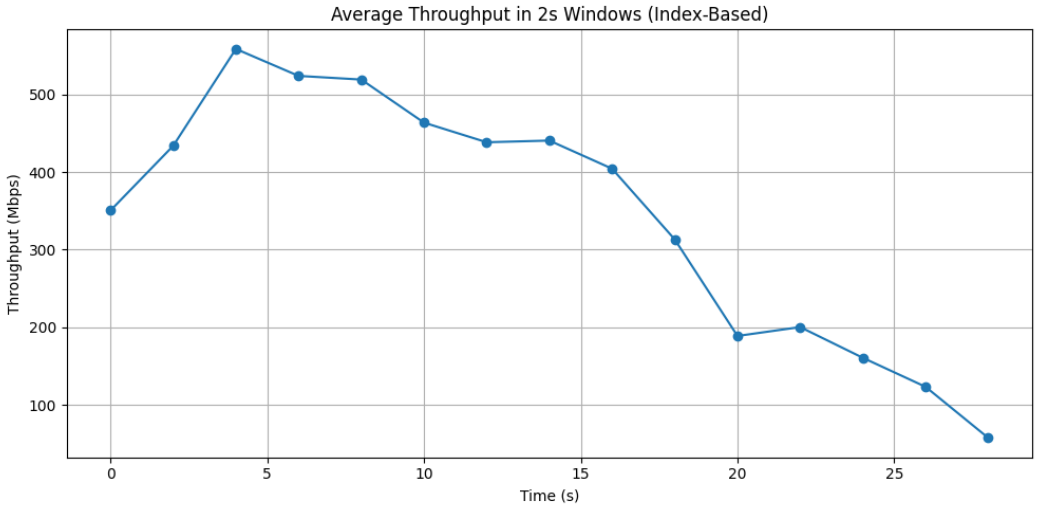
The RSSI starts off strong and gradually worsens over time, as expected, due to the increasing distance between the server and the access point.



The frame loss rate increases as RSSI decreases, eventually reaching extreme values close to one. Its values are higher than in scenario two because, during the experiment, physical obstructions such as walls and doors increased.



Given that in our scenarios, we consider **Rate Gap = Data Rate − Throughput**, we can observe that the plot of Rate Gap is identical to the one of Data Rate, but in a smaller number scale, of course.



Once again, given that in our scenarios, we consider **Rate Gap = Data Rate − Throughput**, the plot of the throughput is again identical to the other two plots, but in a different scale.
Since **Throughput = (1-FrameLoss)·DataRate**, this is only a theoretical max throughput, based on the frameloss and the theoretical max data rate, and not a realistic depiction of the SpeedTest.

Final Scenario Comparison

2.4 GHz, Low RSSI vs. 5GHz, Low RSSI

Under low RSSI conditions, both frequency bands suffered from degraded performance, but in different ways. The 5GHz band, though it maintained lower frame loss rates (around 25%), saw a sharp reduction in data rate as the signal weakened. This is due to its higher frequency, which is more sensitive to attenuation and performs poorly through walls or over distance. On the other hand, the 2.4GHz band with its longer wavelength, preserved link stability more effectively and maintained connectivity in areas where 5GHz struggled. However, this came at the cost of significantly higher frame loss-often exceeding 70%-and a much larger rate gap, revealing inefficiencies in data transmission. Furthermore, 2.4GHz exhibited the highest channel utilization (0.296), indicating increased contention and interference. While neither performs well under low RSSI, **2.4GHz band is marginally more resilient at maintaining connection**, while 5GHz degrades more quickly. In low-signal environments, 2.4GHz may provide a more reliable fallback - though with poor throughput.

