UiO : Department of Informatics
University of Oslo

Group creation in a collaborative P2P channel allocation protocol

Identifying connected groups of access points

Hans Jørgen Furre Nygårdshaug

Master's Thesis, Autumn 2017

Group creation in a collaborative P2P channel allocation protocol

Hans Jørgen Furre Nygårdshaug

26th October 2017

Abstract

Contents

List of Figures

Chapter 1

Introduction

The presence of Wireless LANs is ever expanding, and by 2020 it is esitmated that we have 20 billion connected devices [4]. According to the Wi-Fi Alliance, households alone will be home to over 10 billion devices that are able to connect to the home router [1]. Even while introducing more and more devices to a home in the shape of smart TVs, lights, and audio systems, at the same time a user expects the Wireless LAN connection to deliver optimal speeds to meet our time's demand for universal wireless access. Not only is the number of devices growing, but our traffic patterns has changed. In private homes continous streams of high bitrate video and audio is highly common, and it is not unusual that these flows happen simultaneously in a family home. Also as network-attached storage has become available to consumers, and the popularity of cloud storage is also increasing, the new traffic that is transmitted over Wireless LANs often also includes data transmissions and backups that may happen in a background process. While the physical layer (PHY) traditionally has been upgraded to meet the new demands in bitrate (e.g. fiber over wire, and 802.11ac) Wireless LAN connections struggle under the heavy impact traffic intereference which can not be solved by increasing the physical datarate capacity.

1.1 Motivation

Wireless LAN is deployed in almost all corporate buildings and residencies in the western world, and increasingly also in the rest of the world. The use of these networks used to be limited to laptops that generated small amounts of data, but now the range of devices includes smart-phones, network storage devices, even in some cases servers. The deployment of Wireless LAN and the infrastructure has not changed much over the years to match the new

demands and increased traffic, and in most places coverage is still the main concern, while Quality of Service (QoS) comes second. There has been done much research on how centralized controllers can benefit the deployment of enterprise networks ([8], [7] and [9]), but in this thesis we will address the issues related to deployment to Wireless LANs in residiential areas. Customers who are subscribed to high data rate service level agreements often find they can only receive a comparable data rate over wired LAN. When the Wireless LAN networks in their home is constantly underperforming, it is not unusual for an unknowing customer to upgrade the data rate of their agreement - to no effect. This leads to customers being largely frustrated with their Internet service providers, even though they are not at fault. In many scenarios a customer service representative or a tech-savvy customer may manually switch the operating channel for a wireless access point. If this has no effect a customer might be encouraged to get a router which can transmit a more powerful signal. While this might prove to be a quick fix for the customer who was resourceful enough to deal with the problem, it in turn may trigger a chain reaction of even stronger interference levels for the rest of the inhabitants in the surrounding area. In this thesis we are going to look at a possible way for Wireless LAN access points in residential homes to organize themselves in groups and communicate with each other to collaborate on channel assignment.

1.2 Problem definition

There are 3 non-overlapping channels on the 2.4GHz spectrum that 802.11 Wireless LANs uses. One of the usual ways of selecting a channel is done by letting an access point sense which channel has the lowest interference levels. When channels are selected this selfishly it is very unlikely that the distribution of channels in a confined area (e.g. an apartment block) becomes optimal. Ideally all access points would be configured so the channel allocation is optimal for an entire area. This can be realized with a centralized controller, but in residential WiFi networks there are by default no centralized controller. We will explore how wireless access points can organize themselves in confined groups that collaborate on selecting an optimal channel distribution for the entire group to maximize the efficiency of wireless networks.

1.3 Method

To be able to see if we can form groups in a way that creates clusters of nearby nodes we need to get some data to perform calculation on. We will both create syntethic data and use real world location data of access points to evaluate the performance of the group creations. Then we will take a look at possible solutions to let access points communicate with each other without any bootstrapping or previous association. Then we will consider the problems and challenges that has to be overcome in the process of creating and deploying an architecture as suggested in the thesis.

Chapter 2

Background

We will briefly introduce the relevant aspects of wireless technology and a selection of important concepts from the 802.11 standard, both on the link and physical layer.

2.1 Common challenges

There are some challenges with wireless technologies that are harder to overcome than in wired transmission technologies like Ethernet.

The first step to achieve a succesful transmission is making sure radio B is within radio A's transmit range. The transmit range is decided by the power which the signal is transmitted at, the antenna gain, and the surrounding environment. If there are a lot of solid obstacles, like walls and ceilings, the signal is likely to have a very compromised range.

Even if the surrounding environment is mostly open space, the wireless signal becomes subject of attenuation, which is a physical property of electromagnic waves that weakens the signal the longer it has travelled through a medium. When this medium is air, we refer to the phenomenon as Free Space Path Loss (FSPL). Attentuation limits the transmit range of a radio, and to transmit further it has to increase its transmission power.

2.1.1 Collisions in wireless technology

If there are other nodes nearby that are within radio B's sensing range that sends at the same time as A, radio B experiences radio frequency interference, and thus may not be able to correctly decode the signal of A. In 802.3 Ethernet this is graciously handled by collision detection in the CSMA/CD protocol. As each node can hear everyone on else on a cabled medium, and

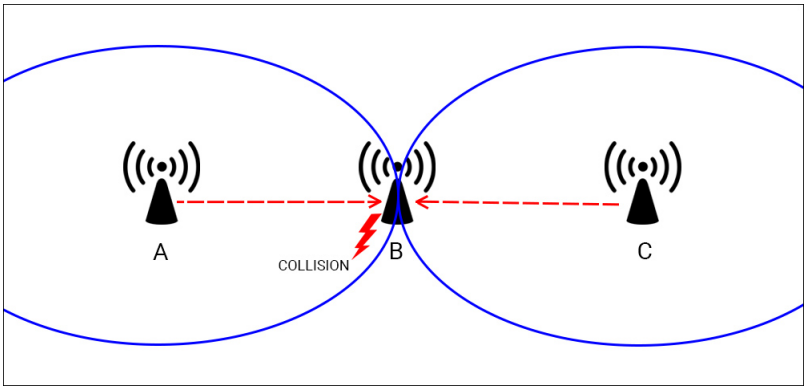

Figure 2.1: A and C sending to node B unknowingly at the same time, resulting in a collision

listening while transmitting is generally not a problem, a node can retransmit if a collision is detected. In wireless techhnologies it is not equally easy to listen while transmitting, and collision detection may be impossible because of the hidden terminal problem.

