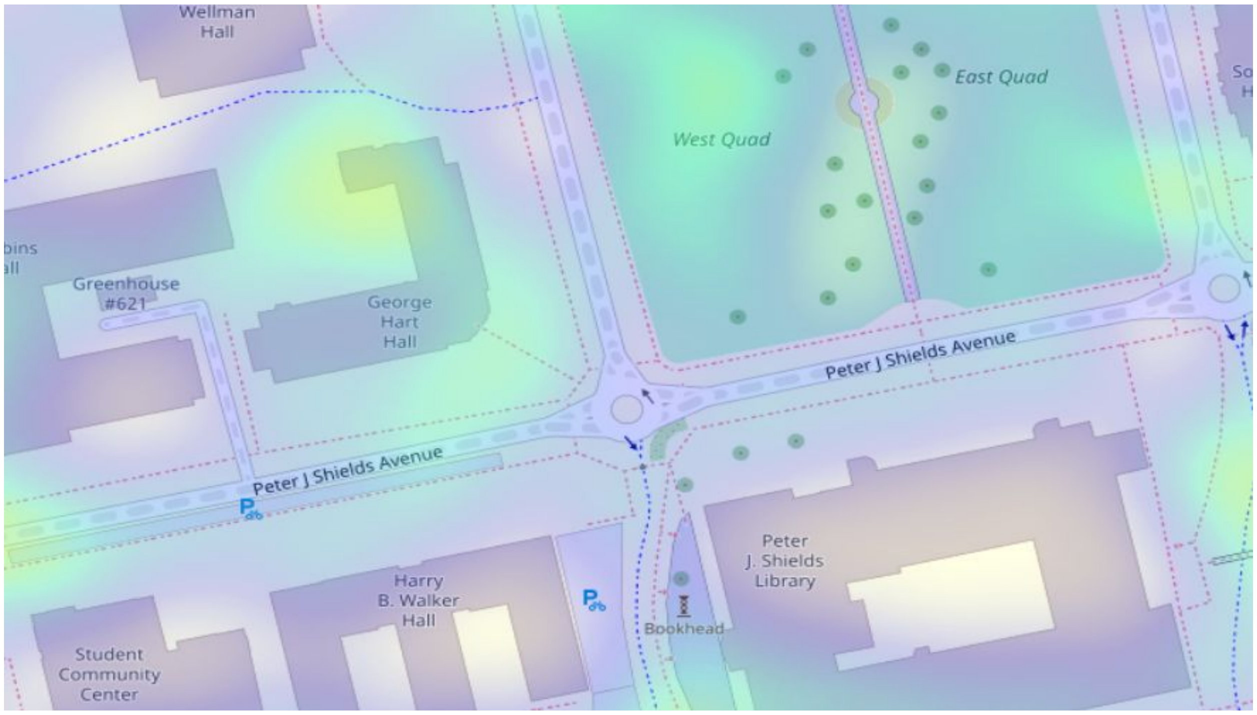
ECS 152c - Professor Chen-Nee Chuah - Spring 2019

# Assessment and Visualization of WiFi Bandwidth on UC Davis Campus

Markus Hankins, Jesus Vizcarra



## Abstract

Students and faculty at UC Davis have the good fortune of ubiquitous wireless Internet access via the Eduroam network. During day to day use, it is not necessary to be conscious of available bandwidth or signal strength of the network as it is adequate enough to meet people's needs. Yet, as undergraduate students studying communication networks, having access to such a complex wide area WiFi network provides an ample opportunity to intimately learn about its technical aspects. We have chosen to focus on measuring its bandwidth and how that correlates with signal strength at the client. Through collection of site survey data across a large sample area of campus, we construct a more cohesive understanding of the network through statistical analysis and data visualization via a heat map. This paper presents our design and implementation of our project.

## Introduction

In deciding to study the campus-wide Eduroam WiFi network, it is obvious there are a vast number of areas one could focus on. The proximity of access points in relation to each other and how that manifests to co-channel interference, especially on the 2.4 GHz band, was one possibility. Another idea was to conduct a site survey of the campus in order to discover other WiFi networks besides Eduroam, and consequently conduct a security audit of them. While these ideas were quite interesting, we decided to settle our focus on certain aspects of a user's experience as it pertains to network availability and signal quality.

Upon settling on these two metrics, we began to specifically consider bandwidth and its potential correlation to signal strength. We felt these were novel aspects that are not usually presented over a large coverage area. Often we encounter bandwidth measurements as they relate to a specific point of presence on a network. Somewhat similarly, projects like WiGLE WiFi network mapping focus on the presence of access points, their location, and related details like security, signal strength, frequency, etc. Therefore, we perceived there to be a gap in the usual data analytics of WiFi networks as they do not measure bandwidth, and, when they do, it is not data that can be extrapolated over a network the size of Eduroam.

After choosing topical areas to study, we began the process of researching methods with which to take measurements and analyze the data. The best system for our purposes consisted of developing our own scanner software whereby we had near full control of the metrics to measure and how to proceed with their subsequent analysis. Consequently, data collection is done at specific points on campus by recording a test site's GPS location. Thus, all tests are performed outdoors, as indoor tests would prove to be difficult in getting accurate GPS coordinates. Ultimately, these tests will lead to a presentation of our statistical analysis and visualization of the data.

## Related Work

We considered other software packages and services with which to gather, analyze, and present data. We considered commercial WiFi analytics software packages such as VisiWave's Site Survey [1] and WiFi Heat Map by SolarWinds. [2] VisiWave's solution has the ability to gather data a point at a time by conducting systematic walks through a coverage area. Importantly, these points are tied to a GPS location, which later can be overlaid onto a map. Somewhat in relation, both VisiWave and

SolarWinds can be tied into a network's access points (AP) where they can then pull data in real time to construct and represent an accurate model of the network along with a chosen set of metrics. Purchasing this software is quite expensive and require AP access we do not have. While they both offer an evaluation period of their products, the complexity of their software is beyond the scope of this project. Additionally, as students, it was important to not abstract away the technical challenges inherent in devising methods to study a network.