2.4 GHz, High RSSI vs. 5GHz, High RSSI

In high RSSI conditions, both the 2.4GHz and 5GHz frequency bands delivered solid performance; however, a closer inspection of link quality metrics reveals key differences. The 5GHz band exhibited higher data rates due to its ability to utilize wider channel bandwidths and more advanced modulation schemes, resulting in theoretical capacities far exceeding those of 2.4GHz. Despite this advantage, the 5GHz scenario also showed slightly elevated frame loss (above 8%) and noticeable spikes in rate gap, suggesting brief periods of inefficiency potentially caused by hardware buffering or momentary interference. The 2.4GHz band, while limited by narrower channels and lower modulation potential, demonstrated great stability - maintaining a near-zero rate gap and minimal channel utilization (0.012 compared to 0.124 in 5GHz), indicating efficient use of its available capacity. However, despite its consistency, 2.4GHz is ultimately outperformed by 5GHz in high signal strength scenarios, where throughput demands are prioritized. Thus, **5GHz is the optimal choice** when strong signal quality is available, especially for high-bandwidth applications.

Conclusion

Through testing across varying RSSI conditions, it is evident that signal strength plays a critical role in determining the optimal Wi-Fi frequency band. While 5GHz offers superior throughput and is the clear choice in environments with strong signal quality, its performance quickly degrades with distance and obstruction. In contrast, 2.4GHz provides more stable and resilient connections in low RSSI scenarios, however at the cost of speed and efficiency. Therefore, the ideal band selection depends not solely on theoretical capabilities, but on the specific environmental conditions and application requirements present in a given network deployment.

Re-evaluating Throughput Formula

So far, our Wi-Fi Doctor has estimated the throughput using the formula **Throughput = (1-FrameLoss)·DataRate**
We can see in the graphs, and we mentioned before, that this formula is not representative of the actual SpeedTest results, mainly because in Wi-Fi Doctor, Throughput is calculated using the theoretical PHY rate as Data Rate.
We will re-imagine the throughput formula, adding in one more factor : **channel utilization.**
Channel utilization is the fraction of time that the wireless channel is occupied by any transmission, essentially "how much of the time the channel is busy, and how much it is idle".
Including channel utilization in the throughput formula is a significant upgrade to the previous one, because the busier the channel, the less airtime is available for our own frames.
To estimate channel utilization, we use the formula :

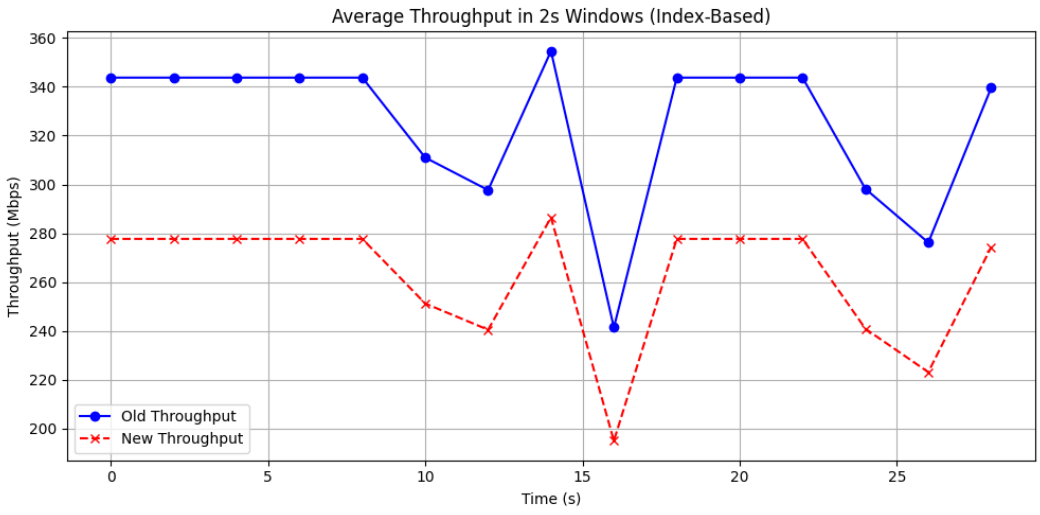
$$ChannelUtilization = \frac{TotalBusyTime}{TotalObservationTime}$$, where:

