UiO : Faculty of Mathematics and Natural Sciences
University of Oslo

Wi-Fi networks Interference

Measuring and quantifying the network throughput performance suppression at 2.4 GHz

Kun Zhu

Master thesis

2024

Abstract

As of 2024, approximately 5.35 billion people are using the Internet, which represents about 66.2% of the global population [1]. While specific statistics on Wi-Fi usage are less frequently reported, it's important to note that Wi-Fi remains a dominant method for Internet access, with billions of devices relying on it for connectivity. For instance, it's projected that 4.1 billion Wi-Fi devices will be shipped in 2024, highlighting the technology's continued relevance and widespread adoption [2].

Radio interference can significantly reduce the throughput performance of a wireless network. In my findings, for a 2.4 GHz wireless network, the interference from a neighbouring wireless network at the same channel can reduce 60% of the throughput performance to that of the best throughput performance without interference. That is to say, when you are paying for 100 Mbit/s Internet from your Internet Service Provider (ISP), you can lose as much as 60% of the money you are paying for.

When this is put into an industrial level, I would say that with disrupted or malfunctioning Wi-Fi connections, devices will not function as expected, smart homes will not function as designed, factories will face damaged production lines, people will have to face high, slow connections to others and inconvenience both in their personal lives and workplaces, and many may face financial losses. Thus, a stable Wi-Fi connection is crucial for the users. We believe that knowing how much throughput performance is reduced can help us make better use of networks.

Acknowledgements

This topic was brought up by Torleiv Maseng, who unfortunately passed away in the summer of 2023.

Later Magnus Skjegstad took over the responsibilities of supervising this project together with co-supervisor Tor Skeie. We changed the topic to the current setup to focus on measurements and comparisons. A special thanks to the Norwegian Defence Research Establishment (FFI), the Faraday cage provider. This device allowed me to investigate network performance at different attenuation levels.

Contents

List of Figures

List of Tables

Acronyms

ACI Adjacent channel interference. 2, 3, 6, 11, 14, 15, 25–41, 44, 45, 47, 48

AP Access point. 2, 7, 11, 14

API Application Programming Interface. 9, 15, 16, 49

CCI Co-channel interference. 2, 3, 6, 11, 14, 15, 25–34, 36, 37, 39–41, 44, 45, 47, 48

ISP Internet Service Provide. i, 2

MAC Medium access control. 7

NI None-channel interference. 11, 14, 15, 25–40, 44, 45, 47–49

Std Standard deviation. vii, viii, 24, 32–35, 53

TCP Transmission control protocol. 8

CHAPTER 1

Introduction

The IEEE 802.11 standard for wireless local area networking (WLAN), commercially known as Wi-Fi, is a necessity in our modern day-to-day life. Unlike wired counterparts, wireless computing provides network connectivity without needing physical cables. WLANs offer high bandwidth in limited geographical areas [3]. The technology has developed from Wi-Fi0 (IEEE 802.11) at max 2 Mb/s to, today's most commonly used, Wi-Fi4 (IEEE 802.11n) at max 600 Mb/s and Wi-Fi5 (IEEE 802.11ac) at max 6933 Mb/s. Newer versions of Wi-Fi with higher link throughput are under continuous development. The latest applicable Wi-Fi6 (IEEE 802.11ax) can reach as high as 9608 Mb/s. For Wi-Fi7 (IEEE 802.11be), numerous products were announced in 2022 based on draft standards [4], and it was available in retail. The availability of the final version of Wi-Fi7 is estimated at the end of 2024. In the coming years, the final version of Wi-Fi8 (IEEE 802.11bn) will be adopted in 2028 [5].

In recent years, "smart" devices and "smart" homes, which can connect to the internet or a central control unit to perform tasks automatically or remotely, have become increasingly prominent and important in our daily lives. Not only computers/laptops but also IoT devices, smartphones, smart TVs, fridges, household cleaning robotics and most other kinds of smart devices are equipped with Wi-Fi modules, They all require wireless connections to achieve the "smart" functionalities [6]-[7].

Apart from consumer devices, in the past decades, the fourth industrial revolution (known as Industry 4.0) concept was introduced and came into the public eye. Schwab wrote that "like the revolutions that preceded it, the Fourth Industrial Revolution has the potential to raise global income levels and improve the quality of life for populations around the world." [8]. In this new concept, one of the key contents is digitalization and automation of factories and industries to be more efficient in production. Factories with automation that demand high link throughput are also turning to Wi-Fi technology. The WALN deployed in factories requested rather critical performance in terms of both transmission time and reliability [9]. However, Wi-Fi interference significantly impacts network performance and reliability.

In practical use, theoretical link throughput rates are rarely achieved. the Wi-Fi is already performing at a lower throughput than the rated or designed parameter. Meanwhile, there are many limitation factors to the performance, such as interference, device radio limitations, communication protocols, and other reasons. In addition, the previously mentioned factors can further reduce the throughput performance. From these factors, interference reduces the

throughput performance in radio communication. The throughput, often related to latency, is also a key parameter that measures the experience of using a Wi-Fi network. Throughput is a key performance factor of the Internet when paying for the services from the Internet Service Provider (ISP).

When two or more sources are sharing the same frequency band, simultaneous transmissions (i.e., collisions) can cause interference in wireless communications. Stations need to wait for the channel to become idle before transmission. However, multiple stations can wait for the same channel simultaneously, typically leading to collisions [3]. Usually, there are two types of interference [10]:

- Co-channel interference: undesired transmissions cause this type of interference on the same frequency channel.
- Adjacent channel interference: the transmissions on adjacent or partially overlapped channels cause this type of interference.

Due to the extensive spectrum sharing and the resulting 802.11 impairments such as networks without SSID, invisible networks, overlapping channel interference, etc. End-users can experience significant performance degradation in densely populated urban areas [11]. Because of the interference, the throughput performance can be reduced [10], [11], [12], [13] and [14]. Thus, it is beneficial to understand different types of interference and to understand to what extent the performance is reduced.

By carefully selecting the Wi-Fi Access Points (router) location, Managed networks can reduce interference. Managed networks are available for enterprise business users but are expensive. Private routers are usually deployed randomly without channel and power coordination. Thus, using central servers for coordination outside enterprise environments is difficult. It is not easy to avoid interference without central coordination, due to the complexity of the neighbouring interference from the private environment, this will lead to the reconfiguration of networks. The ISP chooses one particular AP (router) brand, the software suite is tailored to this particular AP. The users who live in multi-dwelling units (flats or apartments) suffer not only from co-channel interference but also from adjacent channel interference from their neighbours.

This project was initiated by a company called Maseng AS, which is developing a software suite for Wi-Fi access points (APs). Maseng AS is trying to address the interference between Wi-Fi signals, its system is called Empathetic Radio and is based upon a completely distributed peer-to-peer architecture. The control channels consist of encrypted wired backhaul connections between access points (APs). This APs exchange reports from neighboring devices to form clusters. They then identify which clusters cause the least interference with other networks and proceed to allocate channels and power accordingly. The goal is to achieve all operations automatically without operator or customer intervention. We redesigned this master project to cover part of the mentioned goals. We believe that understanding and quantifying the interference's impact on performance levels is essential before developing any algorithms for different types of interference.

Thus, this master thesis focuses on measuring the effect on the performance of the co-channel interference (CCI), and adjacent channel interference (ACI) to the operational channels. This will be done by using two networks, each equipped with one router and one client. Later by observing the existing results

from the CCI and the ACI, I decided to also measure the interference between the overlapping channels, the interfered router operates at channel 6.

Questions to be studied in this thesis

In order to know what kind of algorithms for channel and power allocation to be developed, we must first find out the following questions:

- Which channel imposes the most interference on the operating channels?
- How much performance reduction is caused by interference?
- Can we consider the interference significant?

CHAPTER 2

Wi-fi technology and interference

Among the various factors that limit Wi-Fi performance, signal interference is one of the most prevalent issues. It negatively affects the throughput of the radio system during communication. Wi-Fi is one of the commonly used ways for internet connections, I want to talk about the relevant technology in this chapter.

2.1 IEEE 802.11 standard and channels

IEEE 802.11 is part of the IEEE 802 set of LAN technical standards. It specifies the medium access control (MAC) and physical layer (PHY) protocols for WLAN communication. These standards form the basis for Wi-Fi products and are the most widely used wireless networking standards globally. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand and are the world's most commonly used wireless computer networking standards. The IEEE 802.11 standards for non-overlapping Wi-Fi channels are in figure 2.1 [15].

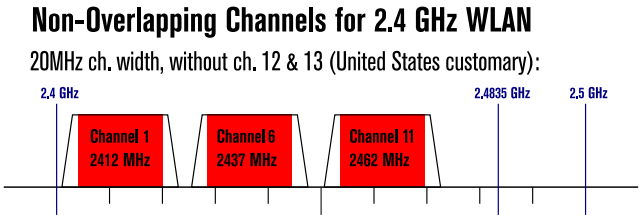


Figure 2.1: 2.4G Wi-Fi non-overlapping channel distribution

2.2 Wi-Fi Interference

Wi-Fi networks operate in the unlicensed Industrial, Scientific, and Medical (ISM) bands, primarily at 2.4 GHz, 5 GHz and 6 GHz. These bands are shared with many other devices, leading to potential interference. Common sources of interference include:

- Other Wi-Fi networks: overlapping channels in densely populated areas can cause significant interference [6]- [12].

- Non-Wi-Fi devices such as microwave ovens, cordless phones, and Bluetooth devices also operate in the ISM bands and can interfere with Wi-Fi signals [12]. This project does not focus on this type of interference.
- ZigBee networks: share the same frequency bands as Wi-Fi. IoT devices often use the ZigBee communication protocol, these devices can cause potential interference. [16].

Impact on Network Performance

Interference can significantly degrade network performance by reducing data throughput, increasing latency, and causing packet loss. Studies [10], [11], [12], [13] and [14] have shown that interference can substantially decrease coverage and capacity, particularly in high-density environments like urban areas and office buildings [17].

Interference between radio signals

Co-channel interference (CCI) and adjacent channel interference (ACI) are common types of interference. The 802.11 standard uses Carrier Sense Multiple Access (CSMA), which follows a listen-before-talk protocol. A station can only transmit if the medium is idle. To elaborate, a station attempts to identify whether a carrier signal from a different station is present prior to initiating its own transmission. If a carrier is detected, the station will pause until the ongoing transmission concludes before starting its own. CCI in most cases will cause the carrier sensing mechanism to report that the medium is busy, leading to performance reduction. In some cases, ACI may behave similarly as CCI, leading to performance reduction [12].

Mitigation Strategies

Some strategies can be used to mitigate Wi-Fi interference. These include:

- Channel Selection and Management: By dynamically choosing the channels with the least congestion from both traffic and devices, you can help reduce interference. Tools such as Wi-Fi analyzers can aid in finding the best channels.
- Power Control: Adjusting the transmission power of Wi-Fi devices can minimize interference by reducing the overlap of coverage areas.
- Advanced Technologies: The adoption of technologies such as Orthogonal Frequency-Division Multiplexing (OFDM) and Multiple Input Multiple Output (MIMO) can enhance signal robustness and reduce susceptibility to interference [6].
- Regulatory measures [18]: Implementing rules to manage how frequencies are allocated and used can help reduce interference. In other words, specific networks should establish guidelines for measurements and assign different frequencies to various types of devices in specific locations to prevent frequency overlap.

- When designing wireless multichannel mesh networks, placing APs at larger distances can make interference between non-overlapping channels negligible [19].

2.3 Related work

A large body of research has focused on accurately characterizing spectrum utilization in ISM bands. For example, in one study on the IEEE 802.11n standard [12], researchers found that the distance between nearby transmitters restricts concurrent transmissions and reduces spatial reuse. A transceiver can corrupt incoming signals with its outgoing signal if the latter is too strong.

In another work [20], the authors worked on understanding the impact of utilizing larger channel width on energy efficiency and interference. They found that, for larger channel widths at 80 MHz channel width, the higher idle mode power consumption yields substantial throughput improvement. But, the improvements come at the cost of higher power consumption. They also found that increasing the number of independent and separately coded data signals is more energy efficient than expanding the channel width to achieve the same percentage increase in throughput. One of the findings is that the unplanned selection of primary channels and channel widths can severely degrade the throughput of links operating at larger channel widths.

In another study [13], the authors emphasized that collecting accurate and comprehensive statistics at both the MAC and physical layers is essential for troubleshooting issues. They noted that wireless experiments can sometimes be more deceptive than well-planned simulations. Achieving reproducibility can be challenging, and failing to consistently identify the factors influencing a testbed's behavior may result in incorrect conclusions. Therefore, it's important to begin with small, well-calibrated experiments before progressing to more complex and intriguing scenarios.

2.4 What is Ookla

We chose Ookla®, a speed test tool for testing the throughput for this project, it was founded in 2006 and takes 11+ million tests daily [21].

A test taken with Speedtest measures the characteristics of the communication network between a device and multiple servers using the Speedtest global server network. The fewer links between a device and a server, the more relevant the measurement is to quantifying and understanding the networking capability of a particular device. To that end, Ookla operates a vast testing infrastructure of over 15,000 servers worldwide which allows us to ensure an accurate, meaningful view of network performance.

When aggregated, these measurements describe the network's real-world performance, including information about locations, times, service providers, and devices. For the quality of service metrics Ookla collects, please see the Metrics and Definitions of this Ookla guide [22].

How does Speedtest work

This section introduces how the speed test and its components are determined [21].

Ping measures the round trip time for messages to travel from the requester to a server.

1. This test is performed by measuring the time it takes for the server to reply to a request from the user's client. The client sends a message to the server. Upon receiving that message, the server will send a reply. The round-trip time is measured in ms (milliseconds) 2.1.

$$ping = t_{reply} - t_{request} \tag{2.1}$$

   Where t is the time stamp of corresponding data.

2. This test is repeated multiple times with the lowest value determining the final result.

The download speed measures how quickly one can pull data from a server on the internet to the calling device. Most connections are designed to download much faster than they upload. This is because the majority of online activity, like loading web pages or streaming videos, is driven by downloading content. Download speed is measured in megabits per second (Mbps).

1. The client establishes multiple connections with the server over the default port: 8080 via Transmission Control Protocol (TCP) connection. The client requests the server to send an initial chunk of data U_0. This data amount is determined by the Ookla implementation and a combination of other network module tools, so I did not find relevant numbers used here. I assume the tool is well-structured and reliable for the purpose of this thesis.

2. The client calculates the real-time speed of the transfers and then adjusts the chunk size and buffer size based on this calculation to maximize the usage of the network connection.

3. As the chunks are received by the client, the client will request more chunks throughout the test.

4. During the first half of the test, the client will establish extra connections to the server if it determines additional threads are required to measure the download speed more accurately.

5. The test ends once the configured amount of time T has been reached.

$$speed_d = \frac{1}{T} \sum_{i=0}^{n} D_i \tag{2.2}$$

   Whereas, T is the entire duration of the test. D is the downloading data size and n is the times of requests.

The upload speed measures how quickly one sends data from a client device to the internet. A fast upload speed is helpful when sending large files via email, or using video chat to talk to someone else online (since you have to send your video feed to them). Upload speed is measured in megabits per second (Mbps).

1. The client establishes multiple connections with the server over port: 8080 and sends an initial chunk of data D_0.
2. The client calculates the real-time speed of the transfers. It then adjusts the chunk size and buffer size to maximize usage of the network connection and requests more data.
3. As the server receives the chunks, the client will send more chunks throughout the test.
4. During the first half of the test, the client will establish extra connections to the server if it determines additional threads are required to measure the upload speed.
5. The test ends once the configured amount of time T has been reached.

$$speed_u = \frac{1}{T}\sum_{i=0}^{n} U_i \tag{2.3}$$

Whereas, T is the entire duration of the test. U is the downloading data size and n is the times of requests.

The Speedtest API

The Speedtest API [23] sends certain amounts of data chunks to https://www.speedtest.net/, then the API calculates the ping, download and upload speed and ping using equations 2.1, 2.2 and 2.3 respectively. In addition to the API, I modified the scripts to save the results to my needs. This part is discussed in detail in chapter 3, section Ookla implementation 3.3.

CHAPTER 3

Measurement setup and methods

This chapter discusses the setup and methods used for the project, including the device list, setup illustration, Ookla implementation, test automation, and result plots.

When designing the measurements, we think that there will always be Wi-Fi interference in the environment. So, we believe it is unnecessary to use a Faraday room to exclude the environmental interference, another reason is that getting a Faraday room is too expensive both in time and expense. Meanwhile, the Faraday room might not have a needed internet connection. Thus, the project is based on the assumption that the environmental interference is "consistent" in certain periods of the day and "insignificant".

3.1 Device list

The setup includes 2 routers, 2 Raspberry PIs and one Faraday cage. The Faraday cage was first constructed in 1836 by scientist Michael Faraday, the device is named after him. The Faraday cage is a metal net-enclosed device that can block out electromagnetic waves. One router acts as the interfering source (shielded as in the following texts) on the set channel, while the other router (unshielded as in the following texts) receives the interference from the interfering source on different channels at 0/20/40 dB attenuation levels. Theoretically, the higher the attenuation levels the less the interference the 2 routers will impose on each other.

All measurements are conducted at 2.4 GHz with a 20 MHz bandwidth between channels. The 2 Raspberry PIs are connected to their corresponding routers via Wi-Fi. One of the Raspberry PIs (shielded as in the following texts) measures the performance of the interfered device group. The other PI (unshielded as in the following texts) measures the interfering device group. The results are expected to be 2 sets of throughput datasets with a measurement count of 50 (± 3). The results include NI, CCI and ACI datasets. The interfering device group is placed inside the Faraday cage with an attenuator, which controls the attenuation levels. The attenuation level can be an alternative to simulate the distance and different obstacles between the 2 device groups with their corresponding Wi-Fi networks. Below is the list of devices:

- 2 Heimgard Smart Mesh [24] routers as the APs. The Heimgard Smart Mesh routers can provide up to 900 Mbps throughput in an interference-free environment. However, in this project, the throughput is limited by

the Raspberry PI network card or the connected network speed limit. The routers operate at 20dBm (100mW) using the 802.11n protocol. Table 3.1 shows the detailed technical parameters.

	unshielded	shielded
Hostname	heimgard-7FF0	heimgard-7800
Model	M1DM-AHAJ-G0A0ACKW0-HOv1	
Architecture	ARMv7 Processor rev 4 (v7l)	
Target Platform	airoha/en7523	
Firmware Version	HeimgardOS 23.05.2 7.3.1.17.2.0	
Kernel Version	05.04.1955	

Table 3.1: The router technical parameters

The current routers have problems changing channels for 5 GHz frequency, thus tests and measurements at 5 GHz are not performed.

- 2 raspberry PI 3B+ [25] as clients (measurement devices). The PIs come clean of any OS, they are later installed with Debian Bullseye with Raspberry Pi Desktop [26], table 3.2 shows the technical details:

Release date	July 1st 2022
System	32-bit
Kernel version	5.10
Debian version	11 (bullseye)

Table 3.2: The Raspberry PI 3B+ technical parameters

- One Faraday cage as in figure 3.1a, the Faraday case is modified to fit the experiments. One side is connected to a shielded CAT 6 cable; the other side has an antenna connector connected to the attenuator from the inside and an antenna on the outside.

- An adjustable attenuator as in figure 3.1b, which ranges from 0-49 dB. Two antennas, one connected to the attenuator on the outside of the Faraday case, and one connected to the other side of the attenuator inside the Faraday cage.

(a) Device setup picture.

(b) The attenuator.

- Normal CAT6 Ethernet cables and one shielded CAT cable for router's connections.

3.2 Measurements setup illustration

The measurement setup is illustrated in figure 3.2.

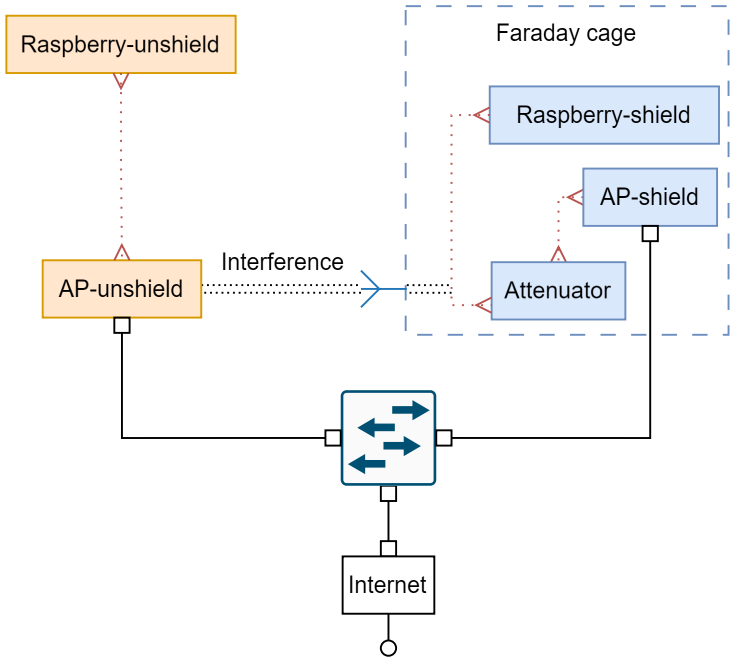


Figure 3.2: Device setup illustration

In this setup, there are 2 APs: one shielded (router-shielded) and one unshielded (router-unshielded). The router-shielded is placed inside a Faraday cage and connected to a switch via a wired connection, while the router-unshielded is connected to the same switch but to a different port.

On the other hand, the router-unshielded is connected to the same switch as the router-shielded in the same manner, but to a different port. Its corresponding client Raspberry-unshielded is connected directly via a Wi-Fi connection.

The shielded Wi-Fi network can impose interference to the unshielded Wi-Fi network, the attenuation levels can be controlled in this setup, and vice versa. The actual placement of devices is shown in figure 3.3.

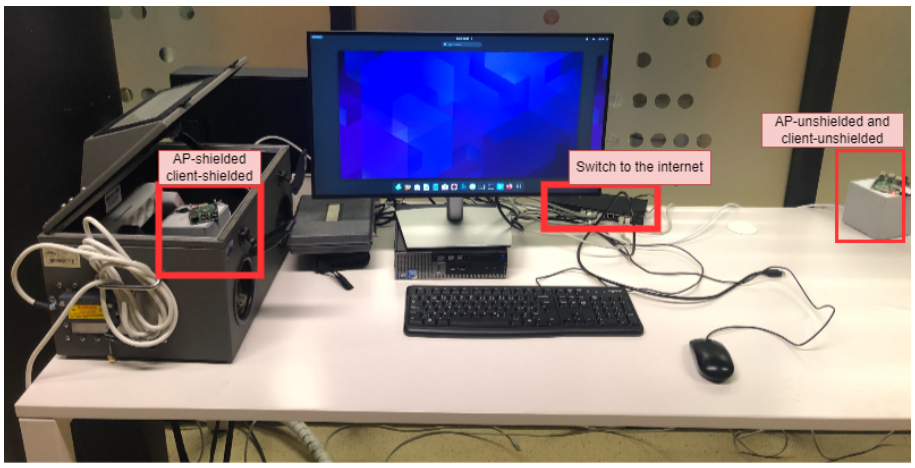


Figure 3.3: Device setup

The 2 routers are put away from each other with a distance of around 1 meter. In between is the network switch. Different attenuation levels can imitate real-life interference levels, either caused by distance or obstacles. In practice, the higher the attenuation levels the less interference the shielded network will send out and receive, the similar situation applies to the unshielded network. This can translate to distance positively correlates to attenuation levels.

For the measurements, I have datasets for non-interference (NI), Co-channel interference (CCI), and adjacent channel interference (ACI). The datasets group should look like the below table 3.3.

shielded / unshielded	off	ch1	ch6	ch11
off	n/a	ni	ni	ni
ch1	ni	cci	aci	n/a
ch6	ni	n/a	cci	aci
ch11	ni	n/a	aci	cci

Table 3.3: The non-overlapping channel measurement matrix

The measurement data matrix includes datasets for NI, CCI and ACI. The shielded group acts as the interference source, while the unshielded group is the

control. The tests are repeated at 0 dB, 20 dB, and 40 dB attenuation levels, resulting in 9 unshielded and 9 shielded results lists for each attenuation level for each attenuation level. The above translates to shielded channel 1 CCI to unshielded, shielded channel 6 ACI to channel 1 and n/a means not applicable. The NI means either the router-shielded or the router-unshielded is turned off and these tests are used for reference to normalize the CCI and ACI results.