The second area of work to consider was whether we could leverage an existing speed test solution in our own software. There are a number of speed test websites and smartphone applications available. Some examples are Ookla's Speedtest.net (iOS, Android, and website), DSLReports.com, and Speakeasy.net (hosted by MegaPath). The terms of service of the latter two restrict usage to their respective website interfaces. Ookla's terms of service is such that they allow virtually unlimited usage as long as their software and associated data is not used for commercial purposes. Their service enables users to keep a record of their results, and, when using their smartphone app, it also records the test's GPS coordinates. However, because an aspect of the project involves correlating the signal strength of the associated access point (AP) at test time, we found that Ookla makes no available record of the WiFi network or any of its details. While unfortunate for our needs, this is most likely due to privacy concerns.

Initially, it was thought this information could be obtained from an app like WiGLE or some other WiFi monitoring app in conjunction with Ookla's app. However, we found these apps reported available APs and not the currently associated one. Even if one existed, there would be additional complications trying to associate the bandwidth test results with the logs from the WiFi logging app.

## Overview of Solution

Below is a high-level overview of our solution. In successive sections, each point's design/implementation, notable setbacks and workarounds will be documented.

- GPS receivers
- Scanner software to collect speed test data, WiFi metrics, GPS coordinates
- Pair of Linux laptops to run scanner software
- Conduct tests within chosen sample area of campus
- Data analysis and representation using Python and R
- Visualize bandwidth as a heat map

## GPS Receivers

A critical component of this project is to correlate the bandwidth tests and other network metrics to a specific point on a campus map. Since the entirety of the tests are outside, we will use GPS receivers to record each test site’s coordinates. Initially, Professor Chuah provided us with one GPS unit to use. The device worked well in helping us to gain an understanding of how GPS signalling works, the types of messages the unit provides, and the requisite software to talk to it.

It became apparent we needed another GPS unit when we decided to use two Linux laptops to collect data, which would decrease overall scan time. From a prior lab involving wardriving, one of us had experience using an Android smartphone to provide GPS to a laptop using an app called BlueNMEA. [6] It provides GPS data on a TCP port. The port is then forwarded to a Linux laptop using Android Debug Bridge (ADB), a common development tool on the Android platform.

For the sake of consistency between the two laptops, it was decided to use an Android phone and BlueNMEA on both setups. There are advantages in having a uniform setup between the two machines, and an important one is that it would be easier to troubleshoot any GPS issues in the future.

### GPS - NMEA 0138

A quick note about the format of the GPS data coming from BlueNMEA. As the app’s name suggests, it sends NMEA sentences over a TCP port. Specifically, NMEA 0138, although we are unsure which revision of 0138 it is. A sample of NMEA 0138 sentences from the Wikipedia page are included below. [7]

```
$GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,19,13,28,070,17,26,23,252,,04,14,186,14*79
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092750.000,A,5321.6802,N,00630.3372,W,0.02,31.66,280511,,,A*43
$GPGGA,092751.000,5321.6802,N,00630.3371,W,1,8,1.03,61.7,M,55.3,M,,*75
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,16,13,28,070,17,26,23,252,,04,14,186,15*77
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092751.000,A,5321.6802,N,00630.3371,W,0.06,31.66,280511,,,A*45
```

After the `$GP` prefix is the type, where the `GGA` type holds the current latitude and longitude. This was the sentence needed by our software.

## Scanner Software

We decided to code software that would run on a Linux laptop with a connected GPS device. By using Python, we found modules with which to interact with the Linux wireless subsystem, perform speed tests using Ookla's services, collect GPS coordinates, and log all the information together in one place. There was some trial and error involved in composing our software stack, so the following summarizes a couple key issues encountered while refining our software stack.

## First Approach

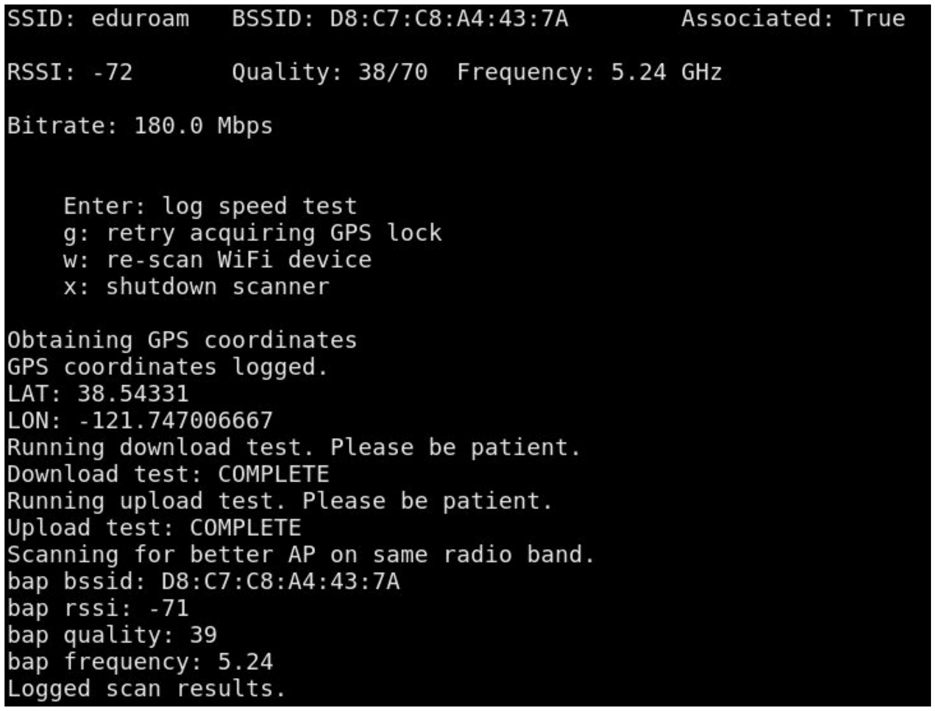
It was necessary to use a couple different Python modules to interface with the Linux wireless subsystem. One of them was PyRIC. [8] It was used to get the current associated BSSID, frequency, bit rate, RSSI, etc. Initial tests were promising until the field tests began. Intermittently, it would throw exceptions when the wireless card would fluctuate between being associated with an AP and not. Some troubleshooting revealed a possible cause was that the module attempted to use a persistent netlink socket connection to the kernel's NL80211 subsystem. [9] When the state of the NIC changed, the module threw exceptions. PyRIC was replaced with Python WiFi. [10]