Total busy time : Sum of airtime occupied by each of our frames,
Total observation time : Time difference between the first and last captured frame (should be around 30sec, in our case)
The new formula of Throughput becomes:

$$Throughput = DataRate * (1 - Frameloss)(1 - ChannelUtilization)$$

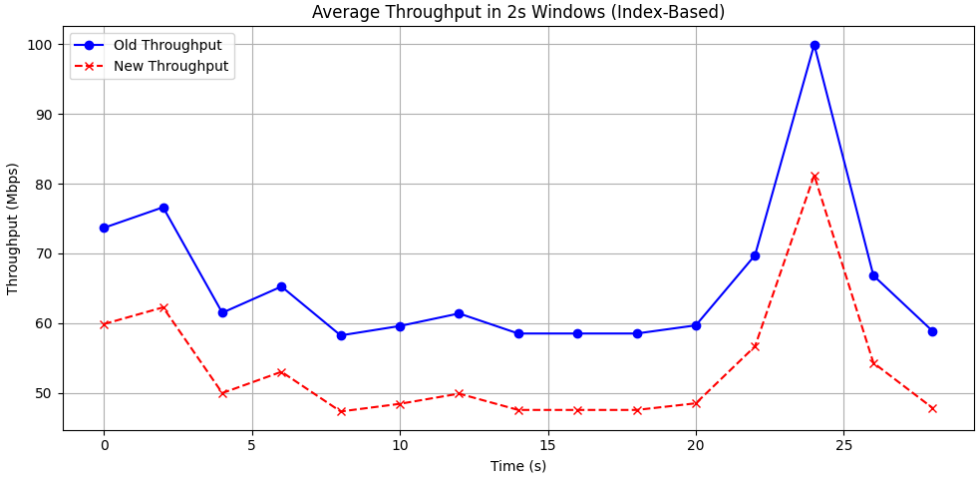
With this improved formula, what we expect to see is **throughput values that are realistic, given our line capabilities (max. 300Mbps).** Of course, it will not match the SpeedTest values, since we mentioned above that our ethernet cable bottlenecks the last-mile part (client side) and caps it to 100Mbps, and Wireshark does not understand that. Wireshark only sees MCS indices that indicate how capable our Wi-Fi connection is.
Therefore, we definitely expect to see lower and more realistic values of throughput than before, **however not necessarily < 300Mbps (in good RSSI, of course)**, since this is merely a better approximation, not a precise one.