The same tests as the table 3.3 are repeated for attenuation levels at 0 dB, 20 dB and 40 dB. Therefore, the test results will include 9 unshielded and 9 shielded results lists.

During the test for non-overlapping setup, I noticed that there is a trend of performance from the best to the worst in the order of NI, ACI and CCI. So I tried to set up overlapping interference to endorse the observed patterns from the non-overlapping setup and measurement. The measurement matrix is as the following table 3.4.

Unshielded \ Shielded	ch1	ch2	ch3	ch4	ch5	ch6	ch7	ch8	ch9	ch10	ch11
ch6	ovl1	ovl2	ovl3	ovl4	ovl5	ovl6	ovl7	ovl8	ovl9	ovl10	ovl11

Table 3.4: The overlapping channel measurement matrix

The measurements are done for only 0 dB attenuation level for reference, so 20 dB and 40 dB attenuation levels are not measured. The reason to measure only the shielded network on channel 6 is that this channel overlaps most channels through the 2.4 GHz Wi-Fi bands. In this setup, the shielded group is fixed at channel 6, and the unshielded group is changing channels from channel 1 to channel 11. Thus, ovl1 means interference from unshielded group channel 1 to the shielded group at channel 6, the same logic applies to other "ovl*" results.

3.3 Ookla and speedtest implementation

In this project, Ookla [21] was recommended as the speed test tool. The tool includes several platform versions, I modified the scripts, as shown in the appendix B.4, from the open source API [23] to perform multiple tests. Measurement steps are set as follows:

1. Adjust device, and change channels according to table 3.3 for the unshielded router and the shielded router.

2. Connect to https://www.speedtest.net via API [23]. The Ookla server chooses the best server available. By the best, it means the lowest PING (equation 2.1) and available. This function is invoked as in the script shown in speedtest.py (appendix B.4)

```
1000    speedtest.get_best_server()
        ...
1002    try:
            fastest = sorted(results.keys())[0]
1004    ...
        best = results[fastest]
1006    best['latency'] = fastest
        ...
```

```
1008        return best
```

3. Store the measurement results.
4. Adjust the attenuation levels for the Faraday cage for the router-shielded.
5. Repeat steps 1 to 4 for different channels and attenuation levels.
6. Analyze and plot the results.

Measurements and automation

This section describes the measurement setup and automation process, including the implementation of speed tests, channel adjustments, and data analysis. The measurements include many repetitions, so I designed scripts to automate the process. Automated measurements help avoid operational interference. In all measurements, 0 dB means that the shielded devices are placed outside the Faraday cage, under the same conditions as the unshielded devices.

I studied and modified speedtest.py (appendix B.4), the speed tests are programmed according to it. The original scripts are cloned from the open-source speed test API [23]. Below is how I implement the speedtestCli:

1. In speedtest.py (appendix B.4), I modified the scripts to save the results to Excel files. Results include download, upload speed, PING, server info, time stamps and other relevant data, but I am only interested in the download and upload speed, the time stamps and PING for the purpose of this project. The modification is made at script line 2003 and similar lines. The utilities.py B.3 helps to sort and store the results:

```
1000        utilities.convert_and_save_to_xlsx(str(results), "
    results/ch1-ni-shield.xlsx")
```

2. I then change the channels at each router from different attenuation levels and confirm the clients are connected to the correct channels after the channels are changed. The correct channel is confirmed from table 3.5. I ran the iw [27] command:

```
1000        iw dev wlan0 link
```

As a result, table 3.5 shows the connection information at different attenuation levels. These results show that the theoretical transmitter and receiver max speed is the same at 72.2 MBit/s.

channel 1	unshielded	shielded
SSID	Heimgard_7FF0_2	Heimgard_7800_2
freq	2412	
signal	-4 dBm	-5 dBm
rx bitrate tx bitrate	72.2 MBit/s	

channel 6	unshielded	shielded
SSID	Heimgard_7FF0_2	Heimgard_7800_2
freq	2437	
signal	-4 dBm	-8 dBm
rx bitrate tx bitrate	72.2 MBit/s	

channel 11	unshielded	shielded
SSID	Heimgard_7FF0_2	Heimgard_7800_2
freq	2462	
signal	-4 dBm	-6 dBm
rx bitrate tx bitrate	72.2 MBit/s	

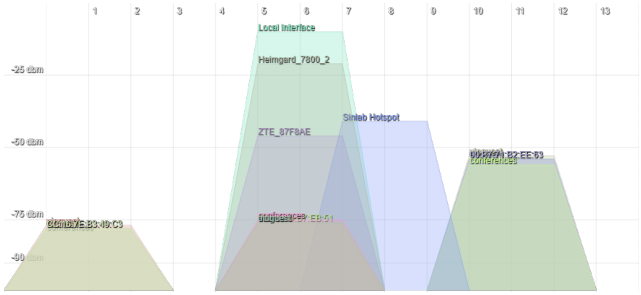
Table 3.5: Channel 1, 6 and 11 connection information from the clients

3. Meantime, I measure the signal strength by changing the attenuator to different attenuation levels. Table 3.6 shows that by closing the Faraday cage, the signal level will be suppressed a great deal. So, in my measurement, to get good enough interference from the shielded device group, I put the devices outside the Faraday cage for all 0 dB attenuation level measurements according to table 3.3.

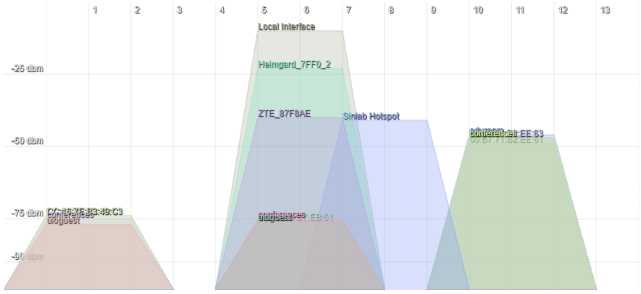
	0 dB lid open	0 dB lid closed	20 dB	40 dB
SSID	Heimgard_7800_2			
freq	2412			
signal	-5 dBm	-55 dBm	-64 dBm	-67 dBm
rx bitrate tx bitrate	72.2 MBit/s			

Table 3.6: Signal strength at different attenuation levels for the shielded router

4. I analyzed the environmental interference from the router-embedded tools at different attenuation levels. The results show the interference level. Figure 3.4 and table 3.7 show the environment interference signal power levels when the shielded device group is set at 0 dB. In the figure, the x-axis is on the top showing channels, and the y-axis is on the left showing the signal strength.



(a) Unshielded



(b) Shielded

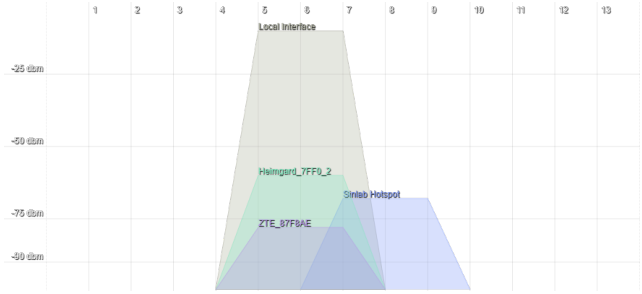
Figure 3.4: Environment interference for different device groups when the attenuation level is set to 0 dB

0dB: interference to the device groups with a bandwidth of 20 MHz					
Unshielded			Shielded		
Signal	SSID	Channel	Signal	SSID	Channel
-10 dBm	Local Interface	6	-10 dBm	Local Interface	6
-21 dBm	**Heimgard_7800_2**	6	-23 dBm	**Heimgard_7FF0_2**	6
-41 dBm	Sinlab Hotspot	8	-40 dBm	ZTE_87F8AE	6
-46 dBm	ZTE_87F8AE	6	-41 dBm	Sinlab Hotspot	8
-53 dBm	uioguest	11	-46 dBm	eduroam	11
-54 dBm	eduroam	11	-47 dBm	uioguest	11
-54 dBm	hidden	11	-47 dBm	hidden	11
-54 dBm	hidden	11	-47 dBm	conferences	11
-56 dBm	conferences	11	-49 dBm	hidden	11
And more					

Table 3.7: Environment interference for different device groups when shielded is set to 0 dB

Figure 3.5 and table 3.8 show the environment interference signal power levels when the shielded device group is set at 20 dB. In the figure, the x-axis is on the top showing channels, and the y-axis is on the left showing the signal strength.

(a) Unshielded



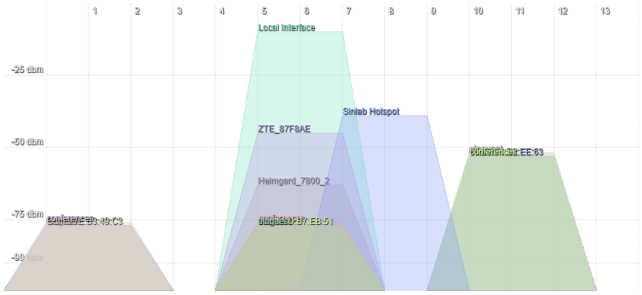
(b) Shielded

Figure 3.5: Environment interference for different device groups when the attenuation level is set to 20dB

20dB: interference to the device groups with a bandwidth of 20 MHz					
Unshielded			Shielded		
Signal	SSID	Channel	Signal	SSID	Channel
-10 dBm	Local Interface	6	-10 dBm	Local Interface	6
-41 dBm	Sinlab Hotspot	8	-58 dBm	**Heimgard_7FF0_2**	6
-44 dBm	ZTE_87F8AE	6	-77 dBm	Sinlab Hotspot	10
-51 dBm	uioguest	11	-77 dBm	ZTE_87F8AE	6
-52 dBm	eduroam	11	-78 dBm	hidden	1
-52 dBm	hidden	11	-78 dBm	hidden	1
-52 dBm	hidden	11	-79 dBm	eduroam	1
-53 dBm	conferences	11	-79 dBm	uioguest	1
-60 dBm	**Heimgard_7800_2**	6	-79 dBm	conferences	1
And more					

Table 3.8: Environment interference for different device groups when the attenuation level is set to 20 dB

Figure 3.6 and table 3.9 show the environment interference signal power levels when the shielded device group is set at 40 dB. In the figure, the x-axis is on the top showing channels, and the y-axis is on the left showing the signal strength.



(a) Unshielded

(b) Shielded

Figure 3.6: Environment interference for different device groups when the attenuation level is set to 40 dB

40dB: interference to the device groups with a bandwidth of 20 MHz					
Unshielded			Shielded		
Signal	SSID	Channel	Signal	SSID	Channel
-10 dBm	Local Interface	6	-10 dBm	Local Interface	6
-63 dBm	**Heimgard_7800_2**	6	-63 dBm	**Heimgard_7FF0_2**	6
-53 dBm	eduroam	11	-72 dBm	Sinlab Hotspot	8
-53 dBm	hidden	11	-73 dBm	ZTE_87F8AE	6
-52 dBm	uioguest	11	-79 dBm	hidden	11
-53 dBm	hidden	11	-79 dBm	uioguest	11
-53 dBm	conferences	11	-79 dBm	conferences	11
-39 dBm	Sinlab Hotspot	8	-79 dBm	eduroam	11
-45 dBm	ZTE_87F8AE	6	-82 dBm	hidden	11
And more					

Table 3.9: Environment interference for different device groups when the attenuation level is set to 40 dB

5. After the confirmations from 2 to 4 for each channel and attenuation level, I started the test. The test includes measuring the upload, download, ping (equation 2.4) and server info. At the same time, I added "–secure" keyword to ensure "https" connection, the reason for this was that I experienced a lot of disconnections and errors due to "http". The exact reasons for this error are unknown, there are no official explanations, I found online discussions and suggestions to add the keyword "–secure", and the program worked afterwards. The terminal invocation:

```
1000 python speedtest.py --secure
```

6. I then use a loop in runtest.py (appendix B.1) to repeat step 5 for 50 ±2 times. The scripts also upload the results to my GitHub [28] to indicate the test completion. The reason for +2 repeats for the shielded device group is to ensure the time coverage of the unshielded group tests. The excessive data pieces were later removed before plotting.

```
script = "speedtest.py"
# the repeat_count can be changed accordingly.
schedule_time = "17:38"
# the repeat_count can be changed accordingly.
repeat_count = 50
while True:
    current_time = datetime.datetime.now().strftime("%H:%M")
    if current_time == schedule_time:
        for i in range(repeat_count):
            subprocess.run(["python", script, "--secure"])
            time.sleep(1)
        git()
        break
    else:
time.sleep(1)
```

7. Step 6 is repeated multiple times for different attenuation levels, the desired results should look like the matrix as shown in table 3.3. Therefore I get 9 results for each attenuation level, and 2 sets of results for both shielded and unshielded, in total there $unshielded_{files} = 9 * 3 = 27 files$, and each file has approximately 50 pieces of measurements that translate to $unshielded_{measurements} = 27 * 50 = 1350$ approximately in total. The same number of measurements apply to shielded as well. So in total, there are about 2700 rows of data.

3.4 Plot the results

After I completed the measurements, I then wrote my scripts in Python(appendix B.2) to plot the data. What is interesting to me is the download and upload from the results, I then extracted the data from the Excel files in the following logic:

1. By using os module, the program walks through the targeted folders and files and adds the files to a list.
2. Then I perform file sorting, in the sequence of ch1, ch6 to ch11.
3. After sorting in certain orders, I transform the download and upload columns into a pandas DataFrame:

```
    df = pd.read_excel(file).rename(columns={"download":dn, "
    upload":up})
```

4. The Pandas DataFrame is transformed into columns of channel-interference, type and channel.
    - Channel-interference: is the column that contains the upload and download throughput values.
    - Type: is the column that contains download or upload as rows. The download and upload correspond to the values in the Excel file.
    - Channel: the column showing the corresponding channel from the Excel file.
5. Plot the DataFrame using Python Seaborn and matplotlib modules.
6. Create a loop to go over steps 1 to 5 for each saved file.

Similar sorting logic, plotting and operations also apply to the overlapping measurement results. The differences are that the overlapping measurements are plotted by channels and in violin plot type.

CHAPTER 4

Results

I set the results into different comparison groups:

- The fixed attenuation levels comparisons: as shown in table 4.1. Later in section 4.1 and appendix A.1, the figures are arranged in order as table 4.1.

Attenuation	Channel comparison		
0 dB	ch1	ch6	ch11
20 dB	ch1	ch6	ch11
40 dB	ch1	ch6	ch11

Table 4.1: Fixed attenuation level comparisons with different channels side by side

In these comparisons, I compare the results of the same device group on different channels. The comparisons are supposed to tell how different channels behave. These comparisons apply to download and upload datasets for unshielded and shielded device groups.

- The fixed channels comparisons: as shown in table 4.2. Later in section 4.2, the figures are arranged in the order of the table 4.2.

Channel	Attenuation comparison		
ch1	0 dB	20 dB	40 dB
ch6	0 dB	20 dB	40 dB
ch11	0 dB	20 dB	40 dB

Table 4.2: Fixed channel comparisons with different channels side by side

In these comparisons, I compare the results of the same channel but from different device groups. The comparisons can tell how strongly the attenuation levels affect the performances. These comparisons apply to download and upload datasets for unshielded and shielded device groups.

- The fixed attenuation levels and channels comparisons: as shown in table 4.3. Later in section 4.3, the figures are arranged in the order of the table 4.3.

Attenuation	Device groups	
0 dB & ch1	unshielded	shielded
0 dB & ch6	unshielded	shielded
0 dB & ch11	unshielded	shielded

Table 4.3: Fixed attenuation and channel comparisons with different device groups side by side

In these comparisons, I compare 2 different device groups at the same attenuation levels and channels in these comparisons. The comparisons tell how different device groups behave. These comparisons apply to both the download and upload datasets for unshielded and shielded device groups.

- The overlapping channels comparisons on the shielded device group: as shown in table 4.4. In these comparisons, I set the shielded APs at the fixed channel 6 and kept change channels on the unshielded device group. These comparisons can tell how overlapping can affect the performances. This comparison is only set for the 0 dB attenuation level.

The interfered group	Interference source
shielded channel 6	unshielded from channels 1 to 11

Table 4.4: Overlapping interference measurement comparisons for the shielded device group

- The overlapping channels comparisons on different device groups: as shown in table 4.5. In these comparisons, the results look into the unshielded device group, the interference source is the shielded device group that operates at channel 6. The comparisons can tell how the channels perform from an interference source at the fixed channel 6. This comparison is only set for the 0 dB attenuation level.

The interfered group	Interference source
unshielded from channels 1 to 11	shielded at ch6

Table 4.5: Overlapping interference measurement comparisons for the unshielded device group

All of the above comparisons include both upload and download plots. The legend of each plot shows the standard deviation of the corresponding data group. The standard deviation (Std) is a parameter that indicates the performance consistency of one device group. In my case the lower the standard deviation, the better the network stability between the clients and the routers. The figures have 2 axes, the download data, as the horizontal axis. Counts that show each data point measured and the Speed (Mbps), the vertical axis that corresponds to the points on the horizontal axis.

In Robinson's work [13], he mentioned that wireless experimentation can sometimes be misleading and is difficult to reproduce the same results. By my understanding, the same results mean the same patterns and the same

measurement numbers for the same measurement setups. Thus, my results of different measurements at different periods varied, but I assumed the environmental interference levels were the same during the same period of the day, so I combined different results from similar measurement periods on different days (after 18:00 each day or whole days in weekends) and concluded the results and plots based on the best and stable measurements.

4.1 Cross channels at the same attenuation level comparisons

This section discusses the cross-channel interference at the same attenuation for ACI, CCI with reference from non-interference (NI), results include all relevant plots showing both upload and download for different attenuation levels from the unshielded and shielded device groups. The unshielded device group is the controlled group, the shielded group is seen as the interference source that can change attenuation levels. Results are shown in plots in this section. All results for the shielded group are shown in appendix A.1.

Cross channels interference at 0 dB

This subsection presents the results for cross-channel interference at the 0 dB attenuation level, results from channels 1, 6 and 11 are plotted and compared side by side according to table 4.6.

Results from the unshielded network at 0 dB attenuation

Figures 4.1 and 4.2 show both the download and upload throughput performance results, we can see that the CCI is greater than ACI and NI. There is no clear pattern of performance between different channels. The unstable performance from different channels of CCI and ACI is probably due to environmental interference as shown in figure 3.4 and table 4.6.

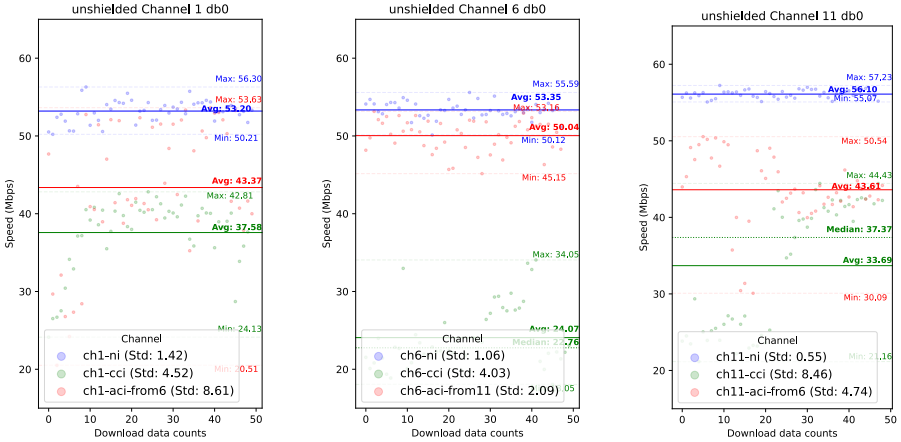


Figure 4.1: Unshielded network, cross channel interference at 0 dB attenuation, download comparison. Std is short for standard deviation

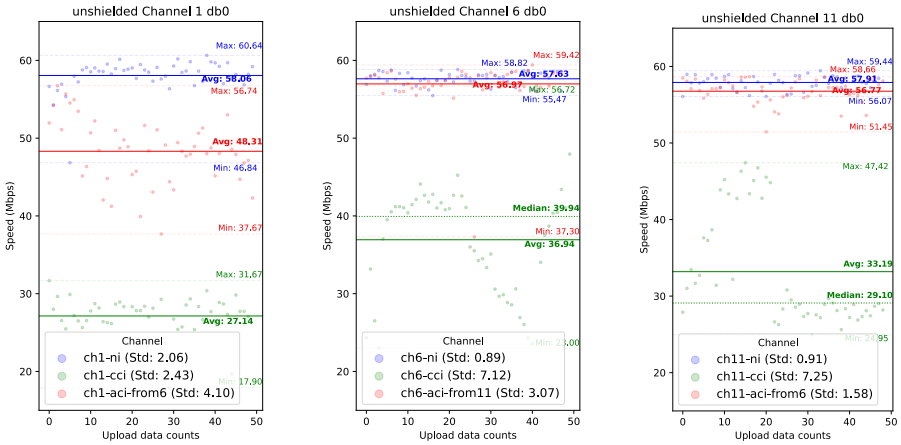


Figure 4.2: Unshielded network, cross channel interference at 0 dB attenuation, upload comparison. Std is short for standard deviation

Cross channels interference at 20 dB

This subsection presents the results for cross-channel interference at the 0 dB attenuation level, results from channels 1, 6 and 11 are plotted and compared side by side according to table 4.7.

Results from the unshielded network at 20 dB attenuation

Figures 4.3 and 4.4 show the download throughput results, the results show that the CCI is greater than ACI and NI. The results seem to show a trend of better performance with higher centre frequencies. The more stable results can be explained because the biggest interference from the shielded group is significantly reduced as shown in figure 3.5 and table 4.7.

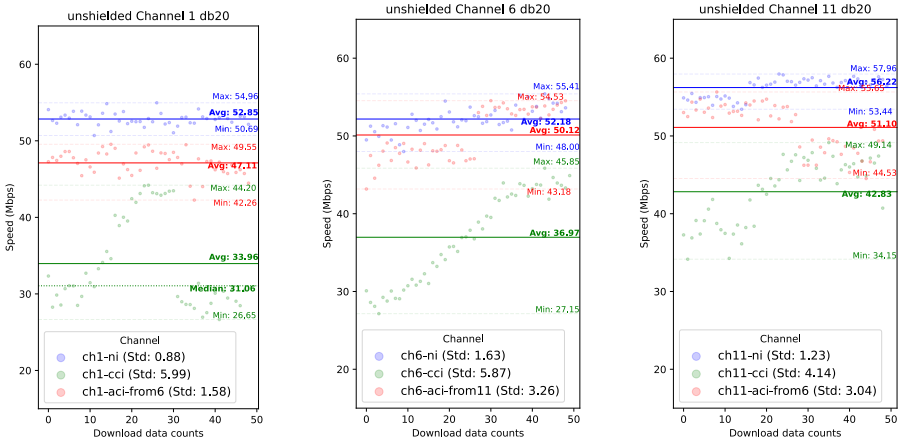


Figure 4.3: Unshielded network, cross channel interference at 20 dB attenuation, download comparison. Std is short for standard deviation.