The hidden terminal problem

The hidden terminal problem is one of the major challenges for wireless technologies. When node A transmits a message to node B, it may not be able to sense what is going on on the opposite side of node B. If a node C transmits at the same time, this signal may not enter node A's sensing range, and hence go undetected by, even though a collision has happened near node B. This is illustrated in figure 2.1 where both A and C has unknowingly sent messages at the same time, and B has not been able to decode any messages because of the colliding messages.

2.2 MAC Layer

The 802.11 MAC layer implements the Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) protocol to control access to the medium. The CSMA/CA protocol is designed to operate in an entirely distributed fashion, where all stations connected to the same BSS operate on the same frequency without coordinated timeslots. As suggested by the name of the protocol, there are two basic operations in the CSMA/CA protocol: **Carrier Sensing (CS)** and **Collision Avoidance (CA)**.

2.2.1 Carrier Sense

In 802.11, carrier sensing (CS) is done in two ways

- **Physical carrier sensing** handled by the physical layer (PHY) as Clear Channel Assessment (CCA), which we will talk about in the PHY subsection.
- **Virtual carrier sensing** is a MAC layer mechanism in place to limit the number of times a node has to check the physical radio. When a valid 802.11 frame is decoded for a listening node, it can read the duration of the transmission from the MAC header. The frame with a duration is called a Network Allocation Vector (NAV). When a NAV is received the channel is marked as busy and the node will back off for the duration of the NAV.

2.2.2 Collision Avoidance

CSMA/CA attempts to avoid collisions in a network layout that includes hidden terminals. **Request To Send/Clear To Send** (RTS/CTS) is the function that allows CSMA/CA to some degree avoid the hidden terminal problem. By letting a node first ask the receiver if it is available for transmission (RTS), it prevents the node from sending the payload frame unless it receives Clear To Send (CTS) frame from the receiver first. The other mechanisms for collision avoidance are:

- **Interframe spacing** (IFS) is the amount of time the channel has to be idle before a sender can compete for channel access. To give priority to certain frame types, different types of frames can have different types of interframe spacing. The type of IFS is usually prefixed with the letter of the frame type. Organized by relative interval length, the differents IFS are:
  - Short IFS (SIFS), before ACK, RTS, CTS.
  - DCF Mode IFS (PIFS), before RTS frames (or DCF data frames if RTS/CTS is disabled)
- **Exponential backoff** is what prevents two competing nodes from sending at the same time. When the channel is clear for DIFS time, a node has to wait another random number of milliseconds before transmitting. This is called backoff. The amount of backoff is randomly chosen from a contention window (CW). The contention window has a low start size, called $CW_{\min}$. A node draws a backoff time in the range

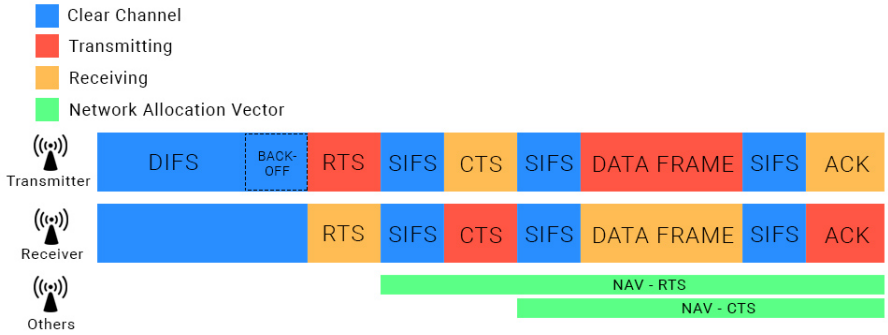

Figure 2.2: The timeline one frame transmission cycle in DCF mode

$(0, 2^n * CW_{\min})$, where n is the number of times the transmission has failed, beginning at $n = 0$, and $CW_{\min} < CW_{\max}$.

2.2.3 Distributed Coordination Function

The distributed coordination function is the main mode of operation in the 801.11 MAC layer, and is supposed to provide fair and reliable transmission for all nodes on the same network. Figure 2.2 shows the frame exchange that happens. The transmitter has to wait DIFS time, before drawing a number from the contenion window and backing off that amount. As no other transmissions has begun during this time and the channel is still clear, the transmitter sends out an RTS frame. When received by the receiver it waits SIFS time before transmitting a CTS frame. The transmitter then sends his data frame, and waits for the ACK that indicates a succesful transmission. The RTS/CTS mechanisms introduces extra overhead, and is sometimes turned off. The size of the payload and the number of stations on the network decides whether it is beneficial to have on or not [2].

2.3 PHY Layer

The physical layer in 801.11 is also divided in two sublayers. The upper sublayer is the Physical Layer Convergence Procedure (PLCP), responsible for CCA and acting as a common interface for MAC layer drivers. The lower sublayer is the Physical Medium Dependent (PMD) which is responsible for modulation and directly interfaces with the radio. It is responsible for transmitting the complete frames, and receive and decode incoming frames.

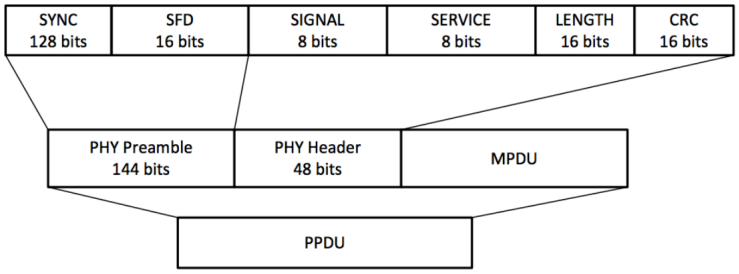

Figure 2.3: DSSS PHY PPDU format from IEEE Std 802.11-2016

2.3.1 PLCP Protocol Data Unit

The physical layer convergence procedure creates PLCP Protocol Data Units (PPDUs), that consists of 3 parts. The preamble, the header and the frame from the mac layer called MAC Service Data Unit (MSDU), see figure 2.3. The frame structure has a long and short format, and changes a little bit for High Rate DSSS (HR-DSSS), but contains mostly the same fields. As they are relevant for the thesis, we will briefly summarize the fields in the preamble and header:

- **Sync** bits are used to acquire the signal and synchronize timing.
- **SFD** stands for Start Frame Delimeter and is there to indicate the start of a frame.
- **Signal** the modulation type used to encode the MPDU and the data rate it is sent with.
- **Service** Reserved for future use.
- **Length** 16-bit fields that indicates the amount of time (in microseconds it will take to transmit the MPDU).
- **CRC** is the cyclic redundancy check that protects the fields signal, service and length.