## Second Approach

After resolving the wireless module issue, results from our field tests were looking good. We began to scan portions of campus, and it was not until we tried mapping our data that we realized there was a problem. We were using GPSd to interface with the TCP port forwarded from the Android phone. [11] It is a commonly used package providing a uniform interface when accessing a variety of GPS devices connected via various means. Occasionally, the GPS coordinate values would be stale. The end effect was that multiple tests had the same coordinates, leading to spurious results on the heat map. We removed GPSd from the stack, and devised a system to open a TCP socket and read the NMEA sentences. They were then parsed using the PyNMEA2 module. [12] [13]

## Final Approach

Once we worked through the aforementioned issues, we settled on a robust software stack.



```
SSID: eduroam   BSSID: D8:C7:C8:A4:43:7A        Associated: True

RSSI: -72       Quality: 38/70  Frequency: 5.24 GHz

Bitrate: 180.0 Mbps


    Enter: log speed test
    g: retry acquiring GPS lock
    w: re-scan WiFi device
    x: shutdown scanner

Obtaining GPS coordinates
GPS coordinates logged.
LAT: 38.54331
LON: -121.747006667
Running download test. Please be patient.
Download test: COMPLETE
Running upload test. Please be patient.
Upload test: COMPLETE
Scanning for better AP on same radio band.
bap bssid: D8:C7:C8:A4:43:7A
bap rssi: -71
bap quality: 39
bap frequency: 5.24
Logged scan results.
```

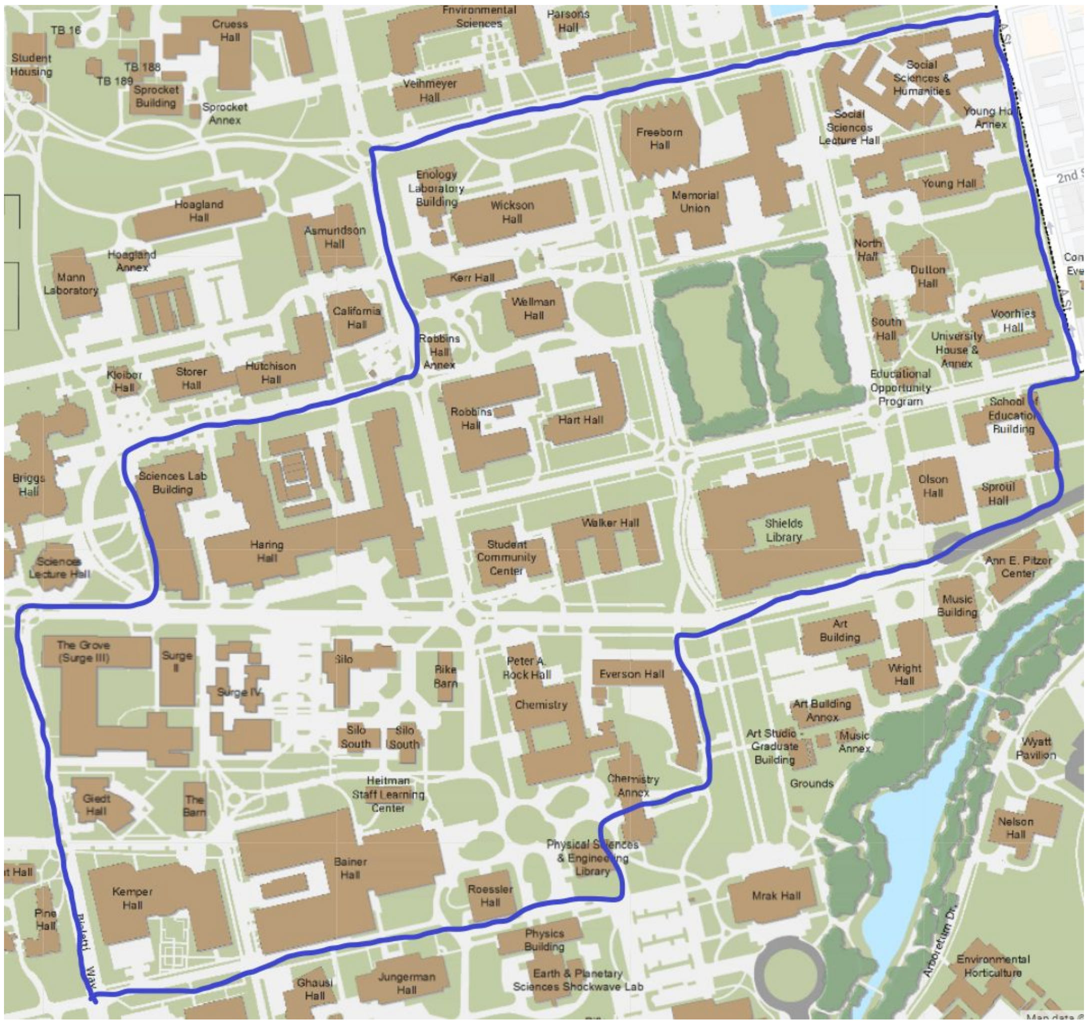
The screenshot above shows the scanner in action after a test run. The wireless modules iwlib and Python WiFi were used to gather metrics about current network conditions. [14] The options to refresh GPS were in case we noticed stale coordinates in the logs. There was also an option to refresh the WiFi data in case we wanted to try to induce a hand-off to a better AP. As mentioned before, a TCP socket would be open to BlueNMEA on the smartphone where NMEA sentences would be captured. The Pynmea2 module would then parse the sentences looking for `GGA` types containing the latitude and longitude coordinates. Once a position was fixed, then if connected to an AP, a speed test was run against Ookla's servers using the speedtest-cli Python module. [15] For the download and upload tests, each one was comprised of three runs which were then averaged. This was necessary to smooth out occasional spikes or dips in a test result. After the speed tests, the scanner would look for other APs on the same band as the currently associated AP. If there was a better AP with a higher RSSI, then an attempt was made to associate with it.

