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

6th December 2017

Abstract

Contents

List of Figures

Chapter 1

Introduction

The presence of Internet connected devices is still rapidly expanding. The latest forecast esitmates that there will be about 27 billion devices connected to the Internet by 2021 [6], while by the end of the 2017 the number connected devices in use will be 8.4 billion [9]. In 2020 households alone will be responsible for over 10 billion devices that are able to wirelessly connect to the home router [1], and wireless traffic will account for 63 percent of all IP traffic in the world [6]. Traditionally devices that are commonly connected to the Internet in a household are computers, phones, watches, smart TVs, audio systems, and lately also private network storage systems. As the era of Internet of Things (IoT) has rapidly descended upon us, increasingly also less obvious utilites are connected to the Internet. These devices may span everything from lights and HVAC systems to coffee machines, fridges and even toasters. Whereas in the early 2000s it was common to own one or two MAC-addresses per person, at the date of writing it is not unusual to possess a two digit number of MAC-addresses.

But it is not only the numbers of devices that has changed. The consumers' expectations and demands are also being altered over time. While ubiquitous connectivity is a buzzword often heard in the context of future 5G networks, ubiquitous access is already expected in modern households. The demand for Wi-Fi coverage extends through the entire home: the Internet radio in the garage, the video stream on the basement couch and the gaming laptop on the bedroom. All demands coverage and expects mobility at the same time. These devices also have something else in common that reflects the change that has happened the last years: demand for high data rate. In 2016 about 73 percent of all IP traffic orginated from video streams, and in 2021 this number is expected to increase all the way up to 82 percent [6].

This demand for continous streams of high bitrate video poses a challenge that can not only be solved by providing larger bandwidth. First of all, video

streams have high Quality of Service (QoS) demands. Buffering of videos and reduction of video quality negatively affects the subjective experience. Secondly, it is not unusual that different video flows occur simultaneously in a single residence. If such traffic is transmitted wirelessly it results in an inherently busy transmission medium. Because of the international spectrum allocation standards, Wi-Fi is largely restricted to a small number of channels. To overcome this challenge, the 802.11 protocol specifies a set of rules which allows only one device in the vicinity to transmit on a channel at the same time. This poses yet another challenge of distributing channels in a way that as few devices on the same channels as possible are close to one another.

In short, while the physical layer (PHY) traditionally has been upgraded to meet the new demands in bitrate (e.g. fiber to the home, and MIMO in 802.11ac) Wireless LAN connections struggle under the heavy impact of RF-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 Wi-Fi 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 QoS comes second.

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 Wi-Fi network in their home is constantly underperforming, it is not unusual a customer to upgrade the data rate of their agreement. If interfering networks were the perpatrators, usually increasing the bandwith will have no effect. This leads to customers being largely frustrated with their Internet service providers, even though it is the underlying technology, and not the ISP which is at fault. A customer service representative or a tech-savvy consumer 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 give a short relief 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.

There has been done research on how centralized controllers can benefit the deployment of large enterprise networks ([14], [13] and [17]), but in this thesis we will address the emerging issue of deployment of Wi-Fi in residential areas and how routers and access points could organize themselves to allow distributed control and coopeation on channel allocation, where centralized controllers are impractical or impossible.

1.2 Problem definition

There are 3 non-overlapping channels on the 2.4GHz spectrum that 802.11 Wireless LANs uses. One of the common ways of selecting a channel is done by letting an access point sense which channel has the lowest interference levels, also called least congested channel search. When channels are selected in a selfish manner, where the only available information about the surrounding networks are obtained via subjective radio readings, it is highly unlikely that the distribution of channels in a confined area (e.g. an apartment block) becomes optimal. Ideally all the access points would be configured so that the channel distribution in this area lead to as few equal channels as possible being adjacent to one another. While this boils down to a NP-complete graph-coloring problem, it is not the focus of this thesis. The main challenge to be considered is the clustering problem of identifying access points that heavily impacts eachother, and consider different approaches to how they can form groups in a distributed manner that seeks to intuitively identify nodes that lie withing the same confined areas, as apartment buildings dense residential areas, etc.