2.3.2 Clear channel assessment

The PLCP layer also provides clear channel assessment. The purpose of clear channel assessment is to give information to the MAC layer if the carrier is available for transmission. There are primarily two ways the physical layer does CCA.

- CCA-ED (CCA-Energy Detect) detects signals that can not be decoded as a 801.11 frame, but is a disturbing signal on the channel. If the CCA-ED value exceeds a threshold, for instance 20 dB, then the CCA shall be indicated as busy by issuing a `PHY-CCA.indication(BUSY)` to the MAC layer.
- CCA-CS (CCA Carrier/Sense) detects a valid 801.11 frame and can properly decode the header fields of a valid PPDU frame. The channel gets marked as busy for as long as the length field in the PPDU header specifies, even if the obvserved signal is weaker than the ED threshold.

2.4 Radio Frequency Interference

Radio Frequency Interference (RF-interference) is the result of two or more signals being transmitted on the same frquency at the same time. A receiver will have problems deciding which parts of the signals belongs to which transmitter, and the signal may altered to the extent that bits are changed or misrepresented. As the 2.4 GHz band that 802.11 utilizes is a part of the ISM band, channel noise or interference can come from sources such as microwaves, bluetooth devices or other WiFi entities.

2.4.1 Impact in 802.11

If the PPDU header gets corrupted by an interfering signal and can not be decoded , the PLCP layer issues a `PHY-RXEND.indication(CarrierLost)` to the MAC layer. According to CSMA/CA the station then has to wait EIFS (Extended Interframe Spacing) time before it can transmit a new frame. EIFS is defined as `ACK transmission time + SIFS + DIFS`. This is because the station that received the corrupt frame have no idea if any neighbouring station, received it correctly, and is about to transmit an ACK-frame. Additionally to waiting the minimum EIFS time, the station also has to wait for an idle channel indication from the PLCP layer again.

2.4.2 Countermeasures

Several countermeasures to limit the impact of RF-interference has been suggested. On the mac layer there is frame aggregation with individual headers. Frame aggregation in 802.11n is a technique that wraps several payloads under the same header and send them all when channel access is granted. This can improve the throughput if the channel is clear, but if the frame gets

corrupted during transmission an increased amount of data is lost. Therefore it can be beneficial to aggregate a frame with individual headers. Even though this gives a slightly increased processing and transmission overhead compared to regular aggreagation where there is no individual headers, each frame canbe selectively acknowledged. This means that only a few frames has to be retransmitted in case of corruption, and not the entire aggregation.

On the physical layer there are a couple of suggested countermeasures:

- Changing the transmission power levels
- Lowering data rates
- Adjusting CCA threshold
- Forward error correction
- Changing packet sizes

[[Fylle ut om de forskjellige]]

It is shown that the previous countermeasures only have limited impact, while changing the channel of an AP remains the most effective. [5].

2.5 Channels

802.11 b/g/n uses the range of frequencies from 2.400-2.500 GHz on the Industrial, Scientific and Medical (ISM) band. The increasingly popular 801.11n/ac also uses a range of 5 GHz frequencies on the Unlicensed National Information Infrastructure (UNII) band, which offers more frequencies [6]. Other than that the properties and challenges for the different bands are ultimately the same. The distribution of the frequencies on the 2.4 GHz band to the different channels is illustrated by figure 2.4. The frequencies listed are the center frequencies of each channel. In practice this means that an AP that transmits on one channel will interfere with close channels in both directions. Two channels are entirely non-interfering if they send on two frequencies f_1 and f_2 so that $|f_1 - f_2| > 0.025$. This means that there are in total three completely non-overlapping channels, 1, 6 and 11. The process of deciding which channel to transmit on is called channel allocation.

2.6 Overview of Channel Allocation Algorithms

There exists many different channel allocation algorithms, and we'll introduce some relevant here.

2.6.1 Least Congested Search

CHNL_ID	Frequency (MHz)
1	2412
2	2417
3	2422
4	2427
5	2432
6	2437
7	2442
8	2447
9	2452
10	2457
11	2462
12	2467
13	2472
14	2484

Figure 2.4: Channel/frequency distribution

Chapter 3

Data acquisition and structure

Before implementing and testing any group creation algorithms, we need to gather usable data to perform testing on. The data consists of the topology size, and where in the topology a node is located.

For basic testing we can create our own data, and either assume the location of nodes, or randomly distribute them on a topology.

In a more reliable testing setup we should ideally get information from a real world data source. The topology would then represent a section of the world, being either a city, town or neighbourhood, and the nodes would be APs most likely located in each household.

3.1 Requirements

The testing data should be a set of nodes that has two coordinates x and y on a two-dimensional grid. Henceforth we will call refer to the populated grid as the topology. When the nodes are placed on the topology, we can compute which nodes it can hear on the radio and add these to the node's SSID-scan list. This list contains the names of all the nodes it can hear, and how loud it is heard measured in $-dBi$. Additionally the following parameters should be variable depending on each test scenario:

- Topology size, variable width of x- and y-axis.
- Number of nodes
- Minimum distance between nodes (in meters)
- Minimum loudness measured in $-dBi$ for a node to account another node as a neighbour (e.g -100 is too low for anyone to hear).

3.2 Program design

The topology generation program consists of two main functionalitites.

The first functionality is being be able to create a topology and generate nodes which are uniformly and randomly positioned on the network topology. The size of the topology, the number of nodes and the minimum distance between nodes are properties that can be given as input arguments to the program.

The second functionality is computing which nodes can hear each other. We are assuming all nodes are transmitting with equal strength, and that the environment is flat and obstacle free. All the neighbouring nodes that can be heard by a node, is added to its list of neighbours, and it stores the $-dBi$ value so it later can be shared with the group.

The resulting program, written in Python 3[3], contains an importable *topology class*. This way, for further testing we can use different data sources to get the positions of nodes, and only let the topology class compute the list of neighbours.