## Linux Laptops

Since two different laptops were being used to collect the data, we wanted the devices to be as similar as possible to ensure consistent test results. Fortunately, we both owned Lenovo laptops (13" and 14") that happened to use the Intel 7265 and 8265 cards, respectively. [16][17] They both use two antennas that are routed around the laptop screen. As the two screens are about the same size, the reception characteristics should be nearly the same. They both support 802.11ac.

## Conduct Tests

We were now ready to conduct our tests across the chosen sample area of campus. Scanning the entirety of UC Davis campus is not only time consuming, but also unnecessary as only a subset of campus has the most foot traffic traversing it. Below is a screenshot of a campus map where the area enclosed by the blue line comprises our sample space. Initially, we considered selecting a wider area that captured places such as the ARC and the dorms. However, after several tests runs in other areas, along with the setback of the GPS issue discussed earlier, we constrained the sample space. This was primarily due to the cumulative time the tests would take and the onset of unfavorable weather conditions.



### Plan of Attack

We devised a plan to systematically conduct tests at points within the sample space. The following considerations were used:

- Perform tests at locations where people walk
- Locations are roughly separated by 50 paces (approx. 1m per pace)
- Attempt to connect to the best AP in the area
- 5 GHz band is preferred
- Upload and download tests are averaged over three runs a piece

Inadvertently, some tests were well within 50 paces of each other. This caused issues when constructing the heat map, which we will go into more detail later. While conducting the scans, the campus map (shown above) was opened along side the terminal running the scanner. It was used as a guide in determining how to divide up the area into manageable chunks, selecting start and stop points within a chunk, and choosing paths to follow.

## Analysis of Results

Now we present the results of our analysis, as well as data representations using graphs. All of the statistical analysis was done using Python and R. The heat map itself was constructed using Leaflet.heat. [18]