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 manual 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 Basic radio 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 the receiving radio is within transmit range of the transmitter. 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

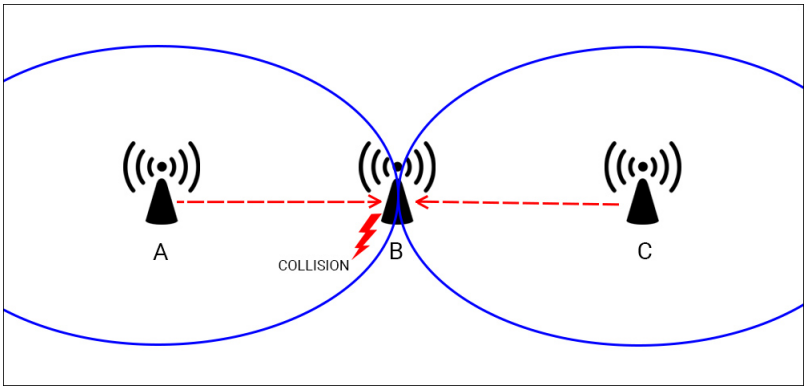

Figure 2.1: A and C sending to node B unknowingly at the same time, resulting in a collision

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

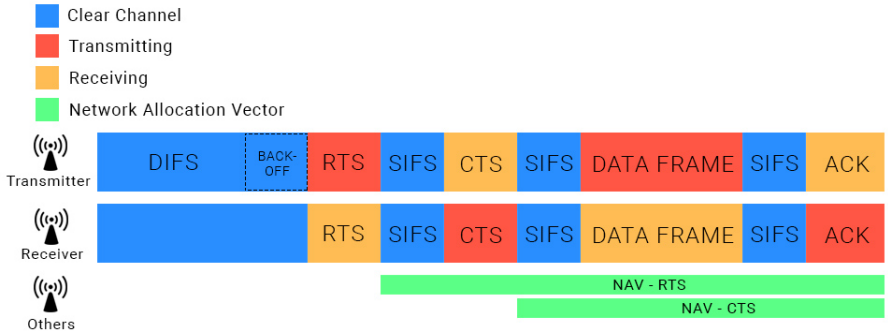

Figure 2.2: The timeline one frame transmission cycle in DCF mode

$(0, 2^n * CW_{\min})$, 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 [3].

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.

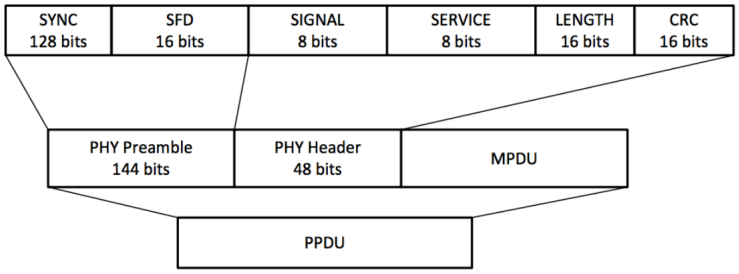

Figure 2.3: DSSS PHY PPDU format from IEEE Std 802.11-2016

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. [10].

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 [11]. 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.

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

Related work

3.1 Brief survey of related work

3.1.1 Cisco RRM

CISCO offers a solution directed at enterprise networks [5], where implementing and managing a centralized controller is a more managable task than in residential networks. Their architecture consists of one or more Wireless Lan Controllers (WLCs). The WLCs creates a group name which all APs part of the same RF-group is aware of. All APs broadcasts their RF-group name, along with the IP-address of their controller. Any other AP sharing the same RF-group name reports the incoming broadcast message to the controller, and then rebroadcasts the packet. This way the controller becomes aware of all APs that can hear each other, similar to the flooding routing mechanism, and they can form logical groups based on this information. If applicable the RF-groups perform leader election to decide which controller becomes the RF Group leader, but there can also a preconfigured leader. The RF group leader has the responsibility of running the relevant channel allocation algorithms and ajusting the radio power level of the APs.