The interference levels between APs is calculated by iterating through all the nodes. For each node N we record its x and y position, and then start a second iteration through the nodes. For each node n in the second iteration we calculate the distance d in meters between N and n using Euclidean distance. Knowing the the distance between the nodes, we can use the formula for free space path loss [11] to compute the $-dBI$ values. The code can be seen in figure 3.1.

For the sake of cross compatibility with other applications, the result of the computation represented in json. A topology consists of as many JSON node-objects as there are nodes.

```
#In topology class
def measureInterference(self):
 for nodeSubject in self._nodes:
  for nodeObject in self._nodes:
    nodeSubject.calculateInterferenceTo(nodeObject)

#In node class
def calculateInterferenceTo(self, nodeObject):
 if self == nodeObject:
  return
 dist = round(self.distanceTo(nodeObject))

#If  nodes have same coordinate, set high interference.
 if (dist == 0):
  dBi = -40
 else:
  dBi  = self.measureDbi(dist) * -1

def measureDbi(self, dist):
 return (20 * math.log(self._frequency, 10)) +
(20 * math.log(dist, 10)) - 27.55
```

Figure 3.1: Computing the interference between nodes

3.3 Data output and visual representation

Figure following illustrates the node structure, and is an example of how a node with two neighbours will look:

```
1: {
 "posX": 100,
 "posY": 100,
 "ssid": "NODE1",
 "neighbourCount": 2,
 "neighbours": {
    0: {
     "ssid": NODE2,
     "dbi": -77.23
    },
    1: {
     "ssid": NODE3,
     "dbi": -79.52
    }
  }
}
```

Figure 3.2: JSON output structure

We can run the program in the following way

```
./GenerateTopology.py -n 200 -w 100 -h 100 --space 10 --dbi 85
```

Which instructs the program to create a topology with 200 nodes. The topology should be 100 by 100 meters large, and there should be at least 10 meters between each node. The *dbi* parameter makes sure that only nodes which can be heard with a $-dBi$-value of -85 or larger should be considered neighbours.

The output is a 3.8 MB large file containing the resulting topology-data in JSON.

By writing a simple JavaScript browser application meant to interpret the output, we can visually represent the nodes on a grid, and the result will look like what can be seen in figure 3.3

3.4 WiGLE

WiGLE (Wireless Geographical Logging Engine) [10] is a project started in 2001 which purpose is to gather information about wireless networks. The data they have accumulated in their database is entirely user submitted. Anyone can download an Android app published by WiGLE, and then use the

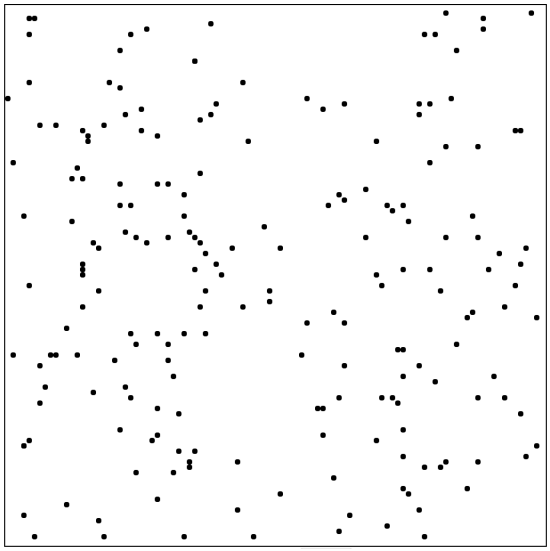

Figure 3.3: Generated topology with random, uniform distribution

app for wardriving [1], then submit the data to WiGLE's centralized database. All the APs can be viewed on an interactive map provided on WiGLE's website. The data can also be accessed through an API call. Using their service is entirely free, but the amount of data that can be requested is throttled on a day-to-day basis. As they openly support research projects, they provided us with an account with a slightly higher daily data limit.

To us this is interesting because we can use the location of APs to create more realistic network topologies, and see how well the algorithm performs on these.

3.4.1 REST API

Their REST API provides data presented in JSON, and offers different services, such as user profile operations, statistical information and network search.

We only need to use the networks search part of the API for our purpose. By passing it a request for nodes between two latitudes and longitudes, the API responds with the APs in that area. A request will look something like what can be seen in figure 3.4 The parameters $latrange1$ and $longrange1$ are the coordinates that marks the beginning of the area we are interested in, where $latrange2$ and $longrange2$ marks the end. As WiGLE at most returns

[1] Wardriving is the act of tracking wireless networks using a laptop or a phone, and then store the information about each network [12].

```
https://api.wigle.net/api/v2/network/search?
    first=0&latrange1=37.808469&latrange2
    =37.746744&longrange1=-122.539232&longrange2
    =-122.381355
```

Figure 3.4: Example of a Wigle API request

information about 100 APs for every query, we need the *start* parameter to tell WiGLE at which index offset we want to begin fetching data from. A start value of 0 means we fetch in the range $0 - 99$, a value of 100 means in the range $100 - 199$ and so on. The JSON response for a succesful request for an AP, can be seen in figure 3.5.

We are primarily interested in the properties *trilong* and *trilat*, which is the triangulated coordinates of the AP.

3.4.2 Using Wigle data

WiGLE provides data about the location of access points. However, to be able to use the data for our group computation, we must translate the global coordinates to two dimensional coordinates. We can then place each AP on a topology with the same format as the generated topologies we created in section **??**. This is necessary for two reasons.

The first reason is that creating a group creation program that can operate on the longitudes and latitudes directly adds more complexity. Both with regards to group computation and visual representation.

The second reason is that the WiGLE data does not contain information about which neighbours each AP can hear, and with what $-dBi$ strengths they are heard. This will have to be computed like earlier, and by parsing it to the format we designed in section 3.3 we can reuse the code to generate the neighbour lists.

The haversine formula [[ref or explain]] gives us the distance in meters between two coordinates. By calculating the distane between the latitude startpoint and the latitude endpoint, we can get the size of one axis in meters. By computing the distance between the longitude start and endpoint we get the size of the other axis. To place a node correctly in the coordinate system, we simply use the haversine formula on the origin coordinates to compute the distances between each axis.