### Initial Statistics

<u>Download Speeds</u>

Max: 154 Mbps

Min: 0 Mbps

<u>Upload Speeds</u>

Max: 193 Mbps

Min: 0 Mbps

<u>Basic Service Set Identifiers (BSSID)</u>

Unique: 170

Total: 239

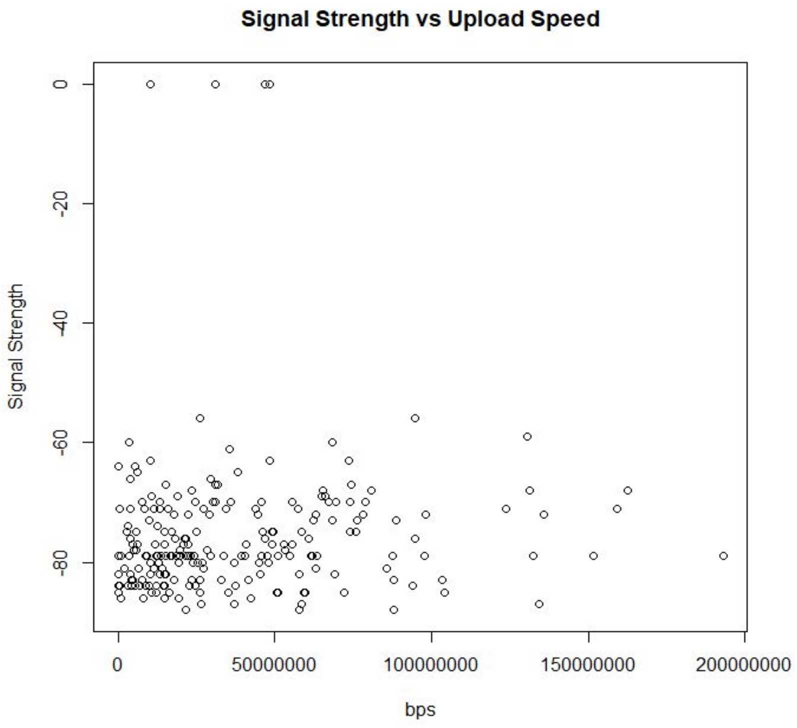
Adjusted[1]: 217

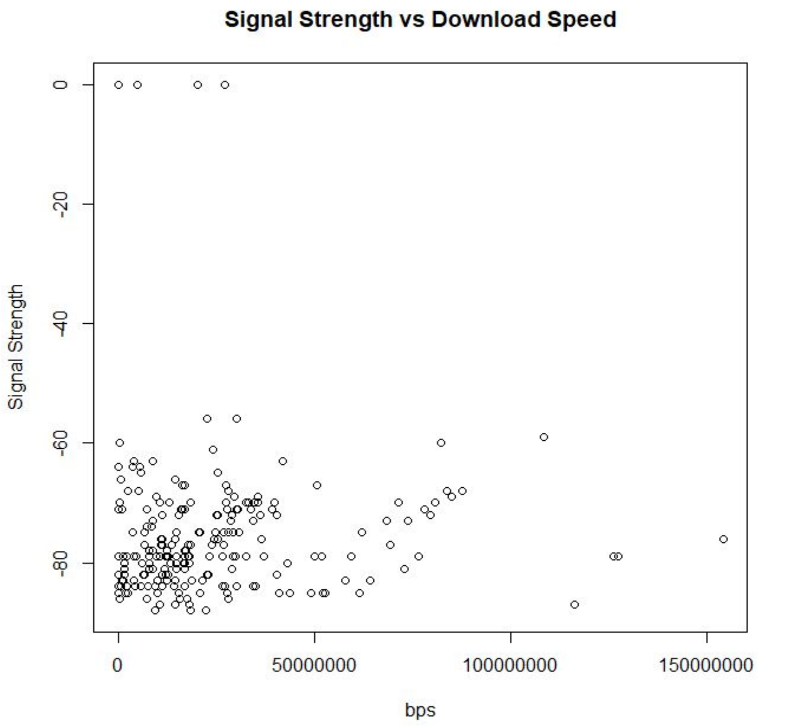
<u>Tests with no Internet access</u>: 4

<u>Tests on 5 GHz band:</u> 90.2%

[1] Number of BSSIDs after trimming test sites that were too close together.

## Correlation of Bandwidth and Signal Strength



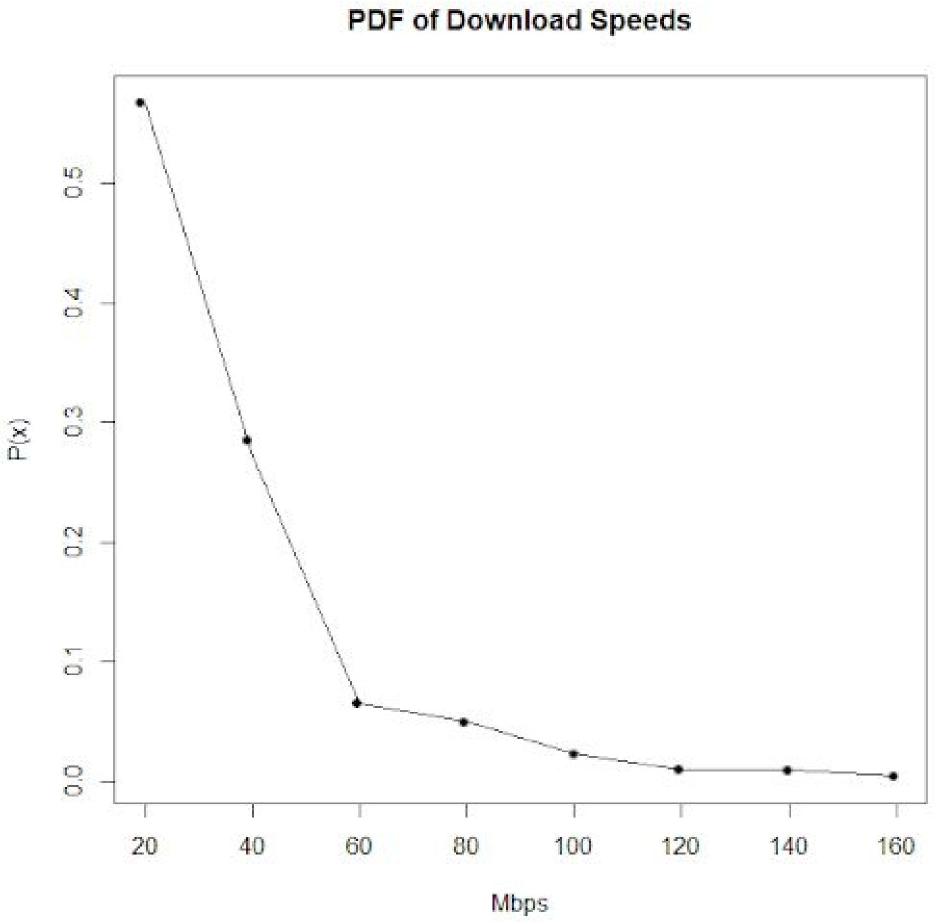


Signal strength as RSSI (higher is better). Outliers at zero had no Internet connectivity.

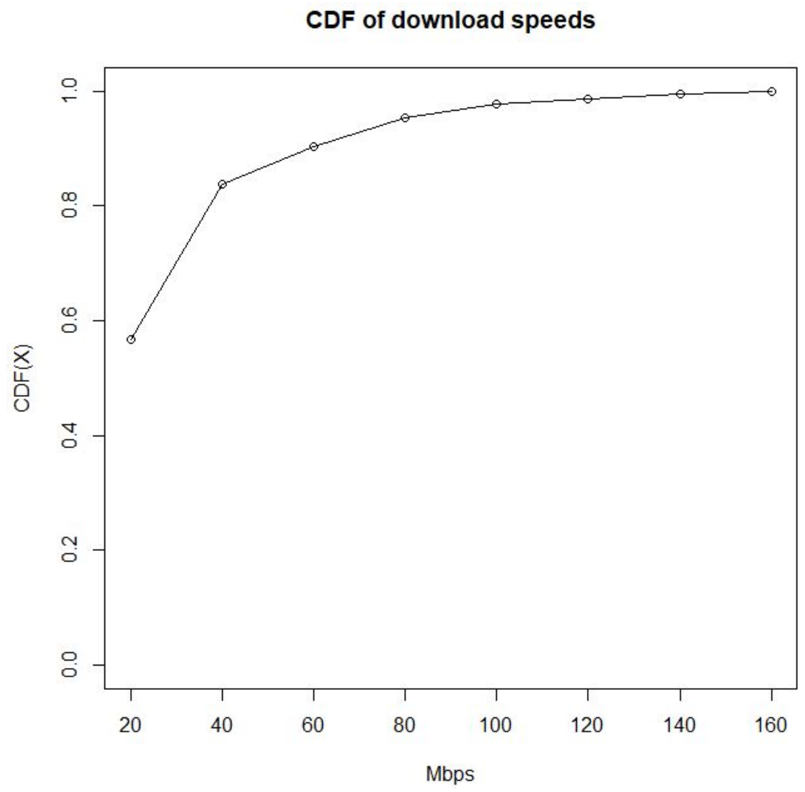
One of the motivations for this project was to see what correlation, if any, exists between signal strength (RSSI) and bandwidth (bps). As can be seen from both the upload and download tests, there is none. The points are just too spread out for most transfer speeds. Ironically, though, it should be noted that the fastest transfer speeds for the download tests occurred with rather low RSSI values.