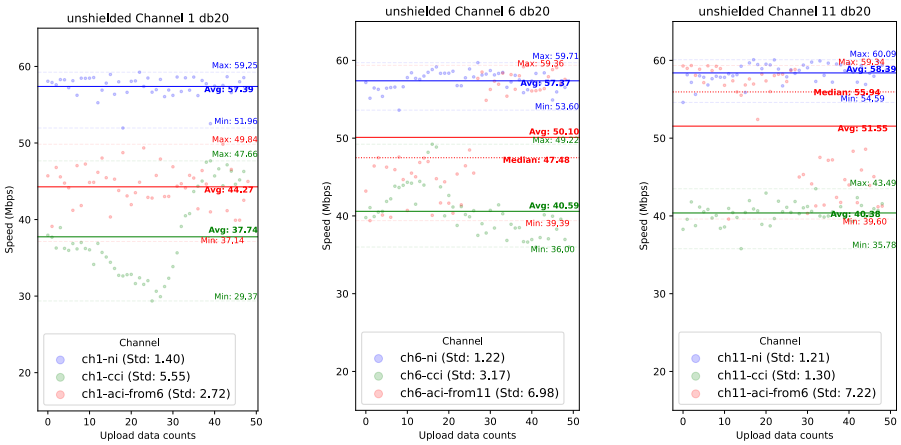


Figure 4.4: Unshielded network, cross channel interference at 20 dB attenuation, upload comparison. Std is short for standard deviation.

Cross channels interference at 40 dB attenuation

This subsection presents the results for cross-channel interference at the 0 dB attenuation level, results from channels 1, 6 and 11 are plotted and compared side by side according to table 4.8.

Results from the unshielded network at 40 dB attenuation

Figures 4.5 and 4.6 show the download and upload throughput results, the results show that the CCI is greater than ACI and NI. The results show a trend of better performance with higher centre frequencies. The more stable results can be explained because the biggest interference from the shielded group is significantly reduced as shown in figure 3.6 and table 4.8. However, there is an outlier for figure unshielded channel 11 at 40 dB. This can be a result of device performance and spatial streams occupied by the other signals from the environment.

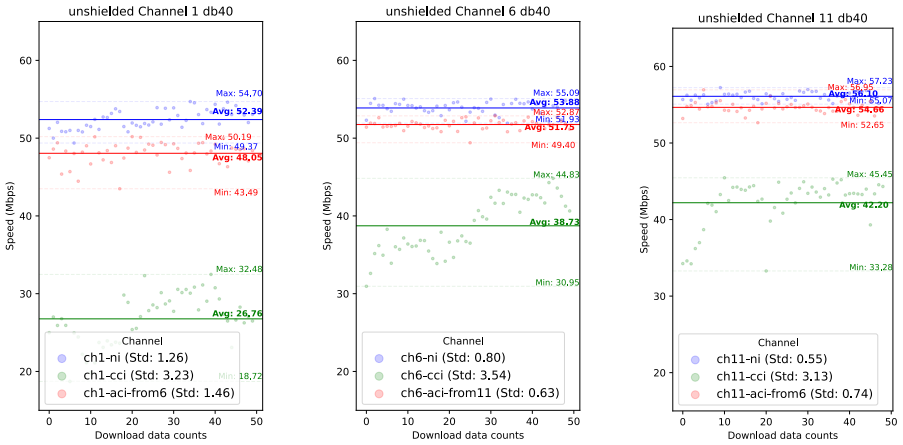


Figure 4.5: Unshielded network, cross channel interference at 40 dB attenuation, download comparison. Std is short for standard deviation

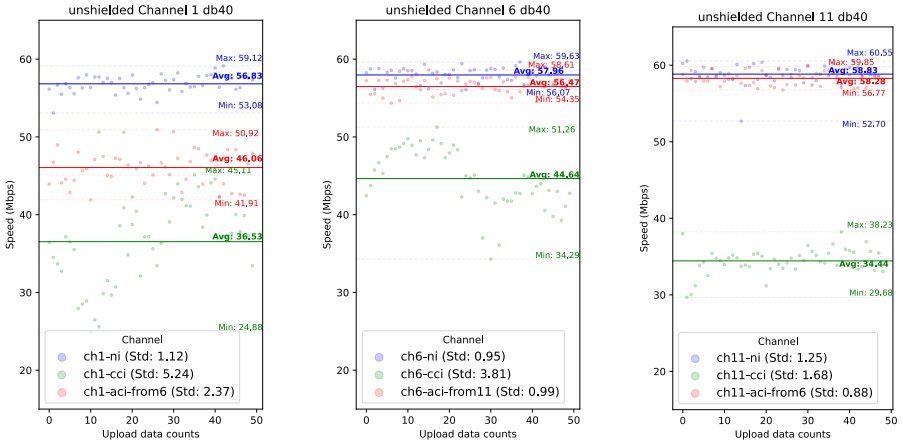


Figure 4.6: Unshielded network, cross channel interference at 40 dB attenuation, upload comparison. Std is short for standard deviation.

4.2 Same channel, cross attenuation levles for ACI, CCI

This section discusses the ACI, CCI with reference from NI for the same channel at different attenuation levels. The results include all relevant plots showing upload and download for different attenuation levels from the unshielded and shielded device groups. The unshielded device group is the control group, the shielded group is seen as the interference source that can change attenuation levels. Results are shown in plots in this section.

Same channel, cross attenuation for channel 1

This subsection presents the results for cross-attenuation level interference at channel 1, results from attenuation levels at 0, 20 and 40 dB are plotted and

compared side by side according to table 4.9.

Results from the unshielded network for channel 1

Figures 4.7 and 4.8 show the download throughput results for channel 1, the results show that the CCI is greater than ACI and NI. The pattern of the CCI going down by increasing the attenuation levels should be reversed. However, we can see that the ACI performs better, but when we combine both the download and upload plots, I say that the patterns are not concluding trends for different attenuation levels, this phenomenon is a normal variation rather than a solid pattern. The cause for this may be due to environmental interference and other unclear factors in the system.

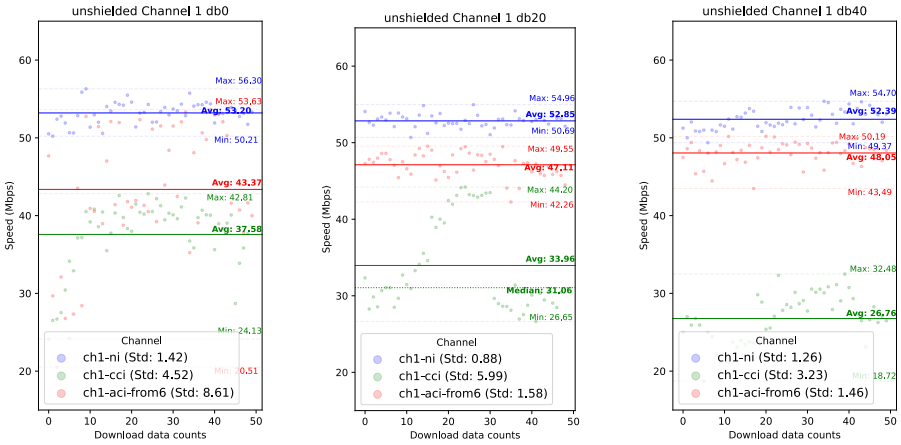


Figure 4.7: Unshielded network, channel 1 interference, cross attenuation, download comparison. Std is short for standard deviation

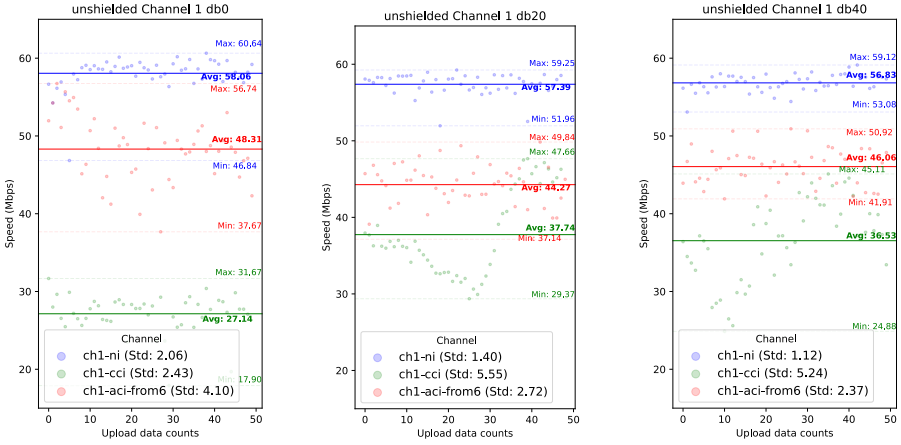


Figure 4.8: Unshielded network, channel 1 interference, cross attenuation, upload comparison. Std is short for standard deviation

Same channel, cross attenuation for channel 6

This subsection presents the results for cross-attenuation level interference at channel 1, results from attenuation levels at 0, 20 and 40 dB are plotted and compared side by side according to table 4.10.

Results from the unshielded network for channel 6

Figures 4.9 and 4.10 show the download and upload throughput results for channel 6, the results show that the CCI is greater than ACI and NI. The trend for this test group is insignificantly good enough to conclude solid patterns that the higher the attenuation levels, the better the overall performance. The only outlier is the unshielded upload from channel 6 at 20 dB.

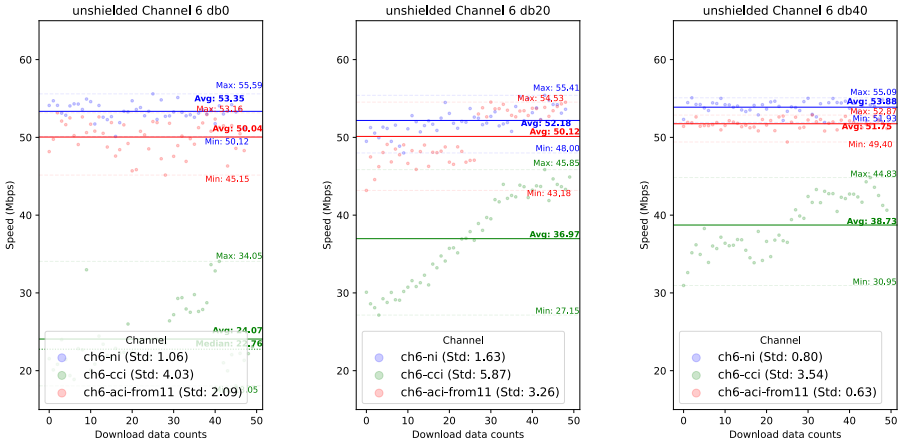


Figure 4.9: Unshielded network, channel 6 interference, cross attenuation, download comparison. Std is short for standard deviation.

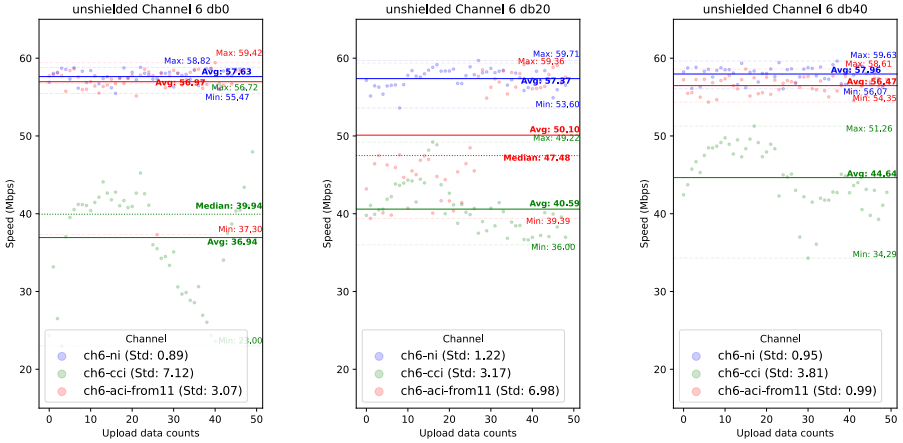


Figure 4.10: Unshielded network, channel 6 interference, cross attenuation, upload comparison. Std is short for standard deviation.

Same channel, cross attenuation for channel 11

This subsection presents the results for cross-attenuation level interference at channel 1, results from attenuation levels at 0, 20 and 40 dB are plotted and compared side by side according to table 4.11.

Results from the unshielded network for channel 11

Figures 4.11 and 4.12 show the download and upload throughput results for channel 11, the results show that the CCI is greater than ACI and NI. The trend for this test group is insignificantly good enough to conclude solid patterns that the higher the attenuation levels, the better the overall performance. The only outlier is the unshielded upload from channel 11 at 40 dB.

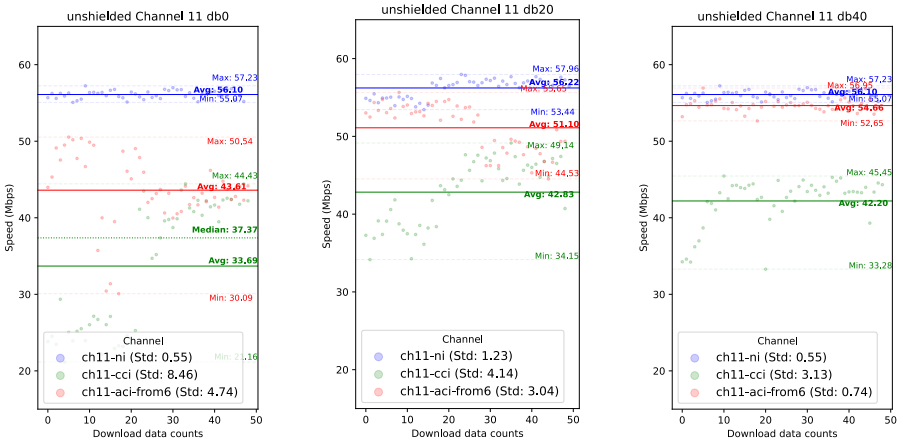


Figure 4.11: Unshielded network, channel 11 interference, cross attenuation, download comparison. Std is short for standard deviation.

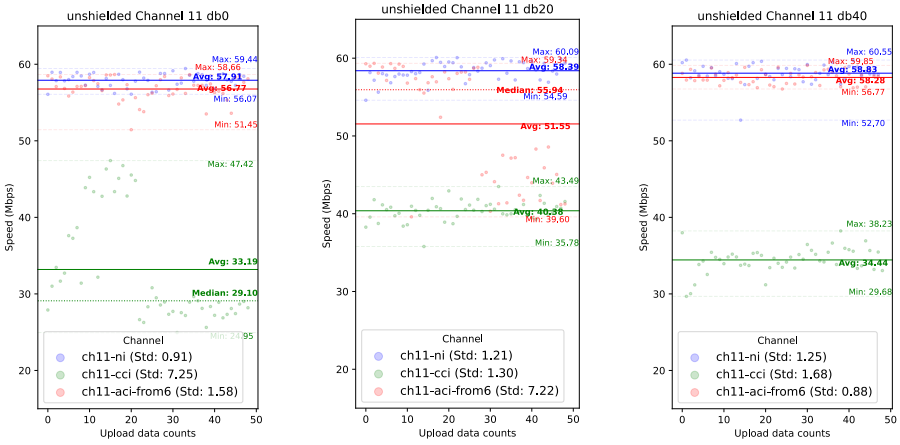


Figure 4.12: Unshielded network, channel 11 interference, cross attenuation, upload comparison. Std is short for standard deviation.

4.3 Cross networks comparison at the identical channel and attenuation

This section discusses the interference from different device groups at the same channel and attenuation levels. That is to say, as shown in table 3.4, when the measurement is at channel 1 and the attenuation level is 0 dB, the comparison is made for the shielded device group and unshielded device group in parallel. The comparisons can also endorse each other for the impact on the throughput performance of the CCI, ACI. Hypothetically, the shielded group should perform better than the unshielded due to the Faraday case filtering of environmental interference.

Results from channel 1 at 0 dB

This subsection presents the results for cross-network performance at channel 1 and 0 dB. Results are plotted and compared side by side. Figure 4.13 and 4.14 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 1 has only 2 datasets for the shielded networks, the NI and ACI.

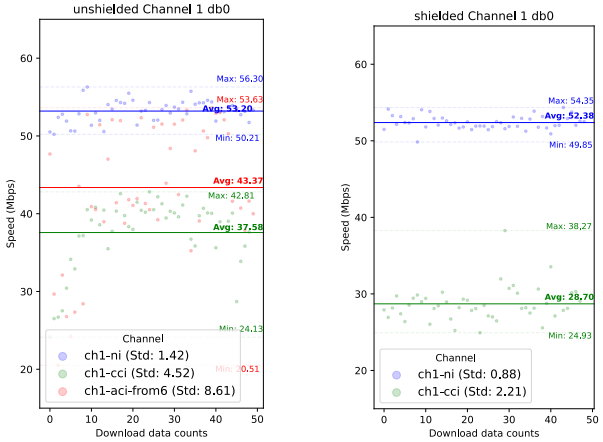


Figure 4.13: Cross networks, channel 1 interference at 0 dB, download comparison. Std is short for standard deviation.

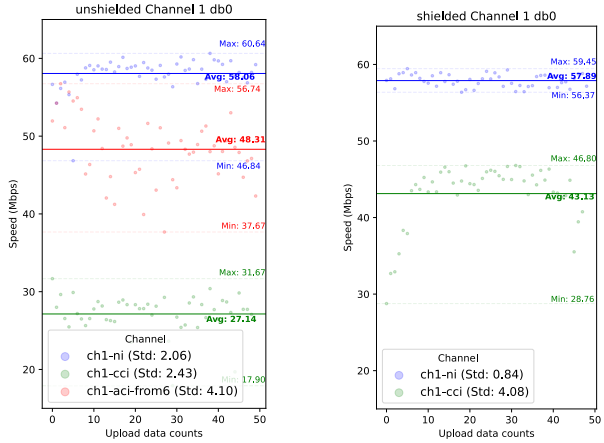


Figure 4.14: Cross networks, channel 1 interference at 0 dB, upload comparison. Std is short for standard deviation.

Results from channel 6 at 0 dB

This subsection presents the results for cross-network performance at channel 6 and 0 dB. Results are plotted and compared side by side. Figure 4.15 and 4.16 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 6 has 4 datasets for the shielded network, the NI, CCI and 2 sets of ACI.

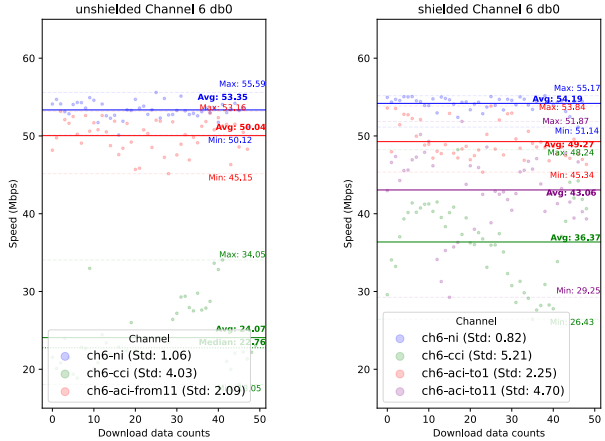


Figure 4.15: Cross networks, channel 6 interference at 0 dB, download comparison. Std is short for standard deviation.

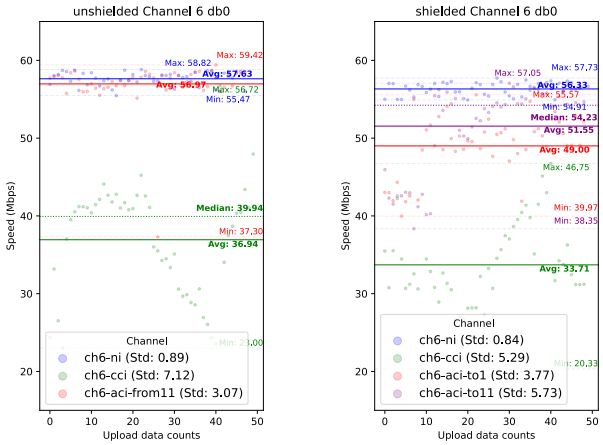


Figure 4.16: Cross networks, channel 6 interference at 0 dB, upload comparison. Std is short for standard deviation.

Results from channel 11 at 0 dB

This subsection presents the results for cross-network performance at channel 11 and 0 dB. Results are plotted and compared side by side. Figure 4.17 and 4.18 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 11 has 3 datasets for both the shielded and unshielded networks, the NI, CCI and ACI.

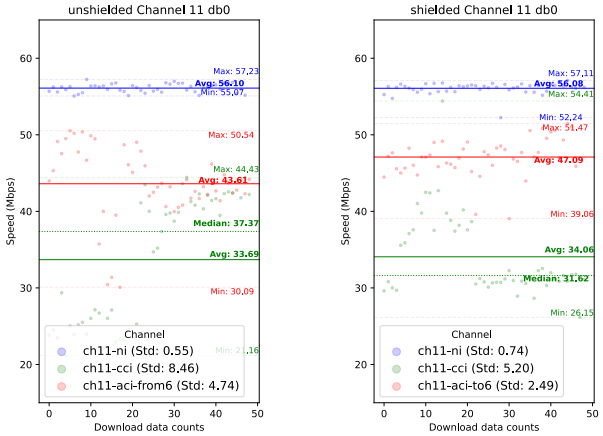


Figure 4.17: Cross networks, channel 11 interference at the 0 dB, download and upload comparison. Std is short for standard deviation.

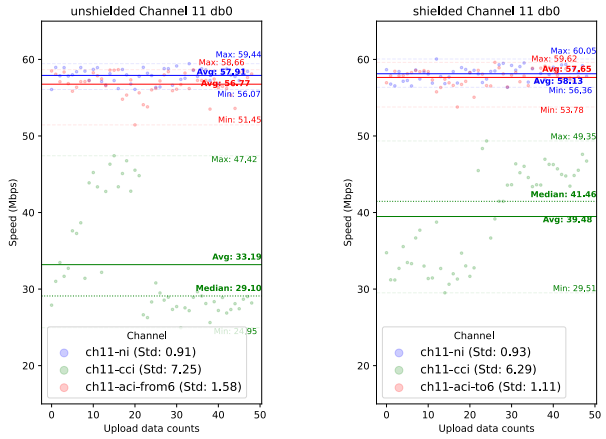


Figure 4.18: Cross networks, channel 11 interference at 0 dB, upload comparison. Std is short for standard deviation.

Results from channel 1 at 20 dB

This subsection presents the results for cross-network performance at channel 1 and 20 dB. Results are plotted and compared side by side. Figure 4.19 and 4.20 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 1 has only 2 datasets for the shielded network, the NI and ACI.

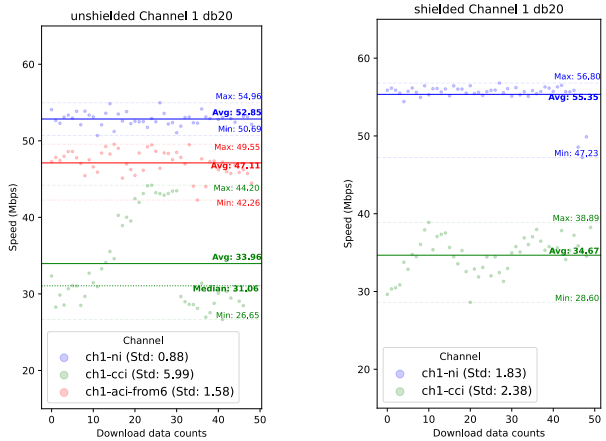


Figure 4.19: Cross networks, channel 1 interference at the same attenuation, download comparison. Std is short for standard deviation.

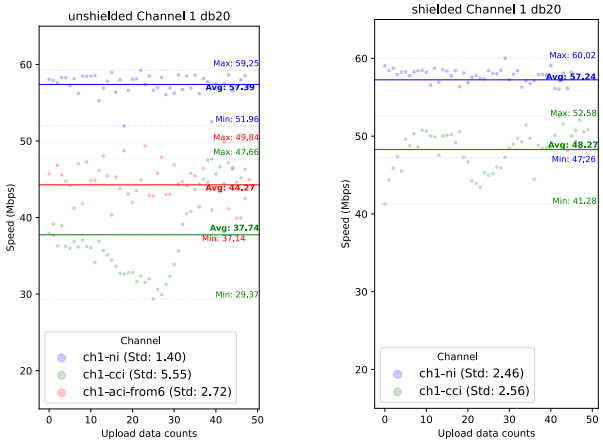


Figure 4.20: Cross networks, channel 1 interference at the same attenuation, upload comparison. Std is short for standard deviation.

Results from channel 6 at 20 dB

This subsection presents the results for cross-network performance at channel 6 and 20 dB. Results are plotted and compared side by side. Figure 4.21 and 4.22 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 6 has 4 datasets for the shielded network, the NI, CCI and 2 sets of ACI.