All of this has been implemented in a python program. As input it takes both latitude and longitude start and end coordinates, then fetches all nodes within that range and inserts them on a plane.

```
{
        "userfound": false,
                "qos": 0,
                "comment": null,
                "lastupdt": "2015-12-22T17:49:34.000Z",
                "bcninterval": 0,
                "dhcp": "?",
                "lasttime": "2015-12-22T17:49:15.000Z",
                "trilong": 10.82792618,
                "netid": "5C:9E:FF:2B:54:84",
                "freenet": "?",
                "trilat": 62.2816925,
                "name": null,
                "firsttime": "2015-12-22T20:55:01.000Z",
                "type": "infra",
                "ssid": "NETGEAR23",
                "paynet": "?",
                "wep": "2",
                "transid": "20151222-00207",
                "channel": 52
}
```


Figure 3.5: REST API response with AP data

Chapter 4

Access point clustering

4.1 Introduction

While it at first sounds incredibly desirable to let the entire population of for instance New York's access points organize themselves in an optimal channel-plan, at second thought the idea may prove to be a little ambitious. Being limited by the NP-complete nature of DSATUR or similar graph coloring algorithms we have to set some reasonable constraints on the size of the *collaborating group*. The amount of nodes that will collaborate on the problem of finding an optimal channel distribution plan may not need to be very high.

Let us for a moment look at an ideal example: a city that only consists of small apartment buildings that are entirely isolated from interference produced in external networks. How could we build a group in such a case? Turns out it is not that hard. Each access point would be able to collaborate with every other observable access point. When computing channel distribution there would be no risk of surpassing the viable amount of nodes to compute the distribution for, as the group is limited to the apartment building. Sadly for us the world is not ideal, and it is not impossible that every access point in New York can observe eachother through a transitive relation. This means the size of the collaborating group would be vast and completely unviable.

This does not mean that creating reasonably sized groups of access points is impossible, but it poses a more difficult challenge. We need to filter out redundant nodes to create an approximated version of the ideal example.

Having a picture of the typical cityscape in mind, we know that buildings are naturally separated by streets, bridges, parks, and so on. Rural areas are similar, but networks are spaced more unevenly and usually only affecting each other in one plane.

Remembering attenuation, we know that RF-interference is also a property of distance. These pieces of information tells us that RSSI readings between access points in two separated buildings should be lower than readings between access points in adjacent apartments. What we will be looking at in the following chapter is how we can utilize the RSSI readings between access points to build a clustering algorithm that creates reasonable groups of access points that discriminates between low-impact and high-impact nodes.

4.2 Problem overview

The main problem we are dealing with is finding a way for access points to group together in clusters that are geographically close to each other. This can either be solved with a centralized coordinator or with a peer-to-peer distributed protocol. In this thesis we will be focusing on the distributed approach, but first we will take a look at the pros and cons of each approach, and the work that has already been done on the field.

4.2.1 Centralized model

Let us briefly envision how a centralized model might look. As we want to propose a solution that works across private consumer networks as well as larger corporate networks, we can not assume that all networks are under the same administrative domain. This is where the centralized approach is limited, and why it may not be ideal.

Just like the distributed version is, the centralized controller is also restricted by the NP-complete nature of all graph colouring algorithms. To be able to create a channel plan which is as optimal as possible, it has to identify groups of nodes that impacts each other severely.

The advantage of the centralized model is the ability to have the full picture of access points readily available. Let us assume the controller is placed at an ISP or a router manifacturer. The controller could potentially know exactly where in the world the access points are placed. Creating groups would be as simple as dividing within naturally separative geographical barriers like roads, streets and buildings.

For a central controller to be able to identify APs that are near eachother and compute the optimal channel distribution, APs need to report their radio readings to the controller. The controller then needs to identify which of all the observed APs it can control, and which is not controllable and has to be treated as noise. There would be no requirement for communication in between nodes, which reduces complexity a lot.

A major drawback of the centralized approach is that nodes that are near eachother have to be under the same controller. If there are too many nodes around that can not be controlled by the controller, all nearby access points would be treated as noise and regular channel allocation schemes would have to be applied. Even though in there is a tendency for apartments in the same building to have the same ISP in Norway, there is no gurantee for that to always be the case.

4.2.2 Distributed approach

Now that we have looked at how a centralized model might look, we can dive further into the main matter of the thesis. A simplistic way to portray the idea behind the distributed approach is: nodes only depend on their own radio RSSI readings to be able to create groups. When the group is formed, all of its members can compute a channel plan which is as optimal possible. In reality this idea has some major challenges that we have to overcome. Here are the most prominent ones:

- The communication channel. Nodes have to be able to communicate with eachother. The 802.11 standard describes no protocol for communication between access points that are not on the same extended service set. This communication would have to happen on the network layer, preferably over TCP. This communication channel would have to convey a messages that enables group creation, most likely a custom protocol.

- The group creation itself. Based on only the RSSI-readings and the communication channel, nodes have to be able to organize themselves in a tight cluster of nodes that includes all nodes that impacts each other most severely. This problem essentially boils down to a clustering problem. This is the main issue we will try to find a reasonable solution for.

- Node state. As all nodes have to compute a channel plan for the group, they all need to have the same consistent information about the members of the groups and every other node's RSSI-readings.

- Channel plan computation. Even though the centralized approach would have a similar problem, it can be more difficult to solve in a distributed fashion, as it is reasonable to assume that an access point has limited computational power. We will not discuss a solution for this in depth, as it is out of the scope of the thesis, but there are already a

couple of algorithms that treat channel allocation as a graph colouring problem.

4.3 Computation assumptions and requirements

Proceeding in the next chapters we will be looking at possible ways for nodes to organize themselves into groups in a peer-to-peer fashion. The data we will be using is the data we fetched in chapter 3 and structured in the JSON format described. The data provides a topology of nodes where all nodes have a list of neighbours that they can hear, with the appropriate RSSI-reading for how loud they can be heard. This is the only information the nodes are given, and based on the RSSI-values of their neighbours they should be able to organize themselves. To simplify the computation procedure and to reduce the workload for building the program we are going to to a number of assumptions.

1. All APs visible on any AP's radio also runs the group creations algorithm. This is an assumption needed to be able to see how well the group performs in an optimal condition. It can be interesting to add rogue nodes that does not take part in group creation at a later point to see how well group creation works in a less optimal condition.

2. If a node has another node in its neighbour list (meaning that they can hear each other on their radio), it also knows how to directly contact the node (e.g. via TCP). There is an underlying and working protocol that lets nodes communicate and exchange information about their RSSI-readings and group membership.

3. All nodes in a group are completely synchronous, and always have an equal image of the state of the group at a given time.