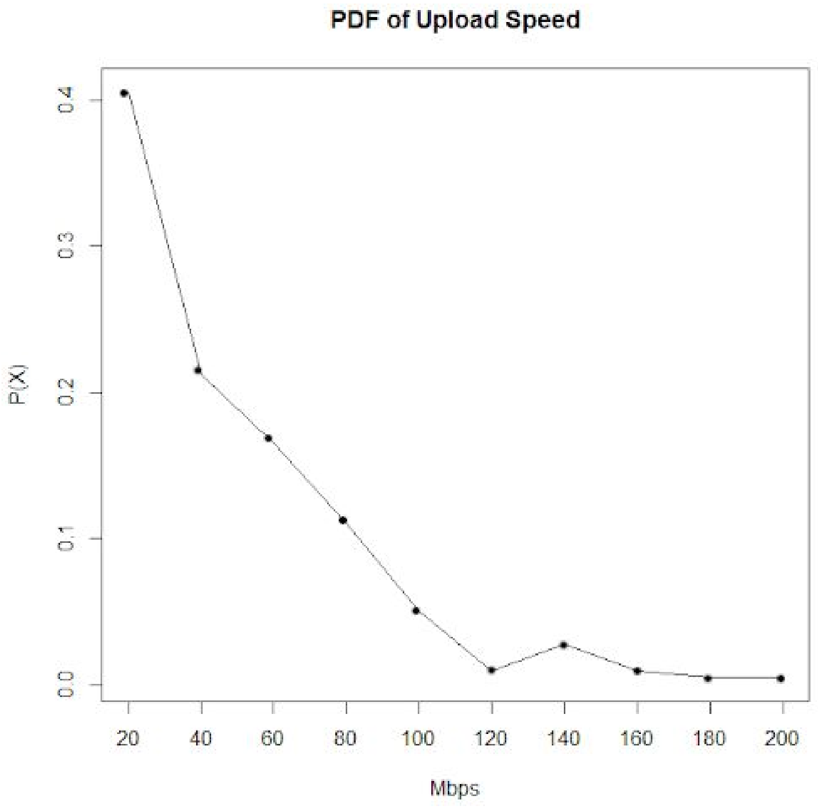
## Statistical Analysis of Transfer Speeds

The next four graphs represent the probabilities of the download and upload speeds recorded in our tests. Even though the values are discrete, they were combined into buckets of 20 Mbps each to lessen the noise in the graphs due to the size of each data set.

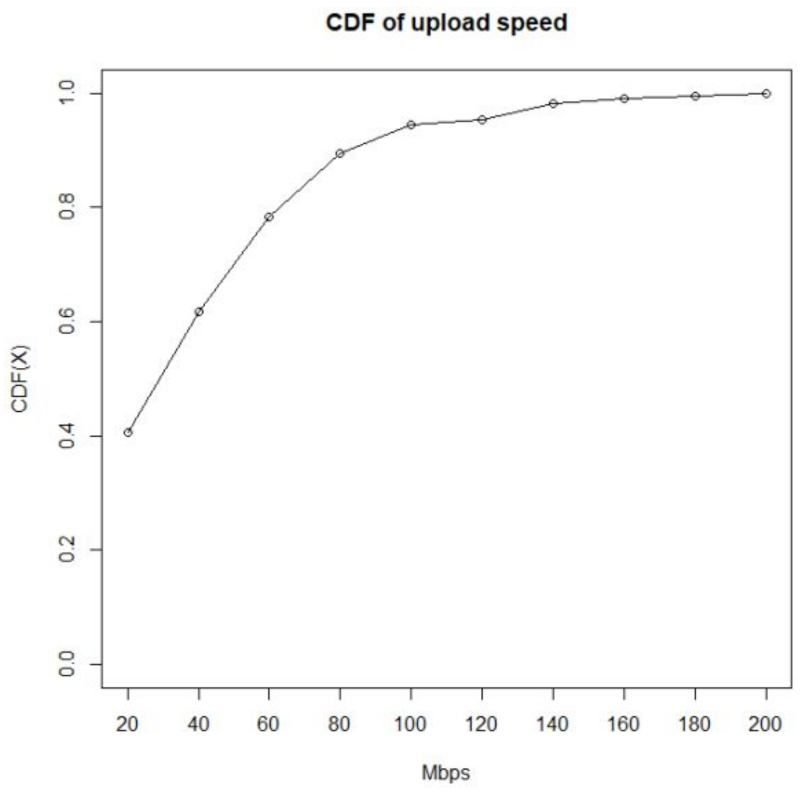


Given a random point within our sample space, the probability of getting at most 20 Mbps is nearly 60%. The next two steps along the distribution see a rather sharp drop in bandwidth. Keep in mind the download data rates when looking over the upload graphs.