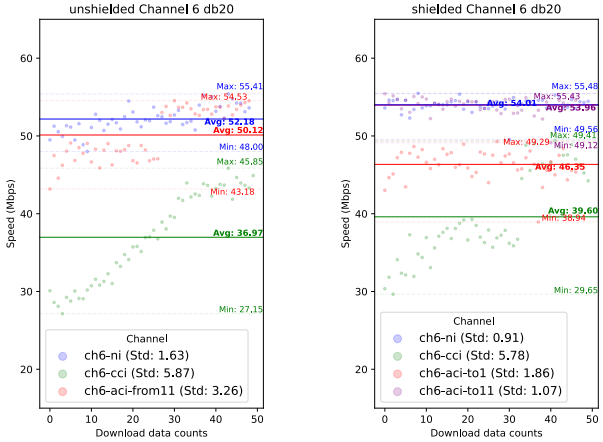


Figure 4.21: Cross networks, channel 6 interference at the same attenuation, download comparison. Std is short for standard deviation.

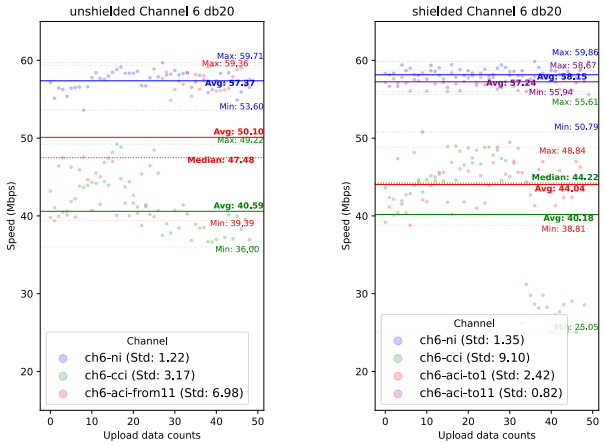


Figure 4.22: Cross networks, channel 6 interference at the same attenuation, upload comparison. Std is short for standard deviation.

Results from channel 11 at 20 dB

This subsection presents the results for cross-network performance at channel 11 and 20 dB. Results are plotted and compared side by side. Figure 4.23 and 4.24 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 11 has 3 datasets for both the shielded and unshielded network, the NI, CCI and ACI.

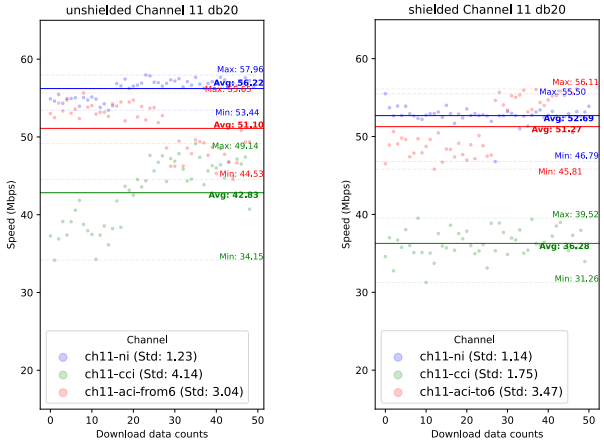


Figure 4.23: Cross networks, channel 11 interference at the same attenuation, download comparison. Std is short for standard deviation.

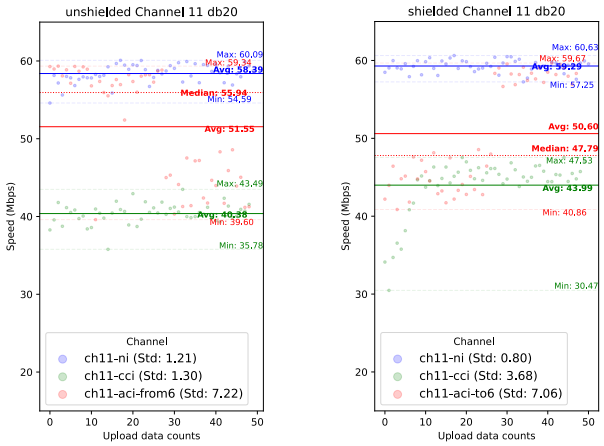


Figure 4.24: Cross networks, channel 11 interference at the same attenuation, upload comparison. Std is short for standard deviation.

Results from channel 1 at 40 dB

This subsection presents the results for cross-network performance at channel 1 and 40 dB. Results are plotted and compared side by side. Figure 4.25 and 4.26 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 1 has only 2 datasets for the shielded network, the NI and ACI.

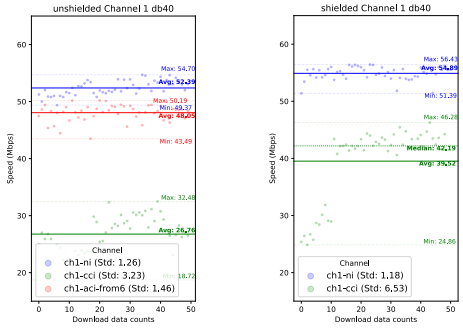


Figure 4.25: Cross networks, channel 1 interference at the same attenuation, download comparison. Std is short for standard deviation.

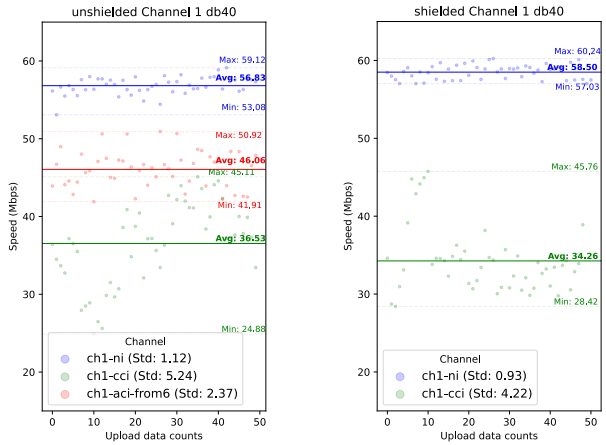


Figure 4.26: Cross networks, channel 1 interference at the same attenuation, upload comparison. Std is short for standard deviation.

Results from channel 6 at 40 dB

This subsection presents the results for cross-network performance at channel 6 and 40 dB. Results are plotted and compared side by side. Figure 4.27 and 4.28 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 6 has 4 datasets for the shielded network, the NI, CCI and 2 sets of ACI.

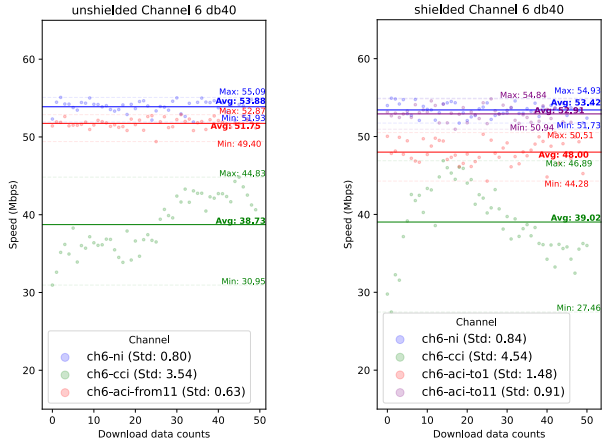


Figure 4.27: Cross networks, channel 6 interference at the same attenuation, download comparison. Std is short for standard deviation.

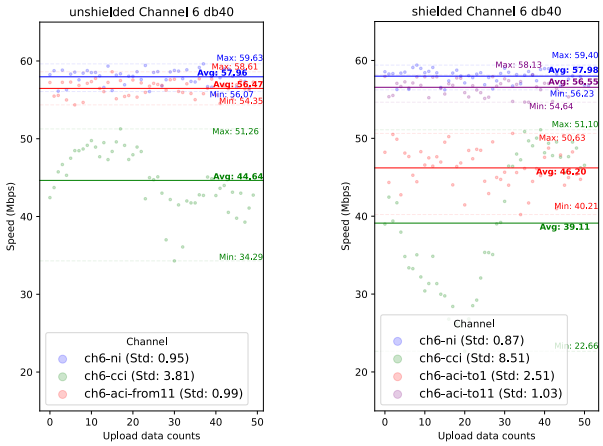


Figure 4.28: Cross networks, channel 6 interference at the same attenuation, upload comparison. Std is short for standard deviation.

Results from channel 11 at 40 dB

This subsection presents the results for cross-network performance at channel 11 and 40 dB. Results are plotted and compared side by side. Figure 4.29 and 4.30 show the results for download and upload respectively. As mentioned in the measurement matrix table 3.3 channel 11 has 3 datasets for both the shielded and unshielded networks, the NI, CCI and ACI.

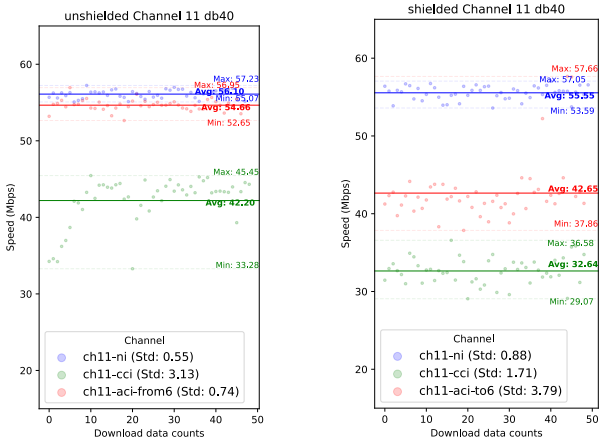


Figure 4.29: Cross networks, channel 11 interference at the same attenuation, download comparison. Std is short for standard deviation.

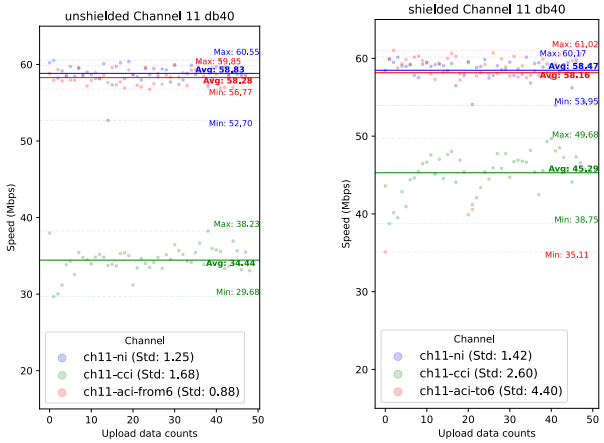


Figure 4.30: Cross networks, channel 11 interference at the same attenuation, upload comparison. Std is short for standard deviation.

4.4 Overlapping channel interference to channel 6 at 0 dB

This section presents the measurement results from the overlapping channels that impose interference to channel 6 at 0dB. The previous section investigated the behaviour of the non-overlapping channel CCI and ACI, we can see that CCI has a much bigger impact on the throughput performance than ACI, this brought me a question "could it be that the more overlapped the channels are, the more the interference is?" Thus, it will be interesting to see if the hypothesis is true. For this section, instead of their environmental condition as in the previous section, terms of shield and unshielded routers are used to distinguish different routers.

Overlapping channel interference to channel 6 from shield network

This subsection presents the download and the upload throughput results for the "shield" network in figures 4.31 and 4.32.

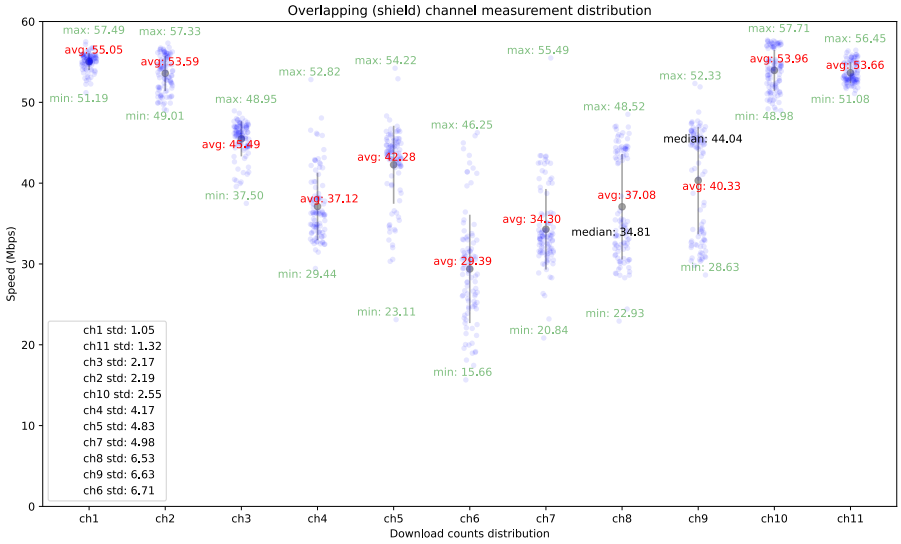


Figure 4.31: Overlapping channel interference to channel 6 at 0 dB attenuation, for the shield network, download. Std is short for standard deviation

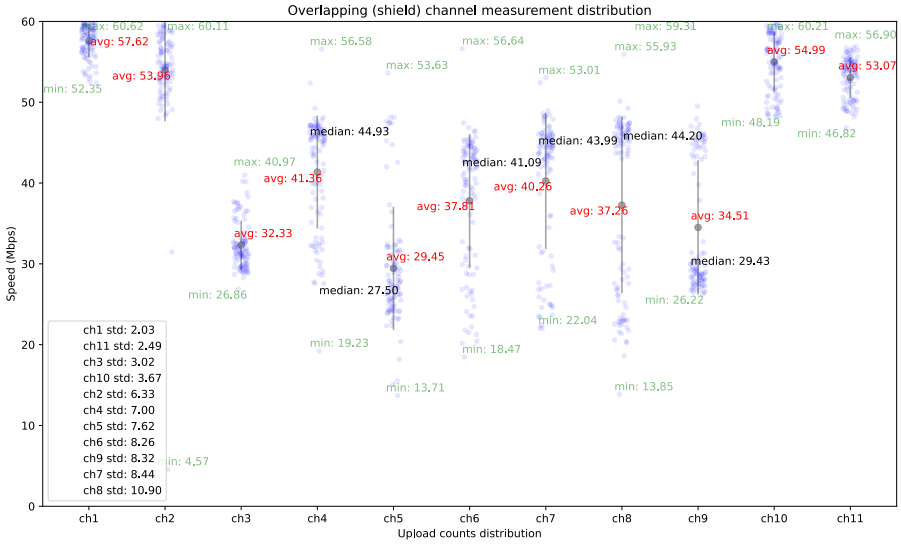


Figure 4.32: Overlapping channel interference to channel 6 at 0 dB attenuation, for the shield network, upload. Std is short for standard deviation

These results are taken from the shielded client, each column means the unshielded client at the relevant channel and imposes interference at channel 6 on the shielded client. From the above, the trend is clear, the less overlapping the channels are the better the Wi-Fi performance. However, some results do not fully fall into this finding. The explanation can be that the environmental interference, and device performance limitation, ie sometimes the devices (raspberry PI) can perform or allocate system resources to tasks other than

Wi-Fi testing. However, if we put the data in categories, we can see from Channels 4 to 8, the performance is the worst; Channels 3 and 9 are the second worst, while Channels 2 and 10 perform better, and the outer channels always perform the best with the smallest standard deviation.

Overlapping channel interference to channel 6 from unshielded network

This subsection presents the download and the upload throughput results for the "unshield" network in figures 4.33 and 4.34.

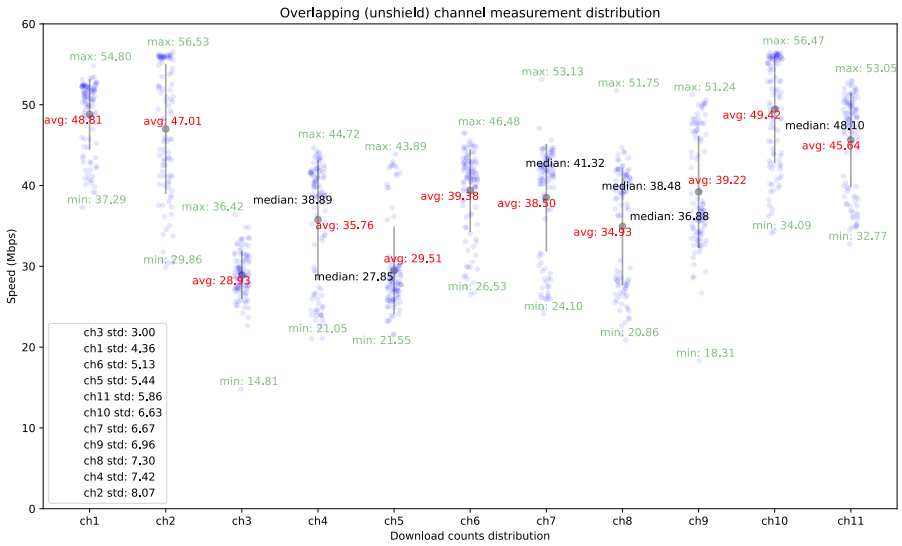


Figure 4.33: Overlapping channel interference to channel 6 at 0 dB attenuation, for the unshielded network, download. Std is short for standard deviation.

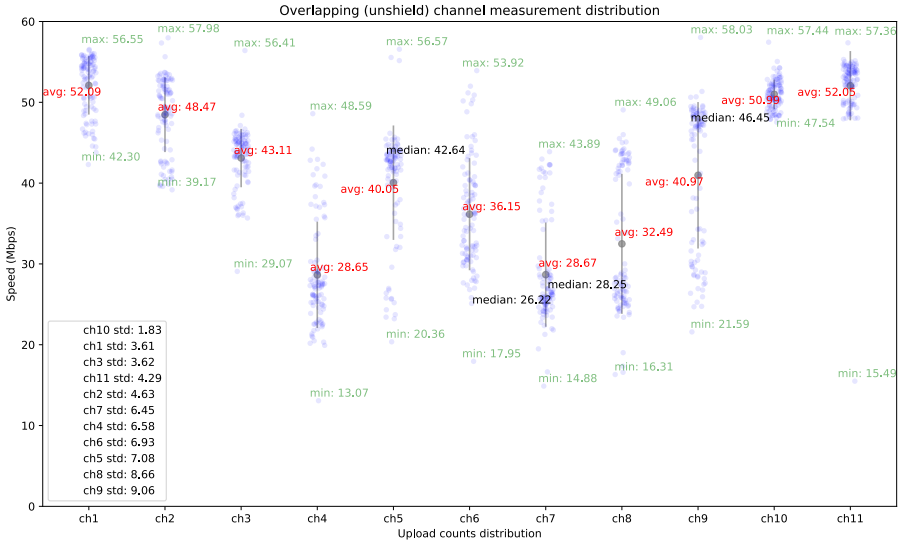


Figure 4.34: Overlapping channel interference to channel 6 at 0 dB attenuation, for the unshielded network, upload. Std is short for standard deviation.

The result of each column is the result of the shielded group at channel 6 imposing interference on the unshielded group at marked channels. Similarly, compared to the shielded group, we can conclude that the less overlapping the channels are, the better the average Wi-Fi performance.

4.5 Interference ratio

The results show a trend or tendency of the interference and device groups' performances, and the results are affected by the environmental interference. I can not say it is ideal, but I think the results are good enough to show the trend of interference. There are some outliers if I compare only the absolute values of the results. To better understand and compare the results, I came up with the "normalized" results for CCI and ACI.

To understand how much the CCI and ACI are when compared to the non-interference (NI), I normalized the performance by dividing its correspondent NI. Thus, I get the interference ratio as the following equations: $R_{cci} = \frac{CCI}{NI}$ and $R_{aci} = \frac{ACI}{NI}$.

The 2 equations for R_{cci} and R_{aci} can tell how much the interference is when compared to different channels and attenuation levels.

- When R < 1, the throughput from the interference group performs worse than the reference (NI).
- The bigger the R-value is, the better the throughput performance of the interference group.

Interference ratio for the same attenuation level

Table 4.6 shows the average performance ratio. The highlighted cells tell that the data is calculated using the median CCI or ACI values over NI. The

highlighted cells mean that the results are the median number instead of the dataset average. The median is better when the data is skewed or contains outliers. It represents the middle value, ensuring that extreme values do not distort the central tendency [29] - [30], the median is used when the difference between mean and median is bigger than 5%, this value is a personal choice.

0db		channel 1		channel 6			channel 11	
		unshielded	shielded	unshielded	shielded		unshielded	shielded
R_{cci}	download	0.71	0.55	0.43	0.67		0.60	0.56
	upload	0.47	0.67	0.64	0.60		0.50	0.71
R_{aci}	ACI from	channel 6		channel 11	channel 1	channel 11	channel 6	channel 6
	download	0.82		0.94	0.79	0.91	0.78	0.84
	upload	0.83		0.99	0.96	0.87	0.98	0.99

Table 4.6: Interference ratio table for 0 dB

20db		channel 1		channel 6			channel 11	
		unshielded	shielded	unshielded	shielded		unshielded	shielded
R_{cci}	download	0.59	0.63	0.71	0.73		0.76	0.76
	upload	0.66	0.84	0.71	0.76		0.69	0.69
R_{aci}	ACI from	channel 6		channel 11	channel 1	channel 11	channel 6	channel 6
	download	0.89		0.96	1.00	0.86	0.91	0.97
	upload	0.77		0.83	0.98	0.76	0.96	0.81

Table 4.7: Interference ratio table for 20 dB

40db		channel 1		channel 6			channel 11	
		unshielded	shielded	unshielded	shielded		unshielded	shielded
R_{cci}	download	0.51	0.77	0.72	0.73		0.75	0.59
	upload	0.64	0.59	0.77	0.67		0.59	0.77
R_{aci}	ACI from	channel 6		channel 11	channel 1	channel 11	channel 6	channel 6
	download	0.92		0.96	0.99	0.90	0.97	0.77
	upload	0.81		0.97	0.98	0.80	0.99	0.99

Table 4.8: Interference ratio table for 40 dB

From table 4.6, 4.7 and 4.8, based on the unshielded group data, I can only conclude that the CCI is always playing a destructive role to its counterpart NI, ACI behaves similarly as CCI, but with smaller effect on the throughput performance.

Interference ratio for the same channel

From table 4.9, 4.10 and 4.11, based on the unshielded group data, I can only conclude that the attenuation level 0 dB is greater than 20 dB and 40 dB, there is no significant difference between 20 and 40 dB. The highlighted cells mean that the results are the median number instead of the dataset average.

channel 1		0 dB		20 dB		40 dB	
		unshielded	shielded	unshielded	shielded	unshielded	shielded
Rcci	download	0.71	0.55	0.59	0.63	0.51	0.77
	upload	0.47	0.67	0.66	0.84	0.64	0.59
Raci	download	0.82		0.89		0.92	
	upload	0.83		0.77		0.81	

Table 4.9: Interference ratio table for channel 1

channel 6		0 dB unshielded	0 dB shielded		20 dB unshielded	20 dB shielded		40 dB unshielded	40 dB shielded	
R_{cci}	download	0.43	0.67		0.71	0.73		0.72	0.73	
	upload	0.64	0.60		0.71	0.76		0.77	0.67	
R_{aci}	ACI from	channel 11	channel 1	channel 11	channel 11	channel 1	channel 11	channel 6	channel 1	channel 11
	download	0.94	0.79	0.91	0.96	1.00	0.86	0.96	0.99	0.90
	upload	0.99	0.96	0.87	0.83	0.98	0.76	0.97	0.98	0.80

Table 4.10: Interference ratio table for channel 6

channel 11		0 dB unshielded	0 dB shielded	20 dB unshielded	20 dB shielded	40 dB unshielded	40 dB shielded
Rcci	download	0.60	0.56	0.76	0.76	0.75	0.59
	upload	0.50	0.71	0.69	0.69	0.59	0.77
Raci	download	0.78	0.84	0.91	0.97	0.97	0.77
	upload	0.98	0.99	0.96	0.81	0.99	0.99

Table 4.11: Interference ratio table for channel 11

CHAPTER 5

Conclusion and discussion

This chapter presents a conclusion based on the results from chapter 4. In addition, I also added some discussions, to elaborate on some of the difficulties I have met in my project, along with some workarounds I made. Furthermore, I listed possible continuous work for similar types of research for different setups or radio interference.