4.4 Proposing an algorithm

With the definition of a connected group in place, and the assumptions and requirements accounted for, we can begin proposing an algorithm.

Henceforth we will for the sake of simplicity be referring to APs that are running the group algorithm as *nodes*.

Each node begins by identifying itself as a member of a group that only contains itself. Let us call it group a. The node shares all of its radio readings with the rest of the members in group a. Group a looks at all the

radio readings of every member, and picks the node with the highest dBm value to contact. Of course in the beginning there is only one node in group a, so this node's radio readings alone will decide which group to merge with.

The neighbour node that has the highest dBm value is in group b, and this is the most crucial node to collaborate with. Hence the group a merges its own group together with the group of neighbour b. This happens in the following way: the members of the two groups exchange information about all their member nodes and their radio readings. The data is now identical for all the members of both groups, and they can make identical choices, hence they are now in the same group.

If the number of members n exceeds a predefined threshold $memberThresh$ after a merge, the group should begin kicking out members from the group, starting with the node that influences the rest of the group the least.

To prevent the group from oscillating between kicking out members and rejoining members, once it reaches its maximum size it should be locked for further merges.

4.5 Program design

The program is designed so that it can be modified to accomodate different algorithm types, while still keeping the basic framework untouched. It is implemented in Python 3 [3], and is designed to run on the output from the data generation program. Seperating these programs is important, as group computation should be fast operation, while the data generation (or fetching) is a slow process and should only be done once every time a new data set is required. A major design choice that had to be made early in the planning process was whether or not the computation should be run in parallel or in sequence.

If the program was implemeneted in a way that computation ran in parallel, it could assign one thread to each node, and let the nodes interact through an interface to one another. Thus simulating a peer-to-peer protocol interface. Comparing with a sequential implementation, a parallel computation could arguably be a better representation of the real world, as the nodes act independely of each other and the order of events is to some degree random. It is exactly because of this chaotic nature of parallel computation that the program is designed to run in sequence. The benefit of this is consistent and recreateable results which for needs outweighs the benefits of a parallel implementation. It also reduces the complexity of doing operations on groups memberships since we don't get any race conditions. This particular point is also accounted for in the program assumptions, where we assume that all

nodes have the exact same state. It has to be said that given a perfect algorithm that always find the best division of groups, the resulting groups should end up being the same whether or not it is done sequentially or in parallel, but at this point we don't know what results to expect.

The group framework consists of 3 classes with different responsibilities:

- **Group**. An object of this class is an abstraction of all the nodes that belong to the same group. Because we have assumed that all nodes have equal information about group membership and RSSI-readings, we can store all the members of each group in a list and let the group object act as a unified entity on behalf of the entire group. All the interfaces and logic for forming groups is placed within this class. A method named `iteration` has the responsibility of triggering the appropriate action based on the state of the group. For instance adding node, removing a node or merging the group with another. If an action was performed the method returns 1, else it returns 0.

- **GroupCollection**. An object of this class should contain all the groups used in a simulation. Its main functional responsibility is looping through all the groups and calling the `iteration` method once. This is done in the GroupCollection's `iterate` method. It accumulates the amount of changes done in all the groups, by adding the return values of the Group object's `iteration` method. It also handles the destruction of groups, and bootstrapping of newly created groups.

- **Simulation**. An object of this class handles the bootstrapping of groups, where all nodes (given in the input file) are parsed and put in their own grown group. Consequently at the beginning of each computation the amount of groups is equal to the amount of nodes. The Simulation class is also responsible for starting and stopping the simulation itself. After bootstrapping all the groups, the Simulation enters a loop where it calls the `iterate` method in the GroupCollections object once every run. All the groups have converged and reached a steady state once the amount of changes returned by the GroupCollection's `iterate` method is equal to zero. The results are written to file.

Algorithm 1 How to write algorithms

this text how to write algorithm with LaTeX2e initialization

while not at end of this document **do** read current understand go to next section current section becomes this one go back to the beginning of current section

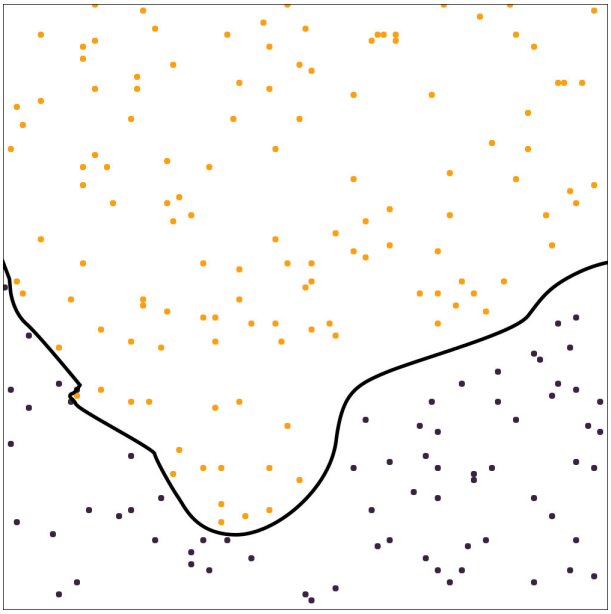

Figure 4.1: 200 nodes, $memberThresh = 128$, $size = 200x200$

4.6 Results

4.6.1 Uniformly distributed nodes

We will look at how groups were created in different topology scenarios. All topologies presented in this section was created by the topology generation program, but with different input parameters. Groups are distinguished by node color, where nodes of the same color represents members of the same group.

Scenario 1

Computed with 200 nodes with a maximum of 128 members in each group.

As can be seen in figure 4.1 the algorithm divides the nodes in two sections. For clarity, a divisive line has been drawn around each group, in case colors are not available. When two major groups merged, the biggest groups surpasssed 128 members and began kicking out members. The excess members formed the black group at the bottom.

Scenario 2

Computed with 200 nodes with a maximum of 10 members in each group.