CDF of download speeds
CDF(X)
Mbps
0.0
0.2
0.4
0.6
0.8
1.0
20
40
60
80
100
120
140
160

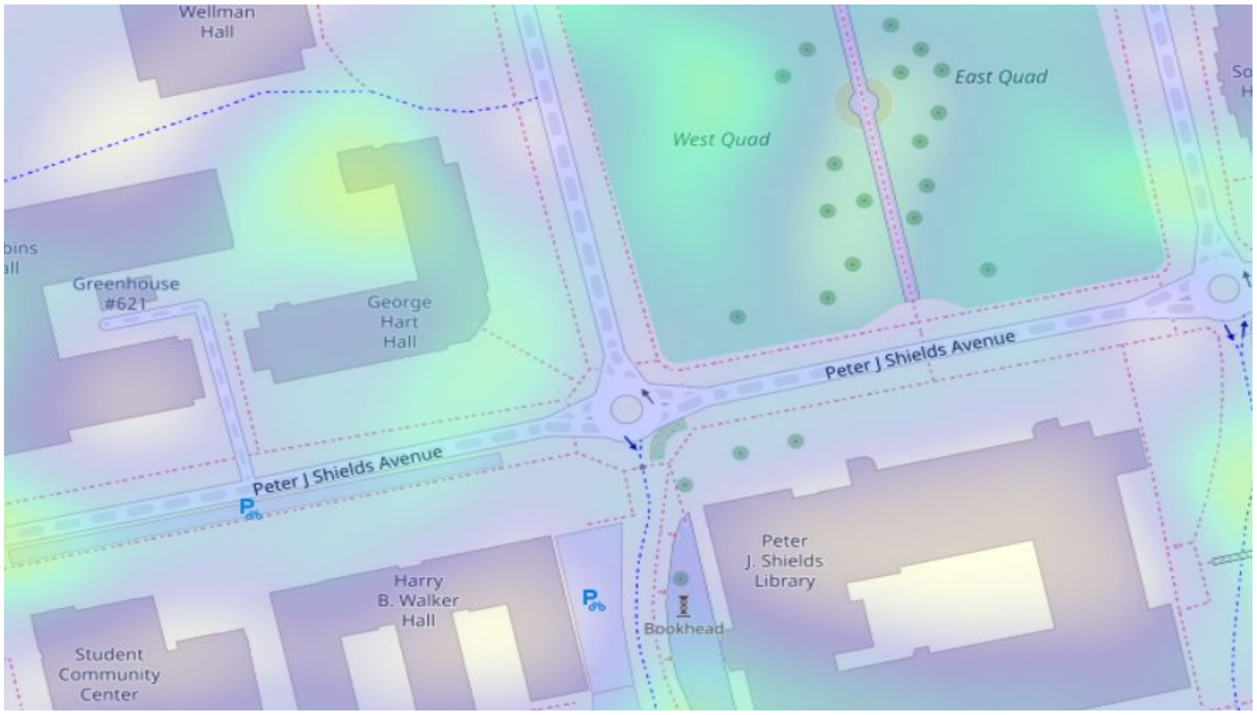

PDF of Upload Speed
P(X)
Mbps
0.0
0.1
0.2
0.3
0.4
20
40
60
80
100
120
140
160
180
200

In analyzing the upload results, it was surprising to see such asymmetry between the upload and download speeds. The first several buckets of the PDF do not have nearly as severe a drop compared to PDF for downloads. Initially, it was thought there might be an issue with the design of the scanner's speed tests. We experimented with various amounts of delay to see if that had any effect on the results, which they did not. Running the tests in total isolation produced results with the same pattern. This gave us confidence that our speed test design was sound.



After taking some time to consider the asymmetric patterns in the results, we formed a hypothesis. It is likely that the network administrators decided to implement a quality of service (QoS) policy to regulate the allocation of download bandwidth amongst all the clients. In general, most people are consuming content as opposed to creating it and serving it out onto the Internet. Seeing as how there are a large number of clients connected to Eduroam, it would make sense to constrain download bandwidth allocation.

## Heat Map Visualization



[2]: Blue: [0,30), Turquoise: [30,60), Yellow: [60,90), Orange: [90,120), Red: [120, 150)   units: Mbps

The heat map is another motivating factor of this project. A screenshot of the map is picture above. As mentioned earlier in the report, the heat map was constructed using Leaflet.heat. It is a Javascript utility that leverages OpenStreetMap.org for the mapping backend. [19] The maps are interactive and meant to be browsed within a web browser, so please use the following links to view them. Please note: the heat intensity changes with the zoom level. With these maps, the appropriate intensity levels are at maximum zoom, which is their default setting.

Download bandwidth heat map:
https://csiflabs.cs.ucdavis.edu/~mhankins/downloadHeatmap.html

Upload bandwidth heat map:
https://csiflabs.cs.ucdavis.edu/~mhankins/uploadHeatmap.html

The Leaflet.heat software takes in GPS coordinates along with an intensity level, which is the transfer speed rounded to the nearest whole number. First, the software must be initialized with a maximum point intensity value that corresponds with the max transfer speed. Secondly, a radius is set for each coordinate.

[2] Bandwidth data collected in May 2019.