3.1.2 DenseAP

DenseAP described in [13] aims to restructure the infrastructure of enterprise networks. The two fundamental changes they suggest is to deploy APs a lot denser (hence the name), and moving the task of associating clients with APs to a centralized controller. The reasoning behind the dense deployment of APs is that signals diminishes quickly in an indoor environment, and ideally an AP can always be associated with an AP in the close vicinity. The argument they use for moving the association decision away from the client

and over to a central controller, is that a client only uses signal strength as the metric to decide which to associative with. They emphasize that this can be suboptimal in conferences and meeting room environments, where many clients seek to associate with an AP at the same time. If all clients pick the same AP it also means all clients will transmit on the same channel, and RF-interference can reduce the throughput on the medium. Their infrastructure consists of DenseAP access points (DAPs) and DenseAP Controllers (DCs). The DAPs sends periodic reports to the DC, which contains information as RSSI measures, channel interference, and associated clients. Based on this information the DC decides which DAP each client should associate with, and also which channel each DAP should transmit on.

3.1.3 HiveOS

HiveOs [15], developed by Aerohive Networks offers distributed protocols and mechanisms to improve Wireless LANs in enterprise networks. The APs in the network are called HiveAPs, and they offer services such as

- Band steering: if an device can operate on the 5Ghz band, it will be forced to connect to the 5Ghz network to optimize the utilization of the radio spectrum.

- Load balancing: all HiveAPs have real-time information about how clients performs. If a client tries to associate with a new HiveAP, it will only be accepted if the new HiveAP has a low enough load to handle more clients. It would also know if other neighbouring APs are better suited to handle the load of the new client. This is achieved by witholding probe responses.

- Channel allocation: by using the Aerohive Channel Selection Protocol, HiveAPs tries to select the channel with the lowest co-channel interference. Their channel selection protocol uses 5 measurements, two static and 3 dynamic. The first static measurements is the number of nearby APs who are operating on the same channel. The more APs there are the higher the penalty, but the penalty per AP reduces as the number of APs increases. This is because the first one(s) are the most critical. The other static cost factor is what power level can be transmitted at the given channel, as this may differ on some 5GHz non-overlapping channels. The dynamic measurements are CRC-error rate, channel utlization, and the utilization of overlapping APs. All of these dynamic factors can penalize a channel with 0 to 3.5

3.1.4 ResFi

ResFi [22] is the only significant related work that directly aims to enable self-organized management in residential deployments of wireless LAN. Works such as [13], [5] and [15] are all directed toward enterprise networks. As we also aim to mainly concern ourselves with residential networks and their infrastrucure, ResFi is especially interesting to look at. ResFi assumes that all access points have two interfaces, one connected to a wired backone (e.g Internet), and another 802.11 compatible wireless interface. They also assume a Radio Resource Management Unit (RRMU), which in most residential homes is a router that controls the channel and power level of the antenna. In short ResFi enables connectivity between distributed access points without imposing a central controller on the access points, and without doing any modifications to hardware and drivers (like modifying standards or requiring propriatary equipment).

Operation

1. An AP that has just booted up beacons a frame on all available channels. This frame contains: a globally routable IP-address, port, and two cryptographic keys. One transient key for group communication, and the public key for the RRMU.

2. All APs that can hear the beacon responds with a probe response containing the same information as in 1.

3. When the scan is complete, the new AP can point-to-point communicate with all other APs using the wired backbone and the globally routable IP-address

Implementation

They implemented ResFi on Ubuntu 14.04. It adds vendor specific information elements to the MAC-header with the IP, port, and cryptographic data. This can be done Linux user space by using their modified version of hostapd [18]. As it runs on python, it could in practice be implemented on any Linux system. It uses a south-bound API to communicate with the RRMU, and a north-bound API to enable applications to use ResFi. ResFi itself provides no specific channel allocation mechanism, nor a group-creation/AP-clustering algorithm.

3.1.5 Distributed Clusteering

DCA and DMAC

3.1.6 SCIFI

SCIFI [2] is a centralized channel allocation protocol for infrastructure Wireless LANs that improves the traditional graph-coloring algorith DSATUR. And while it shows that their central coordinator in fact improves the throughput compared to Wireless LANs that are not configured by SCIFI, the algorithm is just an algorithm for setting a channel in a preconfigured administrative domain. It does not deal with how the administrative domains (or collaborative groups as we call it) are defined.