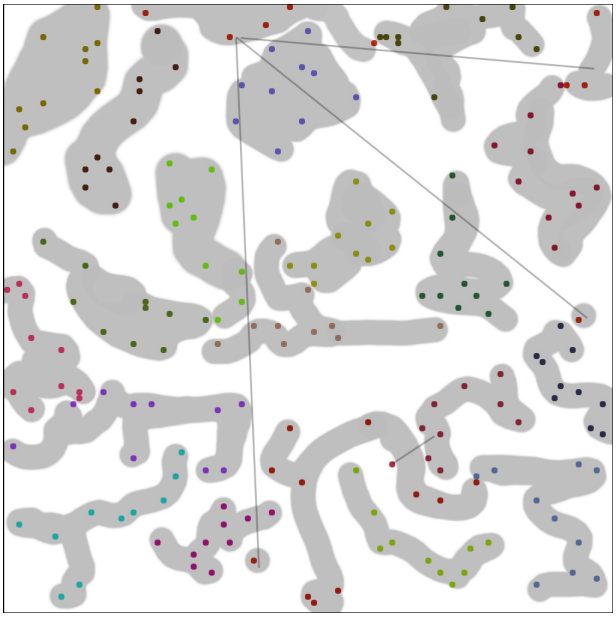

Figure 4.2: 200 nodes, $memberThresh = 10$, $size = 200x200$

The result of this computation, seen in figure 4.2, is a little less obvious. The groups are again distinguished by different color, but for clarity we add a gray connecting blob for nodes in the same group. Also blobs connected with a line are in the same group.

It is worthy to take notice that one group is especially scattered around the graph. At first eyesight, it looks like an algorithm deficiency, but the reason is quite simple: when nodes are kicked out of a group during a merge, they will connet to other nodes that belong in a group where n has not yet reached $memberThresh$. When this have happened a couple of times, everyone has found a group except for the remaining few. These are typically straggler nodes or smaller clusters separated from the others. They are not big enough to reach the group $memberThresh$ on their own, so the merge with other nodes that are in unmaxed groups. Thus, even though they have neighbours which influence them more, they can only merge with nodes further away, because that is the only unlocked group that remains.

Scenario 3

Computed with 5000 nodes, with a maximum for 64 members in each group.

Figure 4.3 shows the result of the computation. Because of the quantity of nodes and the clear separation of groups, they are easily distinguished by color. This topology is much denser than the others, and can vaguely

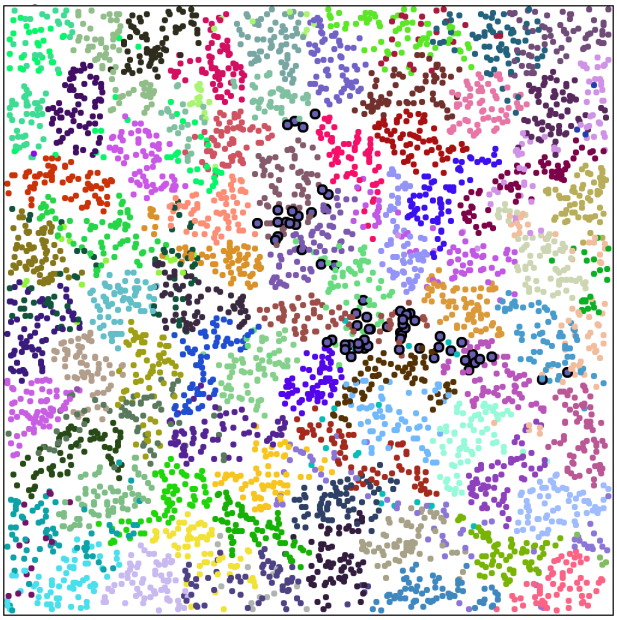

Figure 4.3: 5000 nodes, $memberThresh = 64$, $size = 2000x2000$

resemble the density of highly populated areas.

We can clearly see that the overall tendency is that groups are formed in concentrated areas of nodes. However, some groups are scattered, sometimes all over the map. An example of a scattered group is highlighted in figure 4.3. Its member nodes has a thicker black line around them.

4.7 Results

By running the scripts that parses data from Wigle on populated areas, we should get an idea on how the algorithm performs in more realistic topologies. We will have a look at three scenarios where the group allocation data is based on AP-data.

Three suitable locations has been selected to perform the testing on Lillehammer (Norway) a smaller city, Tynset (Norway) a less densely populated area, and Forks (Washington, United States). All tests were ran with a maximum group size of 128, and a $-dBi$ threshold of -80.

Lillehammer

The computation results of Lillehammer can be seen in figure 4.4. The topology is of medium density, consisting of 4990 APs, and is 2572 meters

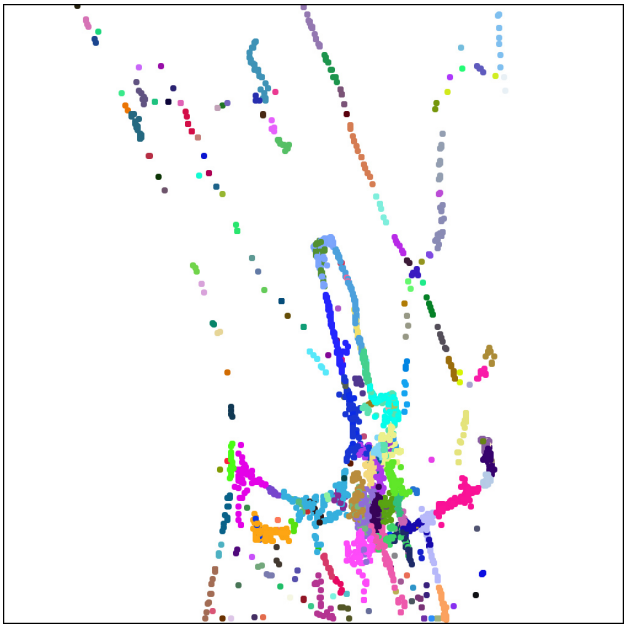

Figure 4.4: Lillehammer

high and 8418 meters wide. From the tight clusters in the middle it is easy to make out the city centre. We can clearly see different groups with different sizes. Some of the smallest groups are highly likely so small because the distance to other nodes is too high for the group to hear. In the denser areas they are occasionally very entangeled, and it can be hard to make out the group borders.

Another thing to notice is that APs are nearly always placed in straight lines. The straight lines are roads, and as Wigle collects data based on triangulation, the nodes that is only seen once will get the position they are observed in, and not an actual triangulated position.

Tynset

The computation results of Tynset can be seen in figure 4.5. The topology consists of 726 APs, is 1670 meters high and 6720 meters wide. Unsurprisingly it resembles Lillehammer on a smaller scale. Again we see some very clearly defined groups, but in the city centre there are groups which overlaps. We can also see nodes that are alone in their group, because they are too far away from anyone else. Much like Lillehammer, this topology is also strongly affected by the weak triangulation of the APs, so most APs seems to be placed on top of a road.