5.1 Conclusion for the results

With the results from chapter 4, I can answer the questions from subsection 1 in chapter 1.

- Which channel imposes the most interference on the operating channels?
  From the experiement, I do not find solid results to answer this question.
- How much performance reduction is caused by interference?
  For co-channel interference (CCI) the reduction can go as high as 60% of the best performance.
  For adjacent channel interference (ACI) the reduction is minuscule in most cases, especially when the interference signal strength is at an attenuation level equivalent to 20 dB and more.
  A more detailed analysis is in the following subsection 5.1 below.
- Can we consider the interference significant?
  Personally speaking, yes. I consider the CCI significant.
  ACI is also significant in most cases.

In general, there is a tendency that the more bandwidths of the channels are overlapped, the more interference there will be. We can see that CCI results have the most suppression, then follow the ACI, and non-interference (NI) is the reference throughput performance.

Non-overlapping interference analysis

For the CCI, R_{cci} ranges from as low as 0.43 to 0.77 for download throughput performance. R_{cci} ranges from as low as 0.47 to 0.77 for upload throughput performance.

For the ACI, R_{aci} ranges from as low as 0.78 to 1.0 for download throughput performance. R_{aci} ranges from as low as 0.77 to 0.99 for upload throughput performance.

In most cases, CCI is significant, it can reduce the network speed to lower than half of the best device throughput performance and should be treated properly. The inconsistency and outliers of the results can be caused by device warmup periods for proper functionality and possibly environmental interference from the lab devices or wakeup of surrounding devices at a certain time. Such implementation of protocols may lead to erratic behaviour [13]. Depending on the condition, if we only check the shielded network, as shown from tables 4.6- 4.8 to 4.9- 4.11, when the attenuation level is more than 20 dB, ACI is minuscule. I can safely conclude that if the attenuation is equivalent to 20 dB and more, ACI is not significant for normal use.

Overlapping interference analysis

From the results of the overlapping analysis based on the shielded device group, as shown in figures 4.31, 4.32, 4.33 and 4.34, the trend is clear the more overlapped the channels are, the more the interference is and the worse the devices perform. In other words, CCI is always greater than ACI, and there is a significant performance drop in CCI when compared to ACI and NI.

5.2 Discussion

When working on the project, I got some suggestions to investigate other network tools, such as iperf. Since iperf delivers results faster and can be further modified according to the needs. For simplicity and the sole purpose of measuring throughput performance, there is no need to connect to the Internet if interference is the only concern. Thus, one can set up 2 identical hosts and 2 identical clients using iperf [31] for throughput interference measurements. When there are 2 hosts, each client can be connected delicately to its host and establish the radio communications, the host can be seen as a "remote" server. In this setup, the measurements can be done similarly to my measurements mentioned in section 3.3 in chapter 3. I will discuss why I did not implement this in the subsection 5.2 of this section.

Difficulties and workarounds

In the project, I faced some difficulties, I would like to bring them up and show some of my solutions.

The most time-consuming difficulty I had to conquer was finding good measurement locations and network connections. I made multiple measurements at different locations. The results of CCI, ACI and NI does not constitute solid results, sometimes I have ACI or even CCI greater than the NI. This is because my network is not well designed for dedicated tests for thousands of repeats, the switch I had was not powerful enough to evenly distribute the network capacity, and my network speed did not support the Raspberry PIs to run at "max" throughput. I can not set up good networks to perform the tests and I discovered this after multiple attempts.

A similar story to my home network happened with several measurements made in the campus lab. Later, I found in this work [13], that in actual physical setup, wireless experimentation can sometimes be more misleading. It is difficult

to reproduce the "same" results, and it is difficult to identify what is driving the behaviour of a testbed that can lead to false findings.

Only after finding this from this related work, did I realise that my measurements for each channel have to be close in measurement time. I implemented the confirmation steps as designed in subsection B.1 in chapter 3. This means, that my test orders of NI for different channels must have been changed according to table 5.1. Only in this setup, did I get better results that made sense. Then, I performed several repeats of the same set and got my results as shown in chapter 4.2 and appendix A.1.

Test orders	Initial
1	NI of channels 1, 6 and 11
2	CCI of channels 1, 6 and 11
3	ACI according to table 3.3

→

Changed
NI, CCI, ACI of channel 1
NI, CCI, ACI of channel 6
NI, CCI, ACI of channel 11

Table 5.1: Measurement order changes

In the meantime, I encountered some other issues. Solving these problems took me quite some time as well.

- Difficult routers. The routers were not equipped with the latest firmware, it has to be returned to the producer and re-factorized. This took some time, but afterwards, I could access the web GUI to change channels and modify the relevant parameters. It is always better that one can have, if not the root, more control over the devices, so you can change the frequencies, transmission power or even wake-up time and beyond freely.
- Environmental interference. The environment is everywhere, some signals are strong and may lead to inaccurate analysis. The biggest interference can be seen on channels 6, 8 and 11 as shown in figure 3.4.
- Slow client devices. Raspberry Pi are not an ideal device for similar projects, the device has quite some limitations for high-performance usage. The computing resources are not optimized for Wi-Fi connection. Meanwhile, one must be familiar with the Linux OS and operate across platforms.
- The Original plan was to design some algorithms to automatically change the channels based on interference. But we lost the project initiator. Thus, the entire following plan was altered to measurement-based tasks. So, the original devices prepared were not well designed for this project. Many devices were added after several experiment rounds. In some cases, I must re-write some scripts for automation, test different measurement steps to compare the results and re-do the experiments several times.
- Plotting can be difficult when you want something unique for your work.

One more big challenge I faced was trying to replace the speed test API with iperf [31]. I spent some time and managed to learn and implement the iperf [32] in one client-to-host setup, and two-clients-to-host setup. However, I was not sure, if the host would allocate resources evenly to 2 clients when they are

connected to the same host, I did not fully understand port forwarding within one host, yet to find a good way to monitor and allocate ports to different clients. After a few weeks of study and multiple attempts, I thought if I were to change to iperf completely, I must change devices and redesign the measurement setup, which leads to too many unknowns. So the plan was not adopted.

In my iperf application, I have the following logic:

- Host starts iperf.exe and allocates ports for the clients.
- Client, revokes the iperf.exe and tries to connect to the host. The clients stop after a certain number of repeats.

Future work

A target or following project can be extended to find out the interference significance and based on the findings to design radio filters, or design algorithms for the routers to automatically change channels to avoid crowded signal channels in the neighbouring environment. There are a few more future works that can be done for similar projects.

From the related work section 2.3, as mentioned in this work [13]: simulation can provide a measure against which experimental testbeds can be calibrated in their simplest form before embarking into the evaluation of sophisticated mechanisms addressing alternative MAC and routing protocols in large scale.

Measure higher frequencies

5 GHz network is being used more and more, and there is a trend to replace 2.4 GHz due to the higher throughput. This project originally planned to perform such measurements, but the routers in my project do not support changing channels on a 5 GHz network. My current setup should be sufficient to work on 5 GHz and 6 GHz if someone would work on similar projects. It will be interesting to see how different frequencies can impact the throughput performance when comparing different frequencies in a wider range than 2.4 GHz.

Reduce environment interference or simulate

In my project, environmental interference is present, and potential active network transmissions from unknown sources can add up to the uncertainty of the device measurement erratic behaviours caused by potential hardware protocols [13]. Thus, it is advised to measure in a "Faraday room" to minimize environmental interference as much as possible.

Appendices

APPENDIX A

The Appendix For the unshielded Group Results

A.1 Plots for the shielded devices

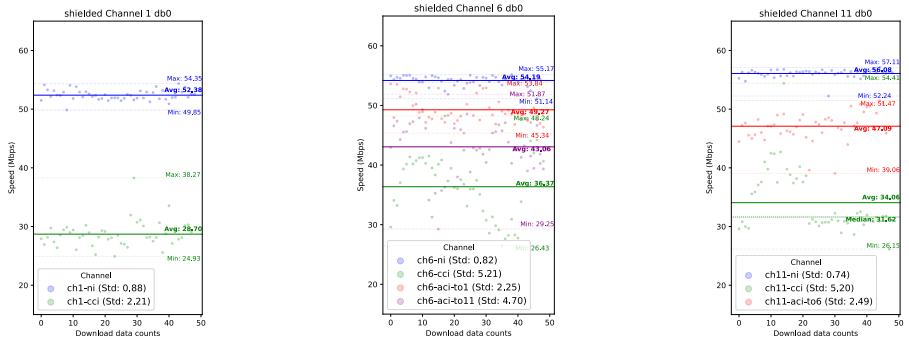


Figure A.1: Shielded AP, cross channel interference at 0 dB attenuation, download comparison. Std is short for standard deviation.

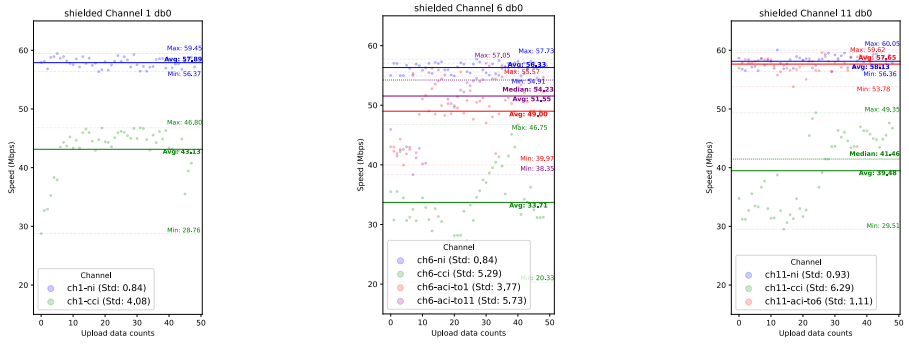


Figure A.2: Shielded AP, cross channel interference at 0 dB attenuation, upload comparison. Std is short for standard deviation.

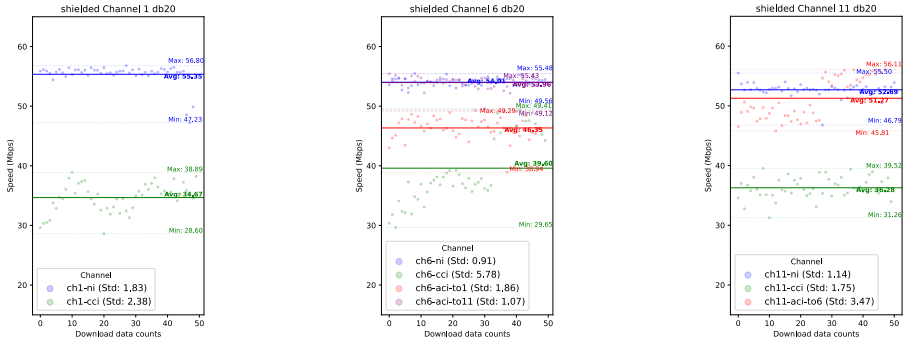


Figure A.3: Shielded AP, cross channel interference at 20 dB attenuation, download comparison. Std is short for standard deviation.

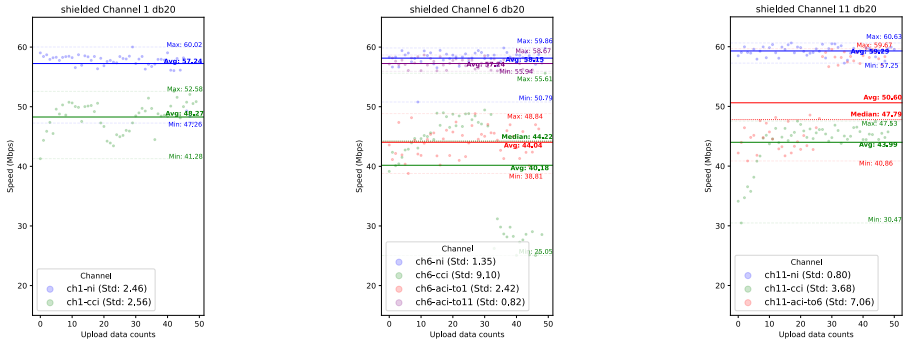


Figure A.4: Shielded AP, cross channel interference at 20 dB attenuation, upload comparison. Std is short for standard deviation.

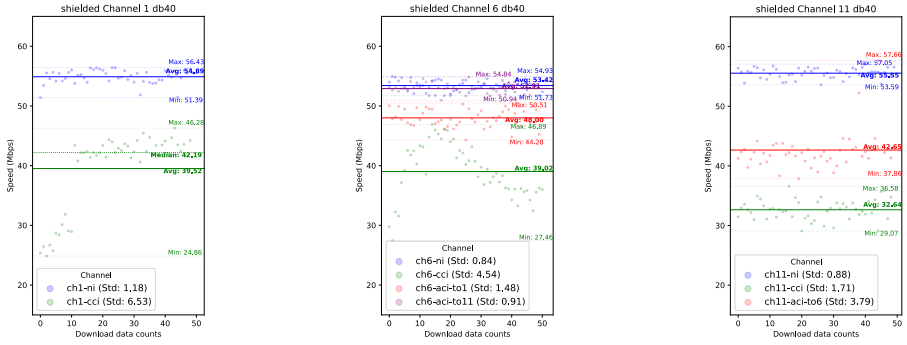


Figure A.5: Shielded AP, cross channel interference at 40 dB attenuation, download comparison. Std is short for standard deviation.

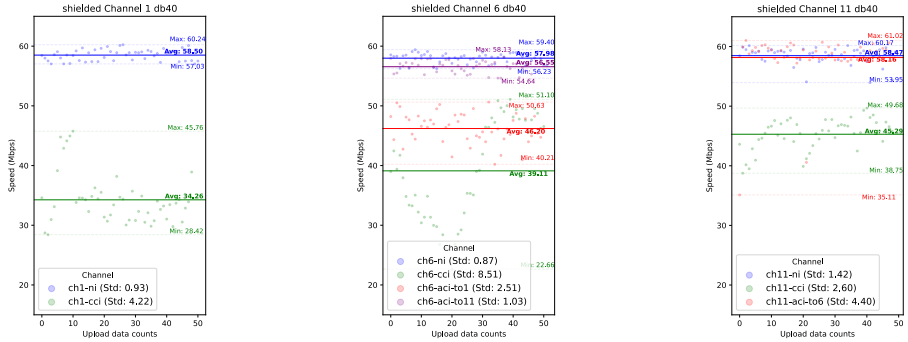


Figure A.6: Shielded AP, cross channel interference at 40 dB attenuation, upload comparison. Std is short for standard deviation.

APPENDIX B

The Appendix For Scripts

This appendix includes the scripts that are written by me, and the source code for speedtest.py from github with some modifications.

B.1 Automated tests

The section contains the python scripts for automated tests and save results to wanted format and files.

```
1000 import subprocess
     import time
1002 import datetime

1004 def git():
         subprocess.run(["git", "add", "-A", "results/"])
1006     print("Changes added successfully.")

1008     subprocess.run(["git", "commit", "-m", "'new results'"])
         print("Changes committed successfully.")
1010
         subprocess.run(["git", "push", "origin", "unshield"])
1012     print("Changes pushed successfully.")

1014 if __name__ == "__main__":
         # Replace 'script.py' with the name of the Python script you
         want to run
1016     start_time = time.time()

1018     script_path = "speedtest.py"
         schedule_time = "21:36"
1020     repeat_count = 52 # The shielded group will have a bigger
         number of repeats than the unshielded to cover the unshielded
         group
         while True:
1022         current_time = datetime.datetime.now().strftime("%H:%M")
             if current_time == schedule_time:
1024             for i in range(repeat_count):
                     subprocess.run(["python", script_path, "--secure"
         ])
1026                 time.sleep(1)
                 git()
1028             break
             else:
1030             time.sleep(1)
```

B.2 Plot the results

The section contains the python scripts for analysing and plotting the results.

```
# %% [markdown]
# Import necessary modules

# %%
import os, re
import pandas as pd
import matplotlib.pyplot as plt
from adjustText import adjust_text
import seaborn as sns

# %% [markdown]
# Extract files and data from saved excel files.

# %%
def extract_files(dir):
    file_list = []
    for root, dirs, files in os.walk(dir):
        for file in files:
            file_list.append(os.path.join(root, file))

    return file_list


def extract_names(file_list):
    names=[]
    for iter in range(len(file_list)):
        name = file_list[iter].split("\\")[-1]
        name = re.sub(r"-shield\.xlsx|-unshield\.xlsx","" , name)
        names.append(name)

    return names


def sort_files(file_list, type="", noi = False):
    # Filter the list to include only 'ch1', 'ch6', and 'ch11'
    filtered_list = [f for f in file_list if 'ch1' in f or 'ch6'
    in f or 'ch11' in f]

    if type == "ovl":
        # Further filter to include only 'ovl' type files
        ovl_list = [f for f in filtered_list if 'ovl' in f]
        # Sort the 'ovl' list by the 'from' number
        ovl_list.sort(key=lambda x: int(x.split('from')[1].split('
    -')[0]))
        return ovl_list
    else:
        # Define the order for 'ch' and secondary order for 'ni',
    'coi', 'aci'
        ch_order = {'ch1': 0, 'ch6': 1, 'ch11': 2}
        if not noi:
            secondary_order = {'ni': 0, 'cci': 1, 'aci': 2}
        elif noi:
            secondary_order = {'cci': 0, 'aci': 1}

        # Sort the remaining filtered list
        sorted_list = sorted(filtered_list, key=lambda x: (
            ch_order.get(x.split('\\')[1].split('-')[0], 3),
            secondary_order.get(x.split('-')[1].split('-')[0], 3)
        ))
```

```
        return sorted_list

# %% [markdown]
# Transform the file into desired dataframe format

# %%
def file_transform(file, name, df_type = "combination"):
    """transform the input file to wanted data_frame"""
    dn = name + "-download"
    up = name + "-upload"
    df = pd.read_excel(file).rename(columns={"download":dn, "upload": up})
    df_transform = df[[dn, up]].div(1e6)

    if df_type == "upload":
        vl_vars = [up]
        type_map = {up : "upload"}
        ch_map = {up: name}
    elif df_type == "download":
        vl_vars = [dn]
        type_map = {dn : "download"}
        ch_map = {dn: name}
    else:
        vl_vars = [dn, up]
        type_map = {dn: 'download', up: 'upload'}
        ch_map = {dn: name, up: name}

    df_melt = df_transform.melt(value_vars=vl_vars, var_name="channel-interference", value_name="speed/Mbps")
    df_melt['type'] = df_melt['channel-interference'].map(type_map)

    df_melt['channel'] = df_melt['channel-interference'].map(ch_map)

    return df_melt

# %% [markdown]
# Sort files for overlapping results.

# %%
def sort_ovl(file_name):
    import re
    match = re.search(r'ch(\d+)', file_name)
    return int(match.group(1)) if match else float('inf')

def ovl_names(file_list):
    names=[]
    for iter in range(len(file_list)):
        name = file_list[iter].split("\\")[-1]
        name = re.sub(r"-ovl-shield\.xlsx|-ovl-unshield\.xlsx","", name)
        names.append(name)

    return names

def same_db(file_list, df_type = "combination", ovl=False):
    df_combinations = []
    if not ovl:
        names = extract_names(file_list)
    elif ovl:
        names = ovl_names(file_list)
```

```
    for iter, file, name in zip(range(len(file_list)),file_list,
    names):
        df = file_transform(file, name, df_type)
        df_combinations.append(df)

        if iter == 0:
            concats = df_combinations[iter]
        elif iter >= 1:
            concats = pd.concat([concats, df_combinations[iter]])

    return concats

# %% [markdown]
# function for extracting single channel with given data frame,
    and relevant channel

# %%
def single_channel(df, channel):
    pattern = re.compile(rf'\b{channel}\b')
    channel_df = df[df['channel-interference'].apply(lambda x:
    bool(pattern.search(x)))]

    return channel_df


def line_plot(df, ch_number, data_type, save_name='', attenuation
    = '0dB', ap_type="shielded", noi=False):

    fig, ax = plt.subplots(figsize=(4, 7))
    ch = df["channel"].unique()
    # Plot each channel category with different colors
    if noi:
        colors = {ch[0]: 'blue', ch[1]: 'red'}
    elif len(ch) == 2:
        colors = {ch[0]: 'blue', ch[1]: 'green'}
    elif len(ch) == 3:
        colors = {ch[0]: 'blue', ch[1]: 'green', ch[2]: 'red'}
    elif len(ch) == 4:
        colors = {ch[0]: 'blue', ch[1]: 'green', ch[2]: 'red', ch
    [3]: 'purple'}
    texts = []

    for channel in df['channel'].unique():
        channel_data = df[df['channel'] == channel]
        x = range(len(channel_data))
        y = channel_data['speed/Mbps']
        ax.scatter(x, y, label=f'{channel} (Std: {y.std():.2f})',
    color=colors[channel], alpha=0.2, s=7)

        # Calculate statistics
        mean = y.mean()
        median = y.median()
        max_val = y.max()
        min_val = y.min()
        std_dev = y.std()

        # Plot statistics
        ax.axhline(y=max_val, color=colors[channel], linestyle='—
    ', linewidth=1, alpha=0.1)
        ax.axhline(y=min_val, color=colors[channel], linestyle='—
    ', linewidth=1, alpha=0.1)
        ax.axhline(y=mean, color=colors[channel], linestyle='-',
```

```
    linewidth=1)
        if abs(mean - median) / mean > 0.05:
            # plt.axhline(y=median, color=colors[channel],
    linestyle=':', linewidth=1)
            ax.axhline(y=median, color=colors[channel], linestyle=
    ':', linewidth=1)

        # Annotate statistics
        texts.append(ax.text(len(channel_data) - 1, mean, f'Avg: {
    mean:.2f}', horizontalalignment='center', size='small', color=
    colors[channel], weight='semibold'))
        if abs(mean - median) / mean > 0.05:
            texts.append(ax.text(len(channel_data) - 1, median, f'
    Median: {median:.2f}', horizontalalignment='center', size='
    small', color=colors[channel], weight='semibold'))
        texts.append(ax.text(len(channel_data) - 1, max_val, f'Max
    : {max_val:.2f}', horizontalalignment='center', size='small',
    color=colors[channel]))
        texts.append(ax.text(len(channel_data) - 1, min_val, f'Min
    : {min_val:.2f}', horizontalalignment='center', size='small',
    color=colors[channel]))

    # Adjust text to avoid overlap
    adjust_text(texts)

    ax.set_xlabel(f'{data_type.capitalize()} data counts')
    ax.set_ylabel('Speed (Mbps)')
    ax.set_title(f'{ap_type} {ch_number} {attenuation}')
    ax.legend(title='Channel', loc="lower left", fontsize="12",
    markerscale = 3)
    # ax.grid(True)

    plt.ylim(0,65)
    pyPath = os.getcwd()
    os.chdir('figures/box/')
    # plt.savefig(f'{save_name}.svg', dpi='figure', bbox_inches='
    tight', transparent=True)
    plt.savefig(f'{save_name}.png', dpi='figure', bbox_inches='
    tight', transparent=True)
    plt.close()
    os.chdir(pyPath)

# %% [markdown]
# results for db0

# %%
def channel_save(attenuation, title_str, ch_str, data_type,
    ap_type = "shielded", noi = False):
    # db_path = f"results/uio/{ap_type}/{attenuation}"
    db_path = f"results/box/{ap_type}/{attenuation}"
    db_files = sort_files(extract_files(db_path), noi=noi)
    db_df = same_db(db_files, data_type)
    db_ch_df = single_channel(db_df, ch_str)
    if "novl" in attenuation:
        attenuation = attenuation.replace("/novl", "")
    save_name = f"{ap_type}_{attenuation}_{ch_str}_{data_type}"

    line_plot(df=db_ch_df, ch_number=title_str, data_type=
    data_type, save_name=save_name, attenuation=attenuation, noi=
    noi, ap_type=ap_type)

# %% [markdown]
```

```
# setup the file directories and channels

# %%
# change everything to shielded and unshielded
attenuations = ["db0", "db20", "db40"]
ap_types = ["shielded", "unshielded"]
title_strs = ["Channel 1", "Channel 6", "Channel 11"]
ch_strs = ["ch1", "ch6", "ch11"]
data_types = ["download", "upload"]

# %% [markdown]
# plot the shield datasets

# %%
for attenuation in attenuations:
    for ch, title in zip(ch_strs, title_strs):
        for type in data_types:
            channel_save(attenuation=attenuation, title_str=title,
     ch_str=ch, data_type=type, ap_type="shielded")

# %% [markdown]
# plot the unshield datasets, changed to add non-interference to
    unshielded noi = noi

# %%
for attenuation in attenuations:
    # if attenuation != "db0":
    ni = True
    for ch, title in zip(ch_strs, title_strs):
        for type in data_types:
            channel_save(attenuation=attenuation, title_str=title,
     ch_str=ch, data_type=type, ap_type="unshielded")

# %% [markdown]
# function for the overlapping dataset

# %%
def ovl_plot(df_ovl, save_name, data_type):

    df = df_ovl
    # Plot
    plt.figure(figsize=(14, 8))
    ax = sns.stripplot(x='channel', y='speed/Mbps', data=df,
    jitter=True, color='blue', alpha=0.1)


    # Calculate statistics
    stats = df.groupby('channel')['speed/Mbps'].agg(['min', 'max',
     'mean', 'median', 'std']).reset_index()
    stats = stats.sort_values(by='std')

    # Calculate and plot statistics
    texts = []
    avg_points = []
    std_devs = {}
    for channel in df['channel'].unique():
        channel_data = df[df['channel'] == channel]['speed/Mbps']
        mean = channel_data.mean()
        median = channel_data.median()
        max_val = channel_data.max()
        min_val = channel_data.min()
        std_dev = channel_data.std()
```

```
            std_devs[channel] = std_dev

            # Plot min and max
            texts.append(ax.text(df['channel'].unique().tolist().index(channel), min_val, f'min: {min_val:.2f}', color='green', alpha=0.5, ha='center'))
            texts.append(ax.text(df['channel'].unique().tolist().index(channel), max_val, f'max: {max_val:.2f}', color='green', alpha=0.5, ha='center'))

            # Plot mean
            texts.append(ax.text(df['channel'].unique().tolist().index(channel), mean, f'avg: {mean:.2f}', color='red', ha='center'))
            avg_points.append((df['channel'].unique().tolist().index(channel), mean))

            # Plot median if the difference between mean and median is bigger than 5%
            if abs(mean - median) / mean > 0.05:
                texts.append(ax.text(df['channel'].unique().tolist().index(channel), median, f'median: {median:.2f}', color='black', ha='center'))

            # Plot standard deviation
            ax.errorbar(df['channel'].unique().tolist().index(channel), mean, yerr=std_dev, fmt='o', color='gray', alpha=0.7)

        # Add legend for standard deviation
        for i, row in stats.iterrows():
            plt.plot([], [], ' ', label=f"{row['channel']} std: {row['std']:.2f}")

        plt.legend()
        plt.ylim(0, 60)

        # Adjust text to avoid overlap
        adjust_text(texts)

        plt.xlabel(f'{data_type} counts distribution')
        plt.ylabel('Speed (Mbps)')
        plt.title(f'Overlapping ({save_name.split("_")[1]}) channel measurement distribution')

        pyPath = os.getcwd()
        os.chdir('figures/')
        plt.savefig(f'{save_name}.svg', dpi='figure', bbox_inches='tight', transparent=True)
        plt.close()
        os.chdir(pyPath)
        # plt.show()

# %%
ovl_s = "results/uio/shielded/ovl"
ovl_us = "results/uio/unshielded/ovl"

# shield dataset
ovl_s_files = sorted(extract_files(ovl_s), key=sort_ovl)
ovl_s_names = ovl_names(ovl_s_files)

s_dn = same_db(ovl_s_files, "download", ovl=True)
s_up = same_db(ovl_s_files, "upload", ovl=True)
```

```
# unshield dataset
ovl_us_files = sorted(extract_files(ovl_us), key=sort_ovl)
ovl_us_names = ovl_names(ovl_us_files)

us_dn = same_db(ovl_us_files, "download", ovl=True)
us_up = same_db(ovl_us_files, "upload", ovl=True)


# %%
ovl_plot(s_dn, "ovl_shield_download", "Download")
ovl_plot(s_up, "ovl_shield_upload", "Upload")
ovl_plot(us_dn, "ovl_unshield_download", "Download")
ovl_plot(us_up, "ovl_unshield_upload", "Upload")
```