Coordinates with overlapping radii gave the false impression that the bandwidth was higher than it actually was. However, if the radius was too low, then every point would be isolated from each other. As a result we picked a radius that would minimize interference while not eliminating it all together. It was at this stage where we discovered the bad GPS data issue mentioned earlier. The points were too close together, so they gave spurious results on the heat map. Next, a level of blur was needed to create an effect of cohesion between the points. The heat consistently spreads from a point the length of its radius. Therefore, at the edges we observed a fringing of the heat dissipation. To minimize this fringing, we experimented with blurring them. The end result is a nice blending between the coordinates. Lastly, we had to calculate color gradient intervals that accurately represented our statistical findings. After some effort, we finally arrived at a scheme that closely resembles the upload and download PDFs.

## Conclusion

After completing our scans and analyzing the results, we arrived at, and in one case unexpectedly discovered, some interesting insights. While we had suspicions of what the outcome might be for the bandwidth and signal strength correlation tests, the others were a little surprising. Below is a summary of those findings.

### Bandwidth and Signal Strength Correlation

Even though the results were not altogether unsurprising, there were cases of high transfer speeds (e.g. 100 Mbps) achieved with rather poor signal strength (e.g. RSSI -75). This is interesting because this is not often what people think should happen. In considering cases like the example given above, we looked at where the results occurred. From what we could gather, the APs at those locations most likely had very few other clients with which to contend. Not even necessarily on the same AP, but also on the same frequency. A benefit of 5 GHz WiFi is the abundance of channels. Of which, the Eduroam network design minimizes co-channel interference by properly spacing out APs using the same frequency. To the original point, though, a distant AP can yield higher than expected transfer speeds if the number of clients associated to the same AP is low.

### Consistent Asymmetry in Upload vs Download Speeds

Given any random point within the sample space, the probability is high that upload speeds are faster than download speeds. When this pattern started to emerge, it was thought there might be an issue with the design and timing of the speed tests. Various amounts of delay were introduced between the download and upload tests, with the thought that there might be contention issues with other clients concerning exponential back off with our test attempting to send some much data at a time. However, changing the timings yielded no change over several areas. What is interesting is that a few locations did not experience this issue at all. This gave us confidence in the design and timing of our tests. A likely explanation is that some kind of quality of service (QoS) algorithm was put into place by the network administrators. Clients are more apt to download large amounts of data (e.g. watching HD YouTube videos), so implementing policy constraints to affect how the download bandwidth is allocated makes perfect sense.

**Download Speeds Fall within 20 Mbps**

Choosing a random a point within the sample space has a near 60% probability that the download speed is at most 20 Mbps. This is not altogether surprising when taking that last observation's asymmetry conclusion into account. Still, though, it was expected that the speeds would be higher given the better upload speeds, overall.

**Prevalence of 5 GHz Access Points**

Within the sample space, there is a 90% chance a client associates with an AP using the 5 GHz band. This finding caught us by surprise. At the outset, the thinking was that 2.4 GHz would be predominant due to its ability to travel longer distances. However, it suffers from issues of co-channel interference in areas dense with other APs using the same band. Using the 5 GHz band allows for many more channels with less of the aforementioned interference.

**Current Reflections and Future Improvements**

The effort expended in the project's design was outweighed by the effort of its implementation, which includes conducting the tests. While an expected level of effort was made to ensure sound and accurate measurements, it was dwarfed by what it took to take those measurements themselves. The last several runs with the scanner took about 10 hours to complete. Before, when the bad GPS data issue was occurring, we spent nearly 3 hours taking measurements. Needless to say, this is not exactly practical. Although, it lead to quite interesting findings. Ideally, a network administrator would use software like Visiwave to gain an accurate, and real-time, model of the Eduroam network.

In the future, this project could be conducted in a more efficient manner. Putting aside using software like Visiwave, et al., the data could also be crowd-sourced. An app developed for iOS and Android devices could periodically run the speed tests, record the location (using GPS and other identifying criteria), and send the results to a server. Despite developing a high quality app, there is always a challenge in getting people to install new software, so there would need to be some incentive. One possibility would be to enter the participants in a raffle where the prize is a coveted gift card (e.g. Amazon, iTunes, etc.). The coverage area will be smaller, but it would likely include more relevant areas, especially those inside buildings.

## References

[*]: WiFi Bandwidth Scanner https://github.com/microHertz/WiFi-Bandwidth-Scanner

[1]: VisiWave Site Survey https://visiwave.com/

[2]: SolarWinds Heat Map
https://www.solarwinds.com/network-performance-monitor/use-cases/wifi-heat-map

[3]: Ookla Speed Test https://www.speedtest.net/

[4]: DSLReports.com Speed Test http://www.dslreports.com/speedtest

[5]: Speakeasy.net Speed Test https://www.speakeasy.net/speedtest/

[6]: BlueNMEA https://max.kellermann.name/projects/blue-nmea/

[7]: NMEA 0138 https://en.wikipedia.org/wiki/NMEA_0183

[8]: PyRIC https://github.com/wraith-wireless/pyric

[9]: Netlink http://man7.org/linux/man-pages/man7/netlink.7.html

[10]: Python WiFi http://pythonwifi.tuxfamily.org

[11]: GPS service daemon https://gpsd.gitlab.io/gpsd/index.html

[12]: NMEA http://www.gpsinformation.org/dale/nmea.htm

[13]: PyNMEA2 https://github.com/Knio/pynmea2

[14]: iwlib https://github.com/nathan-hoad/python-iwlib

[15]: speedtest-cli https://github.com/sivel/speedtest-cli

[16]: Intel Dual Band Wireless-AC 7265 card
https://ark.intel.com/content/www/us/en/ark/products/83635/intel-dual-band-wireless-ac-7265.html

[17]: Intel Dual Band Wireless-AC 8265 card
https://ark.intel.com/content/www/us/en/ark/products/94150/intel-dual-band-wireless-ac-8265.html

[18]: Leaflet.heat https://github.com/Leaflet/Leaflet.heat

[19]: OpenStreetMap Foundation https://wiki.osmfoundation.org/wiki/Main_Page