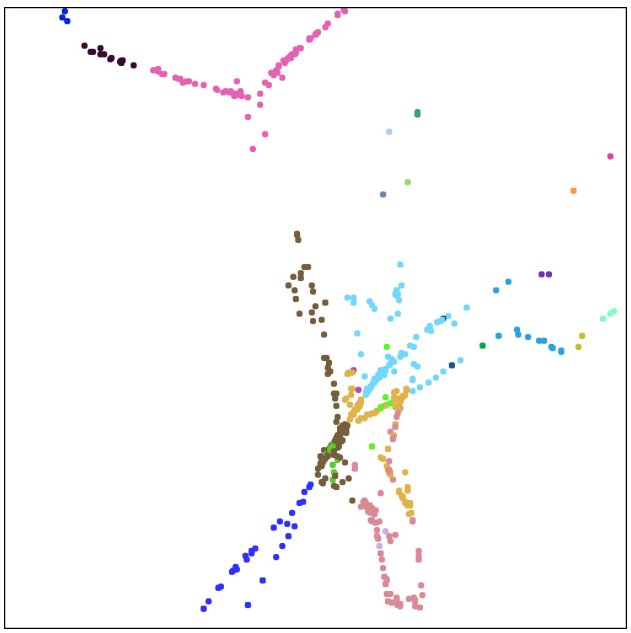

Figure 4.5: Tynset

Forks

The computation results of Forks can be seen in figure 4.6. The topology consists of 1715 nodes, and is 2122 meters high and 4495 meters wide. It is important to include, because it is quite different from the other topologies and represents a variation from the typical town and city structure of Norway. The size of the groups are a little more uniform when comparing it to the others. This can explained by the the smaller area the town is contained within. When a group is not full, it will almost always hear someone that it can merge with. We still have groups overlapping each other in the denser regions in Forks as well. What is worth noticing is that the APs are positioned more realistically as locations of households. American towns looks more like a grid with roads in between households, which makes triangulation easy and a lot more accurate.

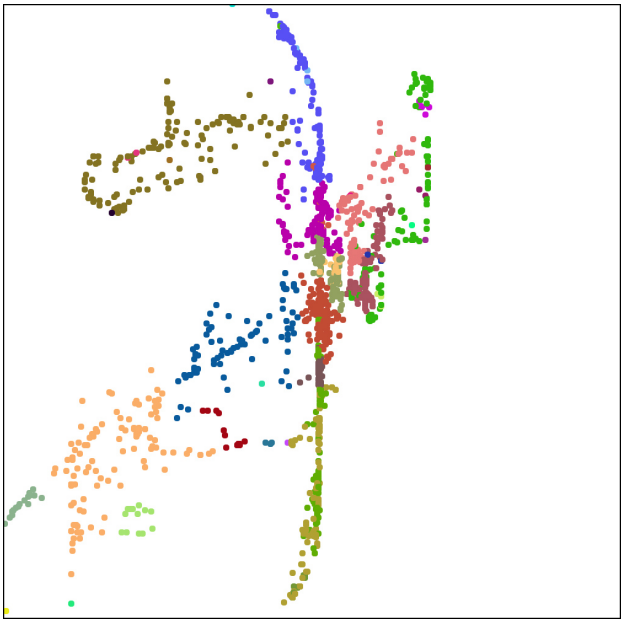

Figure 4.6: Forks

Bibliography

[1] WiFi Alliance. *WiFi Alliance News Release*. Online: accessed 26-September-2017. 2016. URL: https://www.wi-fi.org/news-events/newsroom/wi-fi-device-shipments-to-surpass-15-billion-by-end-of-2016.

[2] G. Bianchi. 'Performance analysis of the IEEE 802.11 distributed coordination function'. In: *IEEE Journal on Selected Areas in Communications* 18.3 (Mar. 2000), pp. 535–547. ISSN: 0733-8716. DOI: 10.1109/49.840210.

[3] Python Software Foundation. *Python 3*. Online: accessed 05-July-2017. 2017. URL: http://python.org.

[4] Gartner. *Gartner Press Release*. Online: accessed 26-September-2017. 2017. URL: http://http://www.gartner.com/newsroom/id/3598917.

[5] Ramakrishna Gummadi et al. 'Understanding and Mitigating the Impact of RF Interference on 802.11 Networks'. In: *SIGCOMM Comput. Commun. Rev.* 37.4 (Aug. 2007), pp. 385–396. ISSN: 0146-4833. DOI: 10.1145/1282427.1282424. URL: http://doi.acm.org/10.1145/1282427.1282424.

[6] 'IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 7: 4.9 GHz-5 GHz Operation in Japan'. In: *IEEE Std 802.11j-2004* (2004), pp. 1–40. DOI: 10.1109/IEEESTD.2004.95388.

[7] Rohan Murty et al. 'Designing High Performance Enterprise Wi-Fi Networks'. In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'08. San Francisco, California: USENIX Association, 2008, pp. 73–88. ISBN: 111-999-5555-22-1. URL: http://dl.acm.org/citation.cfm?id=1387589.1387595.

[8] Rohan Murty et al. 'Dyson: An Architecture for Extensible Wireless LANs'. In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'10. Boston, MA: USENIX Association, 2010, pp. 15–15. URL: http://dl.acm.org/citation.cfm?id=1855840.1855855.

[9] Lalith Suresh et al. 'Towards Programmable Enterprise WLANS with Odin'. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 115–120. ISBN: 978-1-4503-1477-0. DOI: 10.1145/2342441.2342465. URL: http://doi.acm.org/10.1145/2342441.2342465.

[10] WiGLE. *WiGLE: Wireless Network Mapping*. July 2017. URL: https://wigle.net/.

[11] Wikipedia. *Free-space path loss — Wikipedia, The Free Encyclopedia*. Accessed 4th July, 2017. 2017. URL: http://en.wikipedia.org/w/index.php?title=Free-space%5C%20path%5C%20loss&oldid=765763753.

[12] Wikipedia. *Wardriving — Wikipedia, The Free Encyclopedia*. Online; accessed 09-July-2017. 2017. URL: http://en.wikipedia.org/w/index.php?title=Wardriving%5C&oldid=788183563.