B.3 utilities.py

The section contains the python scripts for sorting and saving the speed test results to Excel files.

```
    import os
    import ast
    import pandas as pd


    def write_to_csv(input_str):
        # Convert single quotes to double quotes for valid JSON
        input_str = input_str.replace("'", "\"")

        # Convert the input string to a dictionary
        input_dict = eval(input_str)

        # Extract keys for the header row
        header = []
        for key, value in input_dict.items():
            if isinstance(value, dict):
                header.extend([f"{key}_{subkey}" for subkey in
    value.keys()])
            else:
                header.append(key)

        # Extract values for the data row
        values = []
        for key, value in input_dict.items():
            if isinstance(value, dict):
                values.extend(value.values())
            else:
                values.append(value)

        # Write to CSV
        with open('output.csv', 'w', newline='') as csvfile:
            csv_writer = csv.writer(csvfile)
            csv_writer.writerow(header)
            csv_writer.writerow(values)


    def convert_and_save_to_xlsx(data_str, file_name):
        # Try to open the Excel file
```

```
        try:
1038            df = pd.read_excel(file_name)
        except IOError:
1040            # File doesn't exist, create a new one
                df = pd.DataFrame()
1042
        # Convert the input string to a dictionary
1044    data_dict = ast.literal_eval(data_str)

1046    # Flatten the nested dictionaries
        flattened_data = flatten_dict(data_dict)
1048
        # Append the flattened data to the DataFrame
1050    df = df._append(pd.DataFrame(flattened_data, index=[0]),
    ignore_index=True)

1052    # Save the DataFrame to an Excel file
        df.to_excel(file_name, index=False)
1054
    def flatten_dict(d, parent_key='', sep='_'):
1056        """
        Flatten a nested dictionary by joining keys with a
    separator.
1058        """
        items = []
1060    for k, v in d.items():
            new_key = f"{parent_key}{sep}{k}" if parent_key else k
1062        if isinstance(v, dict):
                items.extend(flatten_dict(v, new_key, sep=sep).
    items())
1064        else:
                items.append((new_key, v))
1066    return dict(items)
```

B.4 speedtest.py

The section contains source code in python for speedtest CLI [23].

```
1000

    #!/usr/bin/env python
1002# -*- coding: utf-8 -*-
    # Copyright 2012 Matt Martz
1004# All Rights Reserved.
    #
1006#    Licensed under the Apache License, Version 2.0 (the "
    License"); you may
    #    not use this file except in compliance with the License.
    You may obtain
1008#    a copy of the License at
    #
1010#         http://www.apache.org/licenses/LICENSE-2.0
    #
1012#    Unless required by applicable law or agreed to in writing,
    software
    #    distributed under the License is distributed on an "AS IS"
    BASIS, WITHOUT
1014#    WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied. See the
```

```
#    License for the specific language governing permissions and
    limitations
#    under the License.

import csv
import datetime
import errno
import math
import os
import platform
import re
import signal
import socket
import sys
import threading
import timeit
import xml.parsers.expat
import utilities
import time


try:
  import gzip
  GZIP_BASE = gzip.GzipFile
except ImportError:
  gzip = None
  GZIP_BASE = object

__version__ = '2.1.4b1'


class FakeShutdownEvent(object):
  """Class to fake a threading.Event.isSet so that users of this
   module
  are not required to register their own threading.Event()
  """

  @staticmethod
  def isSet():
    "Dummy method to always return false"""
    return False

  is_set = isSet


# Some global variables we use
DEBUG = False
_GLOBAL_DEFAULT_TIMEOUT = object()
PY25PLUS = sys.version_info[:2] >= (2, 5)
PY26PLUS = sys.version_info[:2] >= (2, 6)
PY32PLUS = sys.version_info[:2] >= (3, 2)
PY310PLUS = sys.version_info[:2] >= (3, 10)

# Begin import game to handle Python 2 and Python 3
try:
  import json
except ImportError:
  try:
    import simplejson as json
  except ImportError:
    json = None