- **5GHz - High RSSI**



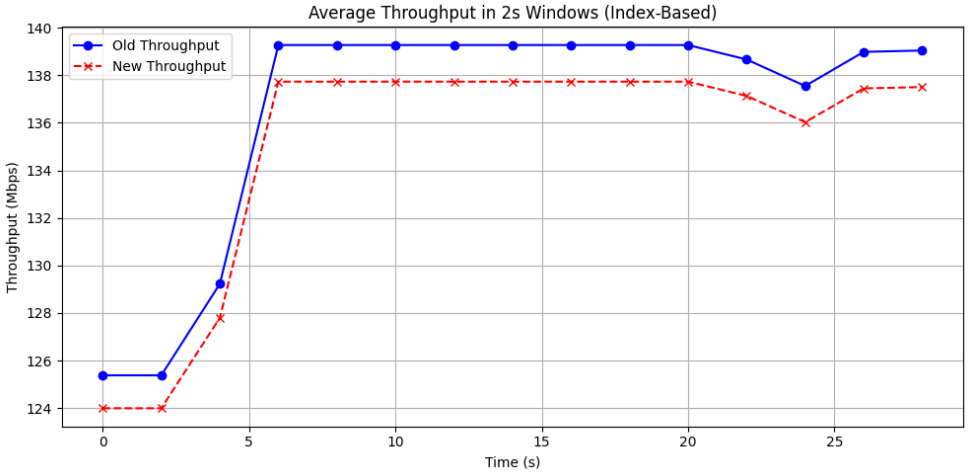
Channel Utilization 0.124

- **5GHz - Low RSSI**



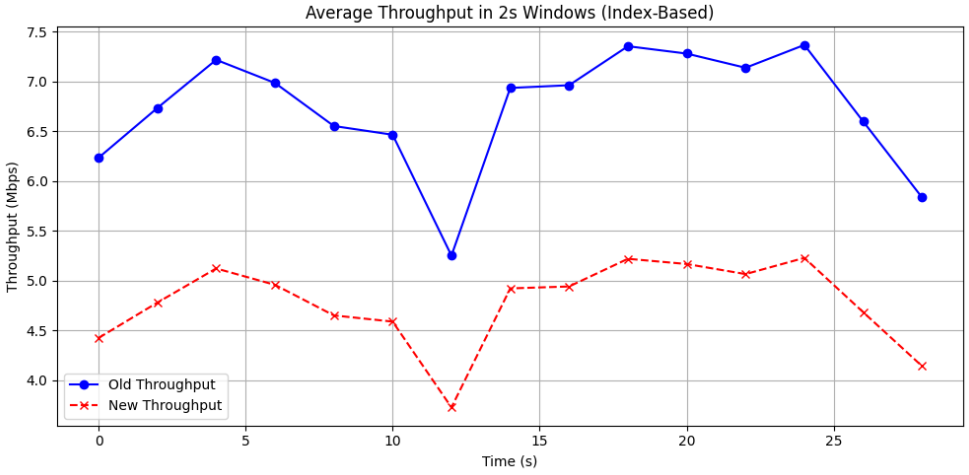
Channel Utilization = 0.187

- **2.4GHz - High RSSI**



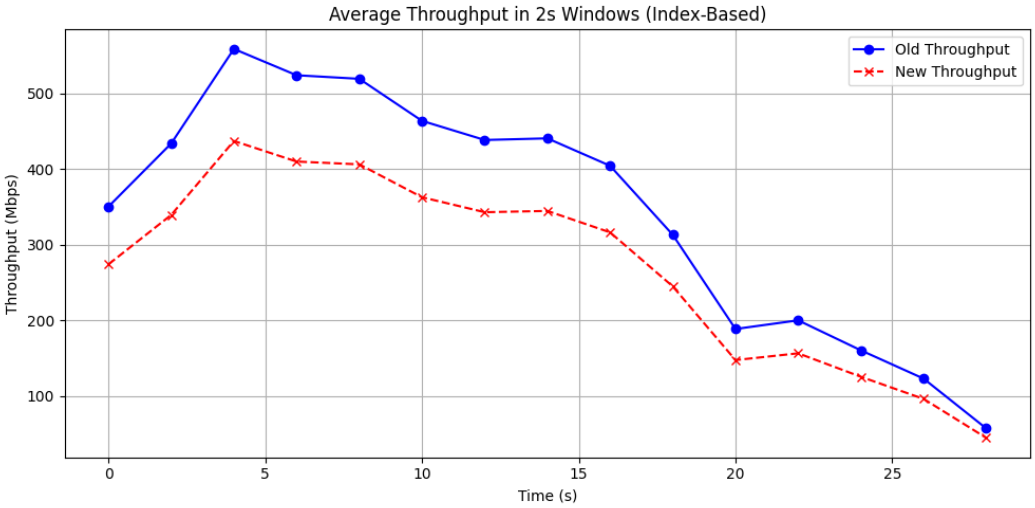
Channel Utilization = 0.012

- **2.4GHz - Low RSSI**



Channel Utilization = 0.296

- **5GHz - Variable RSSI (Mobility)**



Channel Utilization = 0.214