```

```
  try:
1076    import xml.etree.ElementTree as ET
    try:
1078      from xml.etree.ElementTree import _Element as ET_Element
    except ImportError:
1080      pass
  except ImportError:
1082    from xml.dom import minidom as DOM
    from xml.parsers.expat import ExpatError
1084    ET = None

1086  try:
    from urllib2 import (urlopen, Request, HTTPError, URLError,
1088                 AbstractHTTPHandler, ProxyHandler,
                     HTTPDefaultErrorHandler, HTTPRedirectHandler,
1090                 HTTPErrorProcessor, OpenerDirector)
  except ImportError:
1092    from urllib.request import (urlopen, Request, HTTPError,
    URLError,
                      AbstractHTTPHandler, ProxyHandler,
1094                  HTTPDefaultErrorHandler, HTTPRedirectHandler,
                      HTTPErrorProcessor, OpenerDirector)
1096
  try:
1098    from httplib import HTTPConnection, BadStatusLine
  except ImportError:
1100    from http.client import HTTPConnection, BadStatusLine

1102  try:
    from httplib import HTTPSConnection
1104  except ImportError:
    try:
1106      from http.client import HTTPSConnection
    except ImportError:
1108      HTTPSConnection = None

1110  try:
    from httplib import FakeSocket
1112  except ImportError:
    FakeSocket = None
1114
  try:
1116    from Queue import Queue
  except ImportError:
1118    from queue import Queue

1120  try:
    from urlparse import urlparse
1122  except ImportError:
    from urllib.parse import urlparse
1124
  try:
1126    from urlparse import parse_qs
  except ImportError:
1128    try:
      from urllib.parse import parse_qs
1130    except ImportError:
      from cgi import parse_qs
1132
  try:
1134    from hashlib import md5
  except ImportError:
```

```
1136      from md5 import md5

1138    try:
          from argparse import ArgumentParser as ArgParser
1140      from argparse import SUPPRESS as ARG_SUPPRESS
          PARSER_TYPE_INT = int
1142      PARSER_TYPE_STR = str
          PARSER_TYPE_FLOAT = float
1144    except ImportError:
          from optparse import OptionParser as ArgParser
1146      from optparse import SUPPRESS_HELP as ARG_SUPPRESS
          PARSER_TYPE_INT = 'int'
1148      PARSER_TYPE_STR = 'string'
          PARSER_TYPE_FLOAT = 'float'
1150
        try:
1152      from cStringIO import StringIO
          BytesIO = None
1154    except ImportError:
          try:
1156        from StringIO import StringIO
            BytesIO = None
1158      except ImportError:
            from io import StringIO, BytesIO
1160
        try:
1162      import __builtin__
        except ImportError:
1164      import builtins
          from io import TextIOWrapper, FileIO
1166
          class _Py3Utf8Output(TextIOWrapper):
1168        """UTF-8 encoded wrapper around stdout for py3, to override
            ASCII stdout
1170        """
            def __init__(self, f, **kwargs):
1172          buf = FileIO(f.fileno(), 'w')
              super(_Py3Utf8Output, self).__init__(
1174            buf,
                encoding='utf8',
1176            errors='strict'
              )
1178
            def write(self, s):
1180          super(_Py3Utf8Output, self).write(s)
              self.flush()
1182
          _py3_print = getattr(builtins, 'print')
1184      try:
            _py3_utf8_stdout = _Py3Utf8Output(sys.stdout)
1186        _py3_utf8_stderr = _Py3Utf8Output(sys.stderr)
          except OSError:
1188        # sys.stdout/sys.stderr is not a compatible stdout/stderr
          object
            # just use it and hope things go ok
1190        _py3_utf8_stdout = sys.stdout
            _py3_utf8_stderr = sys.stderr
1192
          def to_utf8(v):
1194        """No-op encode to utf-8 for py3"""
            return v
1196
```

```
    def print_(*args, **kwargs):
      """Wrapper function for py3 to print, with a utf-8 encoded stdout"""
      if kwargs.get('file') == sys.stderr:
        kwargs['file'] = _py3_utf8_stderr
      else:
        kwargs['file'] = kwargs.get('file', _py3_utf8_stdout)
      _py3_print(*args, **kwargs)
  else:
    del __builtin__

    def to_utf8(v):
      """Encode value to utf-8 if possible for py2"""
      try:
        return v.encode('utf8', 'strict')
      except AttributeError:
        return v

    def print_(*args, **kwargs):
      """The new-style print function for Python 2.4 and 2.5.

      Taken from https://pypi.python.org/pypi/six/

      Modified to set encoding to UTF-8 always, and to flush after write
      """
      fp = kwargs.pop("file", sys.stdout)
      if fp is None:
        return

      def write(data):
        if not isinstance(data, basestring):
          data = str(data)
        # If the file has an encoding, encode unicode with it.
        encoding = 'utf8' # Always trust UTF-8 for output
        if (isinstance(fp, file) and
            isinstance(data, unicode) and
            encoding is not None):
          errors = getattr(fp, "errors", None)
          if errors is None:
            errors = "strict"
          data = data.encode(encoding, errors)
        fp.write(data)
        fp.flush()
      want_unicode = False
      sep = kwargs.pop("sep", None)
      if sep is not None:
        if isinstance(sep, unicode):
          want_unicode = True
        elif not isinstance(sep, str):
          raise TypeError("sep must be None or a string")
      end = kwargs.pop("end", None)
      if end is not None:
        if isinstance(end, unicode):
          want_unicode = True
        elif not isinstance(end, str):
          raise TypeError("end must be None or a string")
      if kwargs:
        raise TypeError("invalid keyword arguments to print()")
      if not want_unicode:
        for arg in args:
          if isinstance(arg, unicode):
```

```
            want_unicode = True
1258            break
        if want_unicode:
1260        newline = unicode("\n")
            space = unicode(" ")
1262    else:
            newline = "\n"
1264        space = " "
        if sep is None:
1266        sep = space
        if end is None:
1268        end = newline
        for i, arg in enumerate(args):
1270        if i:
              write(sep)
1272        write(arg)
        write(end)
1274
    # Exception "constants" to support Python 2 through Python 3
1276 try:
      import ssl
1278  try:
        CERT_ERROR = (ssl.CertificateError,)
1280  except AttributeError:
        CERT_ERROR = tuple()
1282
      HTTP_ERRORS = (
1284    (HTTPError, URLError, socket.error, ssl.SSLError,
      BadStatusLine) +
        CERT_ERROR
1286  )
    except ImportError:
1288  ssl = None
      HTTP_ERRORS = (HTTPError, URLError, socket.error,
      BadStatusLine)
1290
    if PY32PLUS:
1292  etree_iter = ET.Element.iter
    elif PY25PLUS:
1294  etree_iter = ET_Element.getiterator

1296 if PY26PLUS:
      thread_is_alive = threading.Thread.is_alive
1298 else:
      thread_is_alive = threading.Thread.isAlive
1300


1302 def event_is_set(event):
      try:
1304    return event.is_set()
      except AttributeError:
1306    return event.isSet()


1308
    class SpeedtestException(Exception):
1310  """Base exception for this module"""


1312
    class SpeedtestCLIError(SpeedtestException):
1314  """Generic exception for raising errors during CLI operation
      """
```

```
1316
1317    class SpeedtestHTTPError(SpeedtestException):
1318        """Base HTTP exception for this module"""
1319
1320    class SpeedtestConfigError(SpeedtestException):
1321        """Configuration XML is invalid"""
1322
1323    class SpeedtestServersError(SpeedtestException):
1324        """Servers XML is invalid"""
1325
1326    class ConfigRetrievalError(SpeedtestHTTPError):
1327        """Could not retrieve config.php"""
1328
1329    class ServersRetrievalError(SpeedtestHTTPError):
1330        """Could not retrieve speedtest-servers.php"""
1331
1332    class InvalidServerIDType(SpeedtestException):
1333        """Server ID used for filtering was not an integer"""
1334
1335    class NoMatchedServers(SpeedtestException):
1336        """No servers matched when filtering"""
1337
1338    class SpeedtestMiniConnectFailure(SpeedtestException):
1339        """Could not connect to the provided speedtest mini server"""
1340
1341    class InvalidSpeedtestMiniServer(SpeedtestException):
1342        """Server provided as a speedtest mini server does not
1343        actually appear
1344        to be a speedtest mini server
1345        """
1346
1347    class ShareResultsConnectFailure(SpeedtestException):
1348        """Could not connect to speedtest.net API to POST results"""
1349
1350    class ShareResultsSubmitFailure(SpeedtestException):
1351        """Unable to successfully POST results to speedtest.net API
1352        after
1353        connection
1354        """
1355
1356    class SpeedtestUploadTimeout(SpeedtestException):
1357        """testlength configuration reached during upload
1358        Used to ensure the upload halts when no additional data should
1359         be sent
1360        """
1361
1362    class SpeedtestBestServerFailure(SpeedtestException):
1363        """Unable to determine best server"""
1364
1365
```

```
  class SpeedtestMissingBestServer(SpeedtestException):
1376    """get_best_server not called or not able to determine best
        server"""


1378
    def create_connection(address, timeout=_GLOBAL_DEFAULT_TIMEOUT,
1380                source_address=None):
      """Connect to *address* and return the socket object.
1382
      Convenience function.  Connect to *address* (a 2-tuple ``(host
        ,
1384    port)``) and return the socket object.  Passing the optional
      *timeout* parameter will set the timeout on the socket
      instance
1386    before attempting to connect.  If no *timeout* is supplied,
      the
      global default timeout setting returned by :func:`
      getdefaulttimeout`
1388    is used.  If *source_address* is set it must be a tuple of (
      host, port)
      for the socket to bind as a source address before making the
      connection.
1390    An host of '' or port 0 tells the OS to use the default.

1392    Largely vendored from Python 2.7, modified to work with Python
       2.4
      """
1394
      host, port = address
1396    err = None
      for res in socket.getaddrinfo(host, port, 0, socket.
      SOCK_STREAM):
1398      af, socktype, proto, canonname, sa = res
        sock = None
1400      try:
          sock = socket.socket(af, socktype, proto)
1402        if timeout is not _GLOBAL_DEFAULT_TIMEOUT:
            sock.settimeout(float(timeout))
1404        if source_address:
            sock.bind(source_address)
1406        sock.connect(sa)
          return sock
1408
        except socket.error:
1410        err = get_exception()
          if sock is not None:
1412          sock.close()

1414    if err is not None:
        raise err
1416    else:
        raise socket.error("getaddrinfo returns an empty list")
1418


1420  class SpeedtestHTTPConnection(HTTPConnection):
      """Custom HTTPConnection to support source_address across
1422    Python 2.4 - Python 3
      """
1424    def __init__(self, *args, **kwargs):
        source_address = kwargs.pop('source_address', None)
1426      timeout = kwargs.pop('timeout', 10)
```

```
1428        self._tunnel_host = None

1430        HTTPConnection.__init__(self, *args, **kwargs)

1432        self.source_address = source_address
            self.timeout = timeout
1434
      def connect(self):
1436        """Connect to the host and port specified in __init__."""
            try:
1438          self.sock = socket.create_connection(
                (self.host, self.port),
1440            self.timeout,
                self.source_address
1442          )
            except (AttributeError, TypeError):
1444          self.sock = create_connection(
                (self.host, self.port),
1446            self.timeout,
                self.source_address
1448          )

1450        if self._tunnel_host:
              self._tunnel()
1452

1454  if HTTPSConnection:
        class SpeedtestHTTPSConnection(HTTPSConnection):
1456      """Custom HTTPSConnection to support source_address across
          Python 2.4 - Python 3
1458      """
          default_port = 443
1460
          def __init__(self, *args, **kwargs):
1462        source_address = kwargs.pop('source_address', None)
            timeout = kwargs.pop('timeout', 10)
1464
            self._tunnel_host = None
1466
            HTTPSConnection.__init__(self, *args, **kwargs)
1468
            self.timeout = timeout
1470        self.source_address = source_address

1472      def connect(self):
            "Connect to a host on a given (SSL) port."
1474        try:
              self.sock = socket.create_connection(
1476            (self.host, self.port),
                self.timeout,
1478            self.source_address
              )
1480        except (AttributeError, TypeError):
              self.sock = create_connection(
1482            (self.host, self.port),
                self.timeout,
1484            self.source_address
              )
1486
            if self._tunnel_host:
1488          self._tunnel()
```

```
1490            if ssl:
                  try:
1492                kwargs = {}
                    if hasattr(ssl, 'SSLContext'):
1494                  if self._tunnel_host:
                        kwargs['server_hostname'] = self._tunnel_host
1496                  else:
                        kwargs['server_hostname'] = self.host
1498                self.sock = self._context.wrap_socket(self.sock, **
    kwargs)
                  except AttributeError:
1500                self.sock = ssl.wrap_socket(self.sock)
                    try:
1502                  self.sock.server_hostname = self.host
                    except AttributeError:
1504                  pass
              elif FakeSocket:
1506            # Python 2.4/2.5 support
                  try:
1508                self.sock = FakeSocket(self.sock, socket.ssl(self.sock
    ))
                  except AttributeError:
1510                raise SpeedtestException(
                      'This version of Python does not support HTTPS/SSL '
1512                  'functionality'
                    )
1514            else:
                  raise SpeedtestException(
1516                'This version of Python does not support HTTPS/SSL '
                    'functionality'
1518              )


1520
    def _build_connection(connection, source_address, timeout,
      context=None):
1522    """Cross Python 2.4 - Python 3 callable to build an ``
      HTTPConnection`` or
      ``HTTPSConnection`` with the args we need
1524
      Called from ``http(s)_open`` methods of ``SpeedtestHTTPHandler
      `` or
1526    ``SpeedtestHTTPSHandler``
      """
1528    def inner(host, **kwargs):
        kwargs.update({
1530      'source_address': source_address,
          'timeout': timeout
1532      })
        if context:
1534        kwargs['context'] = context
        return connection(host, **kwargs)
1536    return inner


1538
    class SpeedtestHTTPHandler(AbstractHTTPHandler):
1540    """Custom ``HTTPHandler`` that can build a ``HTTPConnection``
      with the
      args we need for ``source_address`` and ``timeout``
1542    """
      def __init__(self, debuglevel=0, source_address=None, timeout
      =10):
1544    AbstractHTTPHandler.__init__(self, debuglevel)
```

```
        self.source_address = source_address
1546    self.timeout = timeout

1548  def http_open(self, req):
        return self.do_open(
1550      _build_connection(
            SpeedtestHTTPConnection,
1552        self.source_address,
            self.timeout
1554      ),
          req
1556    )

1558  http_request = AbstractHTTPHandler.do_request_


1560
    class SpeedtestHTTPSHandler(AbstractHTTPHandler):
1562  """Custom ``HTTPSHandler`` that can build a ``HTTPSConnection
      `` with the
      args we need for ``source_address`` and ``timeout``
1564  """
      def __init__(self, debuglevel=0, context=None, source_address=
      None,
1566           timeout=10):
        AbstractHTTPHandler.__init__(self, debuglevel)
1568    self._context = context
        self.source_address = source_address
1570    self.timeout = timeout

1572  def https_open(self, req):
        return self.do_open(
1574      _build_connection(
            SpeedtestHTTPSConnection,
1576        self.source_address,
            self.timeout,
1578        context=self._context,
          ),
1580      req
        )
1582
      https_request = AbstractHTTPHandler.do_request_
1584


1586  def build_opener(source_address=None, timeout=10):
      """Function similar to ``urllib2.build_opener`` that will
      build
1588  an ``OpenerDirector`` with the explicit handlers we want,
      ``source_address`` for binding, ``timeout`` and our custom
1590  `User-Agent`
      """
1592
      printer('Timeout set to %d' % timeout, debug=True)
1594
      if source_address:
1596    source_address_tuple = (source_address, 0)
        printer('Binding to source address: %r' % (
      source_address_tuple,),
1598        debug=True)
      else:
1600    source_address_tuple = None

1602  handlers = [
```

```
      ProxyHandler(),
      SpeedtestHTTPHandler(source_address=source_address_tuple,
                           timeout=timeout),
      SpeedtestHTTPSHandler(source_address=source_address_tuple,
                            timeout=timeout),
      HTTPDefaultErrorHandler(),
      HTTPRedirectHandler(),
      HTTPErrorProcessor()
    ]

    opener = OpenerDirector()
    opener.addheaders = [('User-agent', build_user_agent())]

    for handler in handlers:
      opener.add_handler(handler)

    return opener


  class GzipDecodedResponse(GZIP_BASE):
    """A file-like object to decode a response encoded with the
    gzip
    method, as described in RFC 1952.

    Largely copied from ``xmlrpclib``/``xmlrpc.client`` and
    modified
    to work for py2.4-py3
    """
    def __init__(self, response):
      # response doesn't support tell() and read(), required by
      # GzipFile
      if not gzip:
        raise SpeedtestHTTPError('HTTP response body is gzip
    encoded, '
                                 'but gzip support is not available')
      IO = BytesIO or StringIO
      self.io = IO()
      while 1:
        chunk = response.read(1024)
        if len(chunk) == 0:
          break
        self.io.write(chunk)
      self.io.seek(0)
      gzip.GzipFile.__init__(self, mode='rb', fileobj=self.io)

    def close(self):
      try:
        gzip.GzipFile.close(self)
      finally:
        self.io.close()


  def get_exception():
    """Helper function to work with py2.4-py3 for getting the
    current
    exception in a try/except block
    """
    return sys.exc_info()[1]


  def distance(origin, destination):
    """Determine distance between 2 sets of [lat,lon] in km"""
```

```
1662      lat1, lon1 = origin
          lat2, lon2 = destination
1664      radius = 6371  # km

1666      dlat = math.radians(lat2 - lat1)
          dlon = math.radians(lon2 - lon1)
1668      a = (math.sin(dlat / 2) * math.sin(dlat / 2) +
             math.cos(math.radians(lat1)) *
1670         math.cos(math.radians(lat2)) * math.sin(dlon / 2) *
             math.sin(dlon / 2))
1672      c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
          d = radius * c
1674
          return d
1676


1678    def build_user_agent():
          """Build a Mozilla/5.0 compatible User-Agent string"""
1680
          ua_tuple = (
1682        'Mozilla/5.0',
            '(%s; U; %s; en-us)' % (platform.platform(),
1684                    platform.architecture()[0]),
            'Python/%s' % platform.python_version(),
1686        '(KHTML, like Gecko)',
            'speedtest-cli/%s' % ___version___
1688      )
          user_agent = ' '.join(ua_tuple)
1690      printer('User-Agent: %s' % user_agent, debug=True)
          return user_agent
1692


1694    def build_request(url, data=None, headers=None, bump='0', secure
          =False):
          """Build a urllib2 request object
1696
          This function automatically adds a User-Agent header to all
          requests
1698
          """
1700
          if not headers:
1702        headers = {}

1704      if url[0] == ':':
            scheme = ('http', 'https')[bool(secure)]
1706        schemed_url = '%s%s' % (scheme, url)
          else:
1708        schemed_url = url

1710      if '?' in url:
            delim = '&'
1712      else:
            delim = '?'
1714
          # WHO YOU GONNA CALL? CACHE BUSTERS!
1716      final_url = '%s%sx=%s.%s' % (schemed_url, delim,
                      int(timeit.time.time() * 1000),
1718                  bump)

1720      headers.update({
```

```
      'Cache-Control': 'no-cache',
    })

    printer('%s %s' % (('GET', 'POST')[bool(data)], final_url),
        debug=True)

    return Request(final_url, data=data, headers=headers)


def catch_request(request, opener=None):
    """Helper function to catch common exceptions encountered when
    establishing a connection with a HTTP/HTTPS request

    """

    if opener:
      _open = opener.open
    else:
      _open = urlopen

    try:
      uh = _open(request)
      if request.get_full_url() != uh.geturl():
        printer('Redirected to %s' % uh.geturl(), debug=True)
      return uh, False
    except HTTP_ERRORS:
      e = get_exception()
      return None, e


def get_response_stream(response):
    """Helper function to return either a Gzip reader if
    ``Content-Encoding`` is ``gzip`` otherwise the response itself

    """

    try:
      getheader = response.headers.getheader
    except AttributeError:
      getheader = response.getheader

    if getheader('content-encoding') == 'gzip':
      return GzipDecodedResponse(response)

    return response


def get_attributes_by_tag_name(dom, tag_name):
    """Retrieve an attribute from an XML document and return it in
     a
    consistent format

    Only used with xml.dom.minidom, which is likely only to be
    used
    with python versions older than 2.5
    """
    elem = dom.getElementsByTagName(tag_name)[0]
    return dict(list(elem.attributes.items()))


def print_dots(shutdown_event):
    """Built in callback function used by Thread classes for
```

```
    printing
    status
1782  """
    def inner(current, total, start=False, end=False):
1784    if event_is_set(shutdown_event):
          return
1786
        sys.stdout.write('.')
1788    if current + 1 == total and end is True:
          sys.stdout.write('\n')
1790    sys.stdout.flush()
    return inner
1792


1794  def do_nothing(*args, **kwargs):
    pass
1796


1798  class HTTPDownloader(threading.Thread):
    """Thread class for retrieving a URL"""
1800
    def __init__(self, i, request, start, timeout, opener=None,
1802           shutdown_event=None):
      threading.Thread.__init__(self)
1804    self.request = request
      self.result = [0]
1806    self.starttime = start
      self.timeout = timeout
1808    self.i = i
      if opener:
1810      self._opener = opener.open
      else:
1812      self._opener = urlopen

1814    if shutdown_event:
        self._shutdown_event = shutdown_event
1816    else:
        self._shutdown_event = FakeShutdownEvent()
1818
    def run(self):
1820    try:
        if (timeit.default_timer() - self.starttime) <= self.
    timeout:
1822        f = self._opener(self.request)
          while (not event_is_set(self._shutdown_event) and
1824            (timeit.default_timer() - self.starttime) <=
                self.timeout):
1826          self.result.append(len(f.read(10240)))
            if self.result[-1] == 0:
1828          break
          f.close()
1830    except IOError:
        pass
1832    except HTTP_ERRORS:
        pass
1834


1836  class HTTPUploaderData(object):
    """File like object to improve cutting off the upload once the
     timeout
1838  has been reached
    """
```

```
1840
    def __init__(self, length, start, timeout, shutdown_event=None
    ):
1842    self.length = length
        self.start = start
1844    self.timeout = timeout

1846    if shutdown_event:
          self._shutdown_event = shutdown_event
1848    else:
          self._shutdown_event = FakeShutdownEvent()
1850
        self._data = None
1852
        self.total = [0]
1854
    def pre_allocate(self):
1856    chars = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        multiplier = int(round(int(self.length) / 36.0))
1858    IO = BytesIO or StringIO
        try:
1860      self._data = IO(
            ('content1=%s' %
1862          (chars * multiplier)[0:int(self.length) - 9]
            ).encode()
1864      )
        except MemoryError:
1866      raise SpeedtestCLIError(
            'Insufficient memory to pre-allocate upload data. Please
     '
1868        'use --no-pre-allocate'
          )
1870
    @property
1872def data(self):
        if not self._data:
1874      self.pre_allocate()
        return self._data
1876
    def read(self, n=10240):
1878    if ((timeit.default_timer() - self.start) <= self.timeout
    and
            not event_is_set(self._shutdown_event)):
1880      chunk = self.data.read(n)
          self.total.append(len(chunk))
1882      return chunk
        else:
1884      raise SpeedtestUploadTimeout()

1886def __len__(self):
        return self.length
1888


1890class HTTPUploader(threading.Thread):
    """Thread class for putting a URL"""
1892
    def __init__(self, i, request, start, size, timeout, opener=
    None,
1894         shutdown_event=None):
        threading.Thread.__init__(self)
1896    self.request = request
        self.request.data.start = self.starttime = start
```

```
1898        self.size = size
            self.result = 0
1900        self.timeout = timeout
            self.i = i
1902
            if opener:
1904          self._opener = opener.open
            else:
1906          self._opener = urlopen

1908        if shutdown_event:
              self._shutdown_event = shutdown_event
1910        else:
              self._shutdown_event = FakeShutdownEvent()
1912
          def run(self):
1914        request = self.request
            try:
1916          if ((timeit.default_timer() - self.starttime) <= self.
        timeout and
                not event_is_set(self._shutdown_event)):
1918            try:
                  f = self._opener(request)
1920            except TypeError:
                  # PY24 expects a string or buffer
1922              # This also causes issues with Ctrl-C, but we will
        concede
                  # for the moment that Ctrl-C on PY24 isn't immediate
1924              request = build_request(self.request.get_full_url(),
                            data=request.data.read(self.size))
1926              f = self._opener(request)
                f.read(11)
1928            f.close()
                self.result = sum(self.request.data.total)
1930          else:
                self.result = 0
1932        except (IOError, SpeedtestUploadTimeout):
              self.result = sum(self.request.data.total)
1934        except HTTP_ERRORS:
              self.result = 0
1936

1938    class SpeedtestResults(object):
          """Class for holding the results of a speedtest, including:
1940
          Download speed
1942      Upload speed
          Ping/Latency to test server
1944      Data about server that the test was run against

1946      Additionally this class can return a result data as a
          dictionary or CSV,
          as well as submit a POST of the result data to the speedtest.
          net API
1948      to get a share results image link.
          """
1950
          def __init__(self, download=0, upload=0, ping=0, server=None,
          client=None,
1952              opener=None, secure=False):
            self.download = download
1954        self.upload = upload
```

```
        self.ping = ping
1956    if server is None:
          self.server = {}
1958    else:
          self.server = server
1960    self.client = client or {}

1962    self._share = None
        # self.timestamp = '%sZ' % datetime.datetime.utcnow().
      isoformat()
1964    self.timestamp = '%sZ' % datetime.datetime.now().isoformat()
        self.bytes_received = 0
1966    self.bytes_sent = 0

1968    if opener:
          self._opener = opener
1970    else:
          self._opener = build_opener()
1972
        self._secure = secure
1974
      def __repr__(self):
1976    return repr(self.dict())

1978  def share(self):
        """POST data to the speedtest.net API to obtain a share
      results
1980    link
        """
1982
        if self._share:
1984      return self._share

1986    download = int(round(self.download / 1000.0, 0))
        ping = int(round(self.ping, 0))
1988    upload = int(round(self.upload / 1000.0, 0))

1990    # Build the request to send results back to speedtest.net
        # We use a list instead of a dict because the API expects
      parameters
1992    # in a certain order
        api_data = [
1994      'recommendedserverid=%s' % self.server['id'],
          'ping=%s' % ping,
1996      'screenresolution=',
          'promo=',
1998      'download=%s' % download,
          'screendpi=',
2000      'upload=%s' % upload,
          'testmethod=http',
2002      'hash=%s' % md5(('%s-%s-%s-%s' %
                    (ping, upload, download, '297aae72'))
2004                .encode()).hexdigest(),
          'touchscreen=none',
2006      'startmode=pingselect',
          'accuracy=1',
2008      'bytesreceived=%s' % self.bytes_received,
          'bytessent=%s' % self.bytes_sent,
2010      'serverid=%s' % self.server['id'],
        ]
2012
        headers = {'Referer': 'http://c.speedtest.net/flash/
```

```
    speedtest.swf'}
2014    request = build_request('://www.speedtest.net/api/api.php',
                  data='&'.join(api_data).encode(),
2016              headers=headers, secure=self._secure)
    f, e = catch_request(request, opener=self._opener)
2018    if e:
      raise ShareResultsConnectFailure(e)
2020
    response = f.read()
2022    code = f.code
    f.close()
2024
    if int(code) != 200:
2026      raise ShareResultsSubmitFailure('Could not submit results
    to '
                      'speedtest.net')
2028
    qsargs = parse_qs(response.decode())
2030    resultid = qsargs.get('resultid')
    if not resultid or len(resultid) != 1:
2032      raise ShareResultsSubmitFailure('Could not submit results
    to '
                      'speedtest.net')
2034
    self._share = 'http://www.speedtest.net/result/%s.png' %
    resultid[0]
2036
    return self._share
2038
  def dict(self):
2040    """Return dictionary of result data"""

2042    return {
      'download': self.download,
2044      'upload': self.upload,
      'ping': self.ping,
2046      'server': self.server,
      'timestamp': self.timestamp,
2048      'bytes_sent': self.bytes_sent,
      'bytes_received': self.bytes_received,
2050      'share': self._share,
      'client': self.client,
2052    }

2054  @staticmethod
  def csv_header(delimiter=','):
2056    """Return CSV Headers"""

2058    row = ['Server ID', 'Sponsor', 'Server Name', 'Timestamp', '
    Distance',
          'Ping', 'Download', 'Upload', 'Share', 'IP Address']
2060    out = StringIO()
    writer = csv.writer(out, delimiter=delimiter, lineterminator
    ='')
2062    writer.writerow([to_utf8(v) for v in row])
    return out.getvalue()
2064
  def csv(self, delimiter=','):
2066    """Return data in CSV format"""

2068    data = self.dict()
    out = StringIO()
```

```
2070        writer = csv.writer(out, delimiter=delimiter, lineterminator
          ='')
            row = [data['server']['id'], data['server']['sponsor'],
2072              data['server']['name'], data['timestamp'],
                  data['server']['d'], data['ping'], data['download'],
2074              data['upload'], self._share or '', self.client['ip']]
            writer.writerow([to_utf8(v) for v in row])
2076        return out.getvalue()

2078      def json(self, pretty=False):
            """Return data in JSON format"""
2080
            kwargs = {}
2082        if pretty:
              kwargs.update({
2084            'indent': 4,
                'sort_keys': True
2086          })
            return json.dumps(self.dict(), **kwargs)
2088


2090    class Speedtest(object):
          """Class for performing standard speedtest.net testing
          operations"""
2092
          def __init__(self, config=None, source_address=None, timeout
          =10,
2094               secure=False, shutdown_event=None):
            self.config = {}
2096
            self._source_address = source_address
2098        self._timeout = timeout
            self._opener = build_opener(source_address, timeout)
2100
            self._secure = secure
2102
            if shutdown_event:
2104          self._shutdown_event = shutdown_event
            else:
2106          self._shutdown_event = FakeShutdownEvent()

2108        self.get_config()
            if config is not None:
2110          self.config.update(config)

2112        self.servers = {}
            self.closest = []
2114        self._best = {}

2116        self.results = SpeedtestResults(
              client=self.config['client'],
2118          opener=self._opener,
              secure=secure,
2120        )

2122      @property
          def best(self):
2124         if not self._best:
               self.get_best_server()
2126       return self._best

2128      def get_config(self):
```

```
    """Download the speedtest.net configuration and return only
  the data
      we are interested in
      """

      headers = {}
      if gzip:
        headers['Accept-Encoding'] = 'gzip'
      request = build_request('://www.speedtest.net/speedtest-
  config.php',
                  headers=headers, secure=self._secure)
      uh, e = catch_request(request, opener=self._opener)
      if e:
        raise ConfigRetrievalError(e)
      configxml_list = []

      stream = get_response_stream(uh)

      while 1:
        try:
          configxml_list.append(stream.read(1024))
        except (OSError, EOFError):
          raise ConfigRetrievalError(get_exception())
        if len(configxml_list[-1]) == 0:
          break
      stream.close()
      uh.close()

      if int(uh.code) != 200:
        return None

      configxml = ''.encode().join(configxml_list)

      printer('Config XML:\n%s' % configxml, debug=True)

      try:
        try:
          root = ET.fromstring(configxml)
        except ET.ParseError:
          e = get_exception()
          raise SpeedtestConfigError(
            'Malformed speedtest.net configuration: %s' % e
          )
        server_config = root.find('server-config').attrib
        download = root.find('download').attrib
        upload = root.find('upload').attrib
        times = root.find('times').attrib
        client = root.find('client').attrib

      except AttributeError:
        try:
          root = DOM.parseString(configxml)
        except ExpatError:
          e = get_exception()
          raise SpeedtestConfigError(
            'Malformed speedtest.net configuration: %s' % e
          )
        server_config = get_attributes_by_tag_name(root, 'server-
  config')
        download = get_attributes_by_tag_name(root, 'download')
        upload = get_attributes_by_tag_name(root, 'upload')
        times = get_attributes_by_tag_name(root, 'times')
```

```
2188          client = get_attributes_by_tag_name(root, 'client')

2190        ignore_servers = [
              int(i) for i in server_config['ignoreids'].split(',') if i
2192        ]

2194        ratio = int(upload['ratio'])
            upload_max = int(upload['maxchunkcount'])
2196        up_sizes = [32768, 65536, 131072, 262144, 524288, 1048576,
        7340032]
            sizes = {
2198          'upload': up_sizes[ratio - 1:],
              'download': [350, 500, 750, 1000, 1500, 2000, 2500,
2200                 3000, 3500, 4000]
            }
2202
            size_count = len(sizes['upload'])
2204
            upload_count = int(math.ceil(upload_max / size_count))
2206
            counts = {
2208          'upload': upload_count,
              'download': int(download['threadsperurl'])
2210        }

2212        threads = {
              'upload': int(upload['threads']),
2214          'download': int(server_config['threadcount']) * 2
            }
2216
            length = {
2218          'upload': int(upload['testlength']),
              'download': int(download['testlength'])
2220        }

2222        self.config.update({
              'client': client,
2224          'ignore_servers': ignore_servers,
              'sizes': sizes,
2226          'counts': counts,
              'threads': threads,
2228          'length': length,
              'upload_max': upload_count * size_count
2230        })

2232        try:
              self.lat_lon = (float(client['lat']), float(client['lon'])
        )
2234        except ValueError:
              raise SpeedtestConfigError(
2236            'Unknown location: lat=%r lon=%r' %
                (client.get('lat'), client.get('lon'))
2238          )

2240        printer('Config:\n%r' % self.config, debug=True)

2242        return self.config

2244      def get_servers(self, servers=None, exclude=None):
            """Retrieve a the list of speedtest.net servers, optionally
        filtered
2246        to servers matching those specified in the ``servers``
```

```
    argument
      """
2248  if servers is None:
        servers = []
2250
      if exclude is None:
2252    exclude = []

2254  self.servers.clear()

2256  for server_list in (servers, exclude):
        for i, s in enumerate(server_list):
2258      try:
            server_list[i] = int(s)
2260      except ValueError:
            raise InvalidServerIDType(
2262          '%s is an invalid server type, must be int' % s
            )
2264
      urls = [
2266    '://www.speedtest.net/speedtest-servers-static.php',
        'http://c.speedtest.net/speedtest-servers-static.php',
2268    '://www.speedtest.net/speedtest-servers.php',
        'http://c.speedtest.net/speedtest-servers.php',
2270  ]

2272  headers = {}
      if gzip:
2274    headers['Accept-Encoding'] = 'gzip'

2276  errors = []
      for url in urls:
2278    try:
          request = build_request(
2280        '%s?threads=%s' % (url,
                               self.config['threads']['download']),
2282        headers=headers,
            secure=self._secure
2284      )
          uh, e = catch_request(request, opener=self._opener)
2286      if e:
            errors.append('%s' % e)
2288        raise ServersRetrievalError()

2290      stream = get_response_stream(uh)

2292      serversxml_list = []
          while 1:
2294        try:
              serversxml_list.append(stream.read(1024))
2296        except (OSError, EOFError):
              raise ServersRetrievalError(get_exception())
2298        if len(serversxml_list[-1]) == 0:
              break
2300
          stream.close()
2302      uh.close()

2304      if int(uh.code) != 200:
            raise ServersRetrievalError()
2306
          serversxml = ''.encode().join(serversxml_list)
```

```
2308
            printer('Servers XML:\n%s' % serversxml, debug=True)
2310
            try:
2312          try:
                try:
2314              root = ET.fromstring(serversxml)
                except ET.ParseError:
2316              e = get_exception()
                  raise SpeedtestServersError(
2318                'Malformed speedtest.net server list: %s' % e
                  )
2320            elements = etree_iter(root, 'server')
              except AttributeError:
2322            try:
                  root = DOM.parseString(serversxml)
2324            except ExpatError:
                  e = get_exception()
2326              raise SpeedtestServersError(
                    'Malformed speedtest.net server list: %s' % e
2328              )
                elements = root.getElementsByTagName('server')
2330        except (SyntaxError, xml.parsers.expat.ExpatError):
              raise ServersRetrievalError()
2332
            for server in elements:
2334          try:
                attrib = server.attrib
2336          except AttributeError:
                attrib = dict(list(server.attributes.items()))
2338
              if servers and int(attrib.get('id')) not in servers:
2340            continue

2342          if (int(attrib.get('id')) in self.config['
    ignore_servers']
                or int(attrib.get('id')) in exclude):
2344            continue

2346          try:
                d = distance(self.lat_lon,
2348                     (float(attrib.get('lat')),
                      float(attrib.get('lon'))))
2350          except Exception:
                continue
2352
              attrib['d'] = d
2354
              try:
2356            self.servers[d].append(attrib)
              except KeyError:
2358            self.servers[d] = [attrib]

2360        break

2362      except ServersRetrievalError:
            continue
2364
      if (servers or exclude) and not self.servers:
2366    raise NoMatchedServers()

2368  return self.servers
```

```
2370      def set_mini_server(self, server):
            """Instead of querying for a list of servers, set a link to
          a
2372        speedtest mini server
            """
2374
            urlparts = urlparse(server)
2376
            name, ext = os.path.splitext(urlparts[2])
2378        if ext:
              url = os.path.dirname(server)
2380        else:
              url = server
2382
            request = build_request(url)
2384        uh, e = catch_request(request, opener=self._opener)
            if e:
2386          raise SpeedtestMiniConnectFailure('Failed to connect to %s
          ' %
                                            server)
2388        else:
              text = uh.read()
2390          uh.close()

2392        extension = re.findall('upload_?[Ee]xtension: "([^"]+)"',
                               text.decode())
2394        if not extension:
              for ext in ['php', 'asp', 'aspx', 'jsp']:
2396            try:
                  f = self._opener.open(
2398                '%s/speedtest/upload.%s' % (url, ext)
                  )
2400            except Exception:
                  pass
2402            else:
                  data = f.read().strip().decode()
2404              if (f.code == 200 and
                      len(data.splitlines()) == 1 and
2406                  re.match('size=[0-9]', data)):
                    extension = [ext]
2408                break
            if not urlparts or not extension:
2410          raise InvalidSpeedtestMiniServer('Invalid Speedtest Mini
          Server: '
                                           '%s' % server)
2412
            self.servers = [{
2414          'sponsor': 'Speedtest Mini',
              'name': urlparts[1],
2416          'd': 0,
              'url': '%s/speedtest/upload.%s' % (url.rstrip('/'),
          extension[0]),
2418          'latency': 0,
              'id': 0
2420        }]

2422        return self.servers

2424      def get_closest_servers(self, limit=5):
            """Limit servers to the closest speedtest.net servers based
          on
```

```
2426        geographic distance
            """

2428
            if not self.servers:
2430          self.get_servers()

2432        for d in sorted(self.servers.keys()):
              for s in self.servers[d]:
2434            self.closest.append(s)
                if len(self.closest) == limit:
2436              break
              else:
2438            continue
              break
2440
            printer('Closest Servers:\n%r' % self.closest, debug=True)
2442        return self.closest

2444      def get_best_server(self, servers=None):
            """Perform a speedtest.net "ping" to determine which
          speedtest.net
2446        server has the lowest latency
            """

2448
            if not servers:
2450          if not self.closest:
                servers = self.get_closest_servers()
2452          servers = self.closest

2454        if self._source_address:
              source_address_tuple = (self._source_address, 0)
2456        else:
              source_address_tuple = None
2458
            user_agent = build_user_agent()
2460
            results = {}
2462        for server in servers:
              cum = []
2464           url = os.path.dirname(server['url'])
              stamp = int(timeit.time.time() * 1000)
2466          latency_url = '%s/latency.txt?x=%s' % (url, stamp)
              for i in range(0, 3):
2468            this_latency_url = '%s.%s' % (latency_url, i)
                printer('%s %s' % ('GET', this_latency_url),
2470                debug=True)
                urlparts = urlparse(latency_url)
2472            try:
                  if urlparts[0] == 'https':
2474                h = SpeedtestHTTPSConnection(
                      urlparts[1],
2476                  source_address=source_address_tuple
                    )
2478              else:
                    h = SpeedtestHTTPConnection(
2480                  urlparts[1],
                      source_address=source_address_tuple
2482                )
                  headers = {'User-Agent': user_agent}
2484               path = '%s?%s' % (urlparts[2], urlparts[4])
                  start = timeit.default_timer()
2486               h.request("GET", path, headers=headers)
```

```
            r = h.getresponse()
2488        total = (timeit.default_timer() - start)
          except HTTP_ERRORS:
2490        e = get_exception()
            printer('ERROR: %r' % e, debug=True)
2492        cum.append(3600)
            continue
2494
          text = r.read(9)
2496      if int(r.status) == 200 and text == 'test=test'.encode()
    :
            cum.append(total)
2498      else:
            cum.append(3600)
2500      h.close()

2502    avg = round((sum(cum) / 6) * 1000.0, 3)
        results[avg] = server
2504
      try:
2506    fastest = sorted(results.keys())[0]
      except IndexError:
2508    raise SpeedtestBestServerFailure('Unable to connect to
    servers to '
                                       'test latency.')
2510  best = results[fastest]
      best['latency'] = fastest
2512
      self.results.ping = fastest
2514  self.results.server = best

2516  self._best.update(best)
      printer('Best Server:\n%r' % best, debug=True)
2518  return best

2520  def download(self, callback=do_nothing, threads=None):
        """Test download speed against speedtest.net
2522
      A ``threads`` value of ``None`` will fall back to those
    dictated
2524    by the speedtest.net configuration
        """
2526
      urls = []
2528  for size in self.config['sizes']['download']:
        for _ in range(0, self.config['counts']['download']):
2530      urls.append('%s/random%sx%s.jpg' %
              (os.path.dirname(self.best['url']), size, size))
2532          # (os.path.dirname('http://test.telenor.net:8080/
    speedtest/upload.php'), size, size))

2534  request_count = len(urls)
      requests = []
2536  for i, url in enumerate(urls):
        requests.append(
2538      build_request(url, bump=i, secure=self._secure)
        )
2540
      max_threads = threads or self.config['threads']['download']
2542  in_flight = {'threads': 0}

2544  def producer(q, requests, request_count):
```

```
        for i, request in enumerate(requests):
          thread = HTTPDownloader(
            i,
            request,
            start,
            self.config['length']['download'],
            opener=self._opener,
            shutdown_event=self._shutdown_event
          )
          while in_flight['threads'] >= max_threads:
            timeit.time.sleep(0.001)
          thread.start()
          q.put(thread, True)
          in_flight['threads'] += 1
          callback(i, request_count, start=True)

      finished = []

      def consumer(q, request_count):
        _is_alive = thread_is_alive
        while len(finished) < request_count:
          thread = q.get(True)
          while _is_alive(thread):
            thread.join(timeout=0.001)
          in_flight['threads'] -= 1
          finished.append(sum(thread.result))
          callback(thread.i, request_count, end=True)

      q = Queue(max_threads)
      prod_thread = threading.Thread(target=producer,
                        args=(q, requests, request_count))
      cons_thread = threading.Thread(target=consumer,
                        args=(q, request_count))
      start = timeit.default_timer()
      prod_thread.start()
      cons_thread.start()
      _is_alive = thread_is_alive
      while _is_alive(prod_thread):
        prod_thread.join(timeout=0.001)
      while _is_alive(cons_thread):
        cons_thread.join(timeout=0.001)

      stop = timeit.default_timer()
      self.results.bytes_received = sum(finished)
      self.results.download = (
        (self.results.bytes_received / (stop - start)) * 8.0
      )
      if self.results.download > 100000:
        self.config['threads']['upload'] = 8
      return self.results.download

    def upload(self, callback=do_nothing, pre_allocate=True,
    threads=None):
      """Test upload speed against speedtest.net

      A ``threads`` value of ``None`` will fall back to those
    dictated
      by the speedtest.net configuration
      """

      sizes = []
```

```
      for size in self.config['sizes']['upload']:
        for _ in range(0, self.config['counts']['upload']):
          sizes.append(size)

      # request_count = len(sizes)
      request_count = self.config['upload_max']

      requests = []
      for i, size in enumerate(sizes):
        # We set ``0`` for ``start`` and handle setting the actual
        # ``start`` in ``HTTPUploader`` to get better measurements
        data = HTTPUploaderData(
          size,
          0,
          self.config['length']['upload'],
          shutdown_event=self._shutdown_event
        )
        if pre_allocate:
          data.pre_allocate()

        headers = {'Content-length': size}
        requests.append(
          (
            build_request(self.best['url'], data, secure=self.
_secure,
                          headers=headers),
            size
          )
        )

      max_threads = threads or self.config['threads']['upload']
      in_flight = {'threads': 0}

      def producer(q, requests, request_count):
        for i, request in enumerate(requests[:request_count]):
          thread = HTTPUploader(
            i,
            request[0],
            start,
            request[1],
            self.config['length']['upload'],
            opener=self._opener,
            shutdown_event=self._shutdown_event
          )
          while in_flight['threads'] >= max_threads:
            timeit.time.sleep(0.001)
          thread.start()
          q.put(thread, True)
          in_flight['threads'] += 1
          callback(i, request_count, start=True)

      finished = []

      def consumer(q, request_count):
        _is_alive = thread_is_alive
        while len(finished) < request_count:
          thread = q.get(True)
          while _is_alive(thread):
            thread.join(timeout=0.001)
          in_flight['threads'] -= 1
          finished.append(thread.result)
          callback(thread.i, request_count, end=True)
```

```
2666
        q = Queue(threads or self.config['threads']['upload'])
2668    prod_thread = threading.Thread(target=producer,
                                       args=(q, requests, request_count))
2670    cons_thread = threading.Thread(target=consumer,
                                       args=(q, request_count))
2672    start = timeit.default_timer()
        prod_thread.start()
2674    cons_thread.start()
        _is_alive = thread_is_alive
2676    while _is_alive(prod_thread):
          prod_thread.join(timeout=0.1)
2678    while _is_alive(cons_thread):
          cons_thread.join(timeout=0.1)
2680
        stop = timeit.default_timer()
2682    self.results.bytes_sent = sum(finished)
        self.results.upload = (
2684      (self.results.bytes_sent / (stop - start)) * 8.0
        )
2686    return self.results.upload


2688
    def ctrl_c(shutdown_event):
2690  """Catch Ctrl-C key sequence and set a SHUTDOWN_EVENT for our
      threaded
      operations
2692  """
      def inner(signum, frame):
2694    shutdown_event.set()
        printer('\nCancelling...', error=True)
2696    sys.exit(0)
      return inner
2698


2700  def version():
      """Print the version"""
2702
      printer('speedtest-cli %s' % __version__)
2704  printer('Python %s' % sys.version.replace('\n', ''))
      sys.exit(0)
2706


2708  def csv_header(delimiter=','):
      """Print the CSV Headers"""
2710
      printer(SpeedtestResults.csv_header(delimiter=delimiter))
2712  sys.exit(0)


2714
    def parse_args():
2716  """Function to handle building and parsing of command line
      arguments"""
      description = (
2718    'Command line interface for testing internet bandwidth using
       '
        'speedtest.net.\n'
2720    '
     ------------------------------------------------------------',
        '----------\n'
2722    'https://github.com/sivel/speedtest-cli')
```

```
2724    parser = ArgParser(description=description)
        # Give optparse.OptionParser an `add_argument` method for
2726    # compatibility with argparse.ArgumentParser
        try:
2728      parser.add_argument = parser.add_option
        except AttributeError:
2730      pass
        parser.add_argument('--no-download', dest='download', default=
        True,
2732                action='store_const', const=False,
                    help='Do not perform download test')
2734    parser.add_argument('--no-upload', dest='upload', default=True
        ,
                    action='store_const', const=False,
2736                help='Do not perform upload test')
        parser.add_argument('--single', default=False, action='
        store_true',
2738                help='Only use a single connection instead of '
                       'multiple. This simulates a typical file '
2740                   'transfer.')
        parser.add_argument('--bytes', dest='units', action='
        store_const',
2742                const=('byte', 8), default=('bit', 1),
                    help='Display values in bytes instead of bits. Does
        '
2744                   'not affect the image generated by --share, nor '
                       'output from --json or --csv')
2746    parser.add_argument('--share', action='store_true',
                    help='Generate and provide a URL to the speedtest.
        net '
2748                   'share results image, not displayed with --csv')
        parser.add_argument('--simple', action='store_true', default=
        False,
2750                help='Suppress verbose output, only show basic '
                       'information')
2752    parser.add_argument('--csv', action='store_true', default=
        False,
                    help='Suppress verbose output, only show basic '
2754                   'information in CSV format. Speeds listed in '
                       'bit/s and not affected by --bytes')
2756    parser.add_argument('--csv-delimiter', default=',', type=
        PARSER_TYPE_STR,
                    help='Single character delimiter to use in CSV '
2758                   'output. Default ","')
        parser.add_argument('--csv-header', action='store_true',
        default=False,
2760                help='Print CSV headers')
        parser.add_argument('--json', action='store_true', default=
        False,
2762                help='Suppress verbose output, only show basic '
                       'information in JSON format. Speeds listed in '
2764                   'bit/s and not affected by --bytes')
        parser.add_argument('--list', action='store_true',
2766                help='Display a list of speedtest.net servers '
                       'sorted by distance')
2768    parser.add_argument('--server', type=PARSER_TYPE_INT, action='
        append',
                    help='Specify a server ID to test against. Can be '
2770                   'supplied multiple times')
        parser.add_argument('--exclude', type=PARSER_TYPE_INT, action=
        'append',
2772                help='Exclude a server from selection. Can be '
```

```
                'supplied multiple times')
    parser.add_argument('--mini', help='URL of the Speedtest Mini
    server')
    parser.add_argument('--source', help='Source IP address to
    bind to')
    parser.add_argument('--timeout', default=10, type=
    PARSER_TYPE_FLOAT,
                help='HTTP timeout in seconds. Default 10')
    parser.add_argument('--secure', action='store_true',
                help='Use HTTPS instead of HTTP when communicating '
                    'with speedtest.net operated servers')
    parser.add_argument('--no-pre-allocate', dest='pre_allocate',
                action='store_const', default=True, const=False,
                help='Do not pre allocate upload data. Pre
    allocation '
                    'is enabled by default to improve upload '
                    'performance. To support systems with '
                    'insufficient memory, use this option to avoid a
    '
                    'MemoryError')
    parser.add_argument('--version', action='store_true',
                help='Show the version number and exit')
    parser.add_argument('--debug', action='store_true',
                help=ARG_SUPPRESS, default=ARG_SUPPRESS)

    options = parser.parse_args()
    if isinstance(options, tuple):
      args = options[0]
    else:
      args = options
    return args


def validate_optional_args(args):
    """Check if an argument was provided that depends on a module
    that may
    not be part of the Python standard library.

    If such an argument is supplied, and the module does not exist
    , exit
    with an error stating which module is missing.
    """
    optional_args = {
      'json': ('json/simplejson python module', json),
      'secure': ('SSL support', HTTPSConnection),
    }

    for arg, info in optional_args.items():
      if getattr(args, arg, False) and info[1] is None:
        raise SystemExit('%s is not installed. --%s is '
                'unavailable' % (info[0], arg))


def printer(string, quiet=False, debug=False, error=False, **
    kwargs):
    """Helper function print a string with various features"""

    if debug and not DEBUG:
      return

    if debug:
      if sys.stdout.isatty():
```

```
            out = '\033[1;30mDEBUG: %s\033[0m' % string
2828        else:
              out = 'DEBUG: %s' % string
2830      else:
            out = string
2832
          if error:
2834        kwargs['file'] = sys.stderr

2836      if not quiet:
            print_(out, **kwargs)
2838        # return str(out)


2840
        def shell():
2842      """Run the full speedtest.net test"""

2844      global DEBUG
          shutdown_event = threading.Event()
2846
          signal.signal(signal.SIGINT, ctrl_c(shutdown_event))
2848
          args = parse_args()
2850
          # Print the version and exit
2852      if args.version:
            version()
2854
          if not args.download and not args.upload:
2856        raise SpeedtestCLIError('Cannot supply both --no-download
          and '
                                  '--no-upload')
2858
          if len(args.csv_delimiter) != 1:
2860        raise SpeedtestCLIError('--csv-delimiter must be a single
          character')

2862      if args.csv_header:
            csv_header(args.csv_delimiter)
2864
          validate_optional_args(args)
2866
          debug = getattr(args, 'debug', False)
2868      if debug == 'SUPPRESSHELP':
            debug = False
2870      if debug:
            DEBUG = True
2872
          if args.simple or args.csv or args.json:
2874        quiet = True
          else:
2876        quiet = False

2878      if args.csv or args.json:
            machine_format = True
2880      else:
            machine_format = False
2882
          # Don't set a callback if we are running quietly
2884      if quiet or debug:
            callback = do_nothing
2886      else:
```

```
      callback = print_dots(shutdown_event)

    # printer('Retrieving speedtest.net configuration...', quiet)
    try:
      speedtest = Speedtest(
        source_address=args.source,
        timeout=args.timeout,
        secure=args.secure
      )
    except (ConfigRetrievalError,) + HTTP_ERRORS:
      printer('Cannot retrieve speedtest configuration', error=
    True)
      raise SpeedtestCLIError(get_exception())

    if args.list:
      try:
        speedtest.get_servers()
      except (ServersRetrievalError,) + HTTP_ERRORS:
        printer('Cannot retrieve speedtest server list', error=
    True)
        raise SpeedtestCLIError(get_exception())

      for _, servers in sorted(speedtest.servers.items()):
        for server in servers:
          line = ('%(id)5s) %(sponsor)s (%(name)s, %(country)s) '
              '[%(d)0.2f km]' % server)
          try:
            printer(line)
          except IOError:
            e = get_exception()
            if e.errno != errno.EPIPE:
              raise
      sys.exit(0)

    # printer('Testing from %(isp)s (%(ip)s)...' % speedtest.
    config['client'], quiet)

    if not args.mini:
      # printer('Retrieving speedtest.net server list...', quiet)
      try:
        args.server = speedtest.get_servers(servers=args.server,
    exclude=args.exclude)
        # args.server = speedtest.get_closest_servers(1)
      except NoMatchedServers:
        raise SpeedtestCLIError(
          'No matched servers: %s' %
          ', '.join('%s' % s for s in args.server)
        )
      except (ServersRetrievalError,) + HTTP_ERRORS:
        printer('Cannot retrieve speedtest server list', error=
    True)
        raise SpeedtestCLIError(get_exception())
      except InvalidServerIDType:
        raise SpeedtestCLIError(
          '%s is an invalid server type, must '
          'be an int' % ', '.join('%s' % s for s in args.server)
        )

      # if args.server and len(args.server) == 1:
      #     printer('Retrieving information for the selected
    server...', quiet)
      # else:
```

```
      #     printer('Selecting best server based on ping...',
    quiet)
2944    speedtest.get_best_server()
      # speedtest.get_closest_servers()
2946  elif args.mini:
      speedtest.get_best_server(speedtest.set_mini_server(args.
    mini))
2948
    results = speedtest.results
2950
    # printer('Hosted by %(sponsor)s (%(name)s) [%(d)0.2f km]: '
2952 #         '%(latency)s ms' % results.server, quiet)

2954  if args.download:
      # printer('Testing download speed', quiet,
2956  #         end=('', '\n')[bool(debug)])
      speedtest.download(
2958    callback=callback,
        threads=(None, 1)[args.single]
2960    )
      # printer('Download: %0.2f M%s/s' %
2962  #         ((results.download / 1000.0 / 1000.0) / args.units
    [1],
      #          args.units[0]),
2964  #         quiet)
    else:
2966    printer('Skipping download test', quiet)

2968  if args.upload:
      # printer('Testing upload speed', quiet,
2970  #         end=('', '\n')[bool(debug)])
      speedtest.upload(
2972    callback=callback,
        pre_allocate=args.pre_allocate,
2974    threads=(None, 1)[args.single]
      )
2976  # printer('Upload: %0.2f M%s/s' %
      #         ((results.upload / 1000.0 / 1000.0) / args.units
    [1],
2978  #          args.units[0]),
      #         quiet)
2980  else:
      printer('Skipping upload test', quiet)
2982
    printer('Results:\n%r' % results.dict(), debug=True)
2984
    if not args.simple and args.share:
2986    results.share()

2988  if args.simple:
      printer('Ping: %s ms\nDownload: %0.2f M%s/s\nUpload: %0.2f M
    %s/s' %
2990          (results.ping,
           (results.download / 1000.0 / 1000.0) / args.units[1],
2992         args.units[0],
           (results.upload / 1000.0 / 1000.0) / args.units[1],
2994         args.units[0]))
    elif args.csv:
2996    printer(results.csv(delimiter=args.csv_delimiter))
    elif args.json:
2998    printer(results.json())
```

```
3000        if args.share and not machine_format:
              printer('Share results: %s' % results.share())
3002
            utilities.convert_and_save_to_xlsx(str(results), "results/box/
            db0/ch1-ni-unshield.xlsx")
3004        # utilities.convert_and_save_to_xlsx(str(results), "results/f2
            /db0/ch1-coi-shield.xlsx")
            # utilities.convert_and_save_to_xlsx(str(results), "results/f2
            /db0/ch1-aci-shield.xlsx")
3006        # utilities.convert_and_save_to_xlsx(str(results), "results/f2
            /db0/ch6-coi-shield.xlsx")
            # utilities.convert_and_save_to_xlsx(str(results), "results/f2
            /db0/ch6-aci-shield.xlsx")
3008        # utilities.convert_and_save_to_xlsx(str(results), "results/f2
            /db0/ch11-coi-shield.xlsx")
            # utilities.convert_and_save_to_xlsx(str(results), "results/f2
            /db0/ch11-aci-shield.xlsx")
3010
            # utilities.convert_and_save_to_xlsx(str(results), "results/f2
            /db20/ch11-aci-shield.xlsx")
3012
          def main():
3014        try:
              shell()
3016        except KeyboardInterrupt:
              printer('\nCancelling...', error=True)
3018        except (SpeedtestException, SystemExit):
              e = get_exception()
3020          # Ignore a successful exit, or argparse exit
              if getattr(e, 'code', 1) not in (0, 2):
3022            msg = '%s' % e
                if not msg:
3024              msg = '%r' % e
                raise SystemExit('ERROR: %s' % msg)
3026
3028
          if __name__ == '__main__':
3030        main()
```

Bibliography

[1] Kemp, S. 'Internet use in 2024.' (), [Online]. Available: https://datareportal.com/reports/digital-2024-deep-dive-the-state-of-internet-adoption.

[2] www.statista.com. 'Www.statista.com.' (), [Online]. Available: https://datareportal.com/reports/digital-2024-deep-dive-the-state-of-internet-adoption.

[3] Crow, B., Widjaja, I., Kim, J. and Sakai, P., 'Ieee 802.11 wireless local area networks,' *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[4] wikipedia. 'Internet standard.' (), [Online]. Available: https://en.wikipedia.org/wiki/Internet_Standard.

[5] wikipedia. 'Ieee 802.11bn.' (), [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.11bn.

[6] K., P. and P., K., 'Evolution and impact of wi-fi technology and applications: A historical perspective.,' 2020.

[7] Wenbo, Y., Quanyu, W. and Zhenwei, G., 'Smart home implementation based on internet and wifi technology,' in *2015 34th Chinese Control Conference (CCC)*, 2015, pp. 9072–9077.

[8] Schwab, K., *The fourth industrial revolution*, eng, Genève, 2016.

[9] De Pellegrini, F., Miorandi, D., Vitturi, S. and Zanella, A., 'On the use of wireless networks at low level of factory automation systems,' *IEEE Transactions on Industrial Informatics*, vol. 2, no. 2, pp. 129–143, 2006.

[10] Villegas, E. G., Lopez-Aguilera, E., Vidal, R. and Paradells, J., 'Effect of adjacent-channel interference in ieee 802.11 wlans,' in *2007 2nd International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, 2007, pp. 118–125.

[11] Chounos, K., Keranidis, S., Korakis, T. and Tassiulas, L., 'Characterizing the impact of interference through spectral analysis on commercial 802.11 devices,' in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[12] Zubow, A. and Sombrutzki, R., 'Adjacent channel interference in ieee 802.11n,' in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, 2012, pp. 1163–1168.

[13] Robinson, J., Papagiannaki, K., Diot, C., Guo, X. and Krishnamurthy, L., 'Experimenting with a multi-radio mesh networking testbed,' in *1st workshop on Wireless Network Measurements*, 2005.

[14] Angelakis, V., Papadakis, S., Siris, V. and Traganitis, A., 'Adjacent channel interference in 802.11a: Modeling and testbed validation,' in *2008 IEEE Radio and Wireless Symposium*, 2008, pp. 591–594.

[15] wikipedia. 'List of wlan channels.' (), [Online]. Available: https://en.wikipedia.org/wiki/List_of_WLAN_channels#2.4_GHz_(802.11b/g/n/ax).

[16] Wang, X. and Yang, K., 'A real-life experimental investigation of cross interference between wifi and zigbee in indoor environment,' in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2017, pp. 598–603.

[17] cisco. 'Network-killer.' (), [Online]. Available: https://www.cisco.com/c/en/us/td/docs/wireless/controller/technotes/8-7/b_wireless_high_client_density_design_guide.html.

[18] Souppaya, M. and Scarfone, K., 'Guidelines for securing wireless local area networks (wlans),' *NIST Special Publication*, vol. 800, p. 153, 2012.

[19] Fuxjager, P., Valerio, D. and Ricciato, F., 'The myth of non-overlapping channels: Interference measurements in ieee 802.11,' in *2007 Fourth Annual Conference on Wireless on Demand Network Systems and Services*, 2007, pp. 1–8.

[20] Zeng, Y., Pathak, P. H. and Mohapatra, P., 'A first look at 802.11ac in action: Energy efficiency and interference characterization,' in *2014 IFIP Networking Conference*, 2014, pp. 1–9.

[21] Karthik. 'How does speedtest custom work?' (), [Online]. Available: https://www.ookla.com/about.

[22] Ookla LLC., a. Z. D. c. 'Ookla's speedtest® methodology.' (), [Online]. Available: https://www.ookla.com/resources/guides/speedtest-methodology.

[23] sivel. 'Speedtest-cli github.' (), [Online]. Available: https://github.com/sivel/speedtest-cli.

[24] heimgard. 'Heimgard-smart-mesh.' (), [Online]. Available: https://heimgard.com/support/brukerveiledninger/heimgard-smart-mesh.

[25] raspberrypi. 'Raspberrypi.' (), [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/.

[26] Raspbian. 'Raspbian.' (), [Online]. Available: https://www.raspberrypi.com/software/raspberry-pi-desktop/.

[27] Project, L. W. D. 'About iw.' (), [Online]. Available: https://wireless.docs.kernel.org/en/latest/en/users/documentation/iw.html.

[28] Zhu, K. 'Kun's github repo.' (), [Online]. Available: https://github.com/Fangt-ing/speedtest.git.

[29] Bobbitt, Z. ‘When to use mean vs. median (with examples).’ (), [Online]. Available: https://www.statology.org/when-to-use-mean-vs-median/.

[30] Orn, A. ‘Means and medians: When to use which.’ (), [Online]. Available: https://www.statology.org/when-to-use-mean-vs-median/.

[31] iperf.fr. ‘Iperf - the ultimate speed test tool for tcp, udp and sctp.’ (), [Online]. Available: https://iperf.fr/iperf-doc.php.

[32] Zhu, K. ‘Iperf github.’ (), [Online]. Available: https://github.com/Fangting/iperf-test.git.