Chapter 4

Data acquisition and structure

We don't have the time nor the resources to organize a large enough testbed to purposefully evaluate the algorithms and suggestions that we will look at in the consecutive chapters. A sufficient testbed would require a large amount of routers (100-200 for a low scale test) in a small geographical area, preferably installed in residential apartment buildings. Not only would the creation of such a testbed require a lot of physical equipment, but there is also a lot of logical challenges that would have to be overcome, such as communication protocols and distributed consensus. These are problems we will address at a later point. Additionally, we could argue that it never advisable to go directly from an idea to a testbed anyway, especially without having any empirical indication of which approaches have merit, and which is not worth the effort to implement. The risk is to waste a lot of time and resources to create a real life implementation that could have been identified as pointless in an early calculation or simulation.

In this chapter we will consider how we can develop a tool that generates and visualizes the layout of network topologies based on constructed, artificial data, or real-life location data of access points.

4.1 Requirements

SSID, current channel frequency, radio power, physical data rate and supported 802.11 standards are just a few properties of a single real-life access point. We are going to represent such access points in our network topology, but the access points we are representing will not be in the transmission (CSMA/CA) state where most of theese properties are used. Our access points are in a state we can call the *group discovery state*, where the goal is to find or create a group to be a part of so that a channel can be assigned

before transmission happens. Hence, to perform our computations, we don't have to consider many of these properties. Actually, we will only store each access point's SSID, because this is a practical way to uniquely identify them [1], and the the list of other access points that can be seen through a WLAN scan with and their observed signal strength. These are the only metrics we require, but unfortunately there is no publicly available data source that contains the subjective radio scans of a large amount of access points in the same area.

As a basis for our simulation we will represent nodes on a two-dimensional grid. Each dataset has to contain a set of nodes with two coordinates x and y so they can positioned on the grid. When the nodes are placed on the grid they represent a network topology and it will be possible to compute an estimation of which nodes can hear each other on the radio, and add these to each 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 with the possibility to give variable width of x- and y-axis as input arguments. These unit of the axes is meters.
- Number of nodes to place on the topology
- Minimum distance between nodes (in meters). This is only to avoid unlikely placement and extreme interference of nodes that are placed on top of each other.
- 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).

4.2 Program design

The topology generation program consists of two main functionalitites.

The first functionality is creating 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 are passed as input arguments to the program.

The second functionality is performing the calculation of which nodes can actually hear each other over the radio. We are assuming all nodes are transmitting with equal strength, and that the environment is flat and

[1] In the real world this is of course not the case, but we will enforce it in our computations. We could just as well call it a unique id.

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 interference levels between access points are calculated by iterating through every access point. 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. The formula for isotropic antennas is described by Friis [8], and can be used to derive the formula for free space path loss [21] that is as follows:

$$FSPL(dB) = 10\log_{10}\left(\frac{(4\pi fd)^2}{c}\right)$$

Where $d = distance$, $f = frequency$ and $c = constant$. The constant c is used to account for different units. We will use the meters to denominate the distance, and megahertz for the frequency. The resulting formula which will be implemented in the program is

$$FSPL(dB) = 20\log_{10}(f) + 20\log_{10}(d) - 27.55$$

As we have to compute the distance from all nodes to every other node, the topology generation program has $O(n^2)$ complexity. A simplified version of the code can be seen in figure 4.1.

The resulting program, written in Python 3[7], 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.

```
#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 4.1: Computing the interference between nodes

4.3 Data output and visual representation

The result of the topology generation is stored in a JSON[12] data file. The data file contains the height and width of the the generated topology, as well as the number of nodes. A `nodes` object consists of as many JSON node-objects as there are nodes. Figure 4.2 illustrates the node structure and is an example of how a node with two neighbours will look.

```
{
  "mapWidth": 100,
  "mapHeight": 100,
  "nodeCount": 3,
  "nodes": {
    1: {
    "posX": 50,
    "posY": 50,
    "ssid": "NODE1",
    "neighbourCount": 2,
    "neighbours": {
      0: {
        "ssid": NODE2,
        "dbi": -77.23
        },
      1: {
        "ssid": NODE3,
        "dbi": -79.52
        }
      }
    }
  },
...
}
```

Figure 4.2: JSON output structure

The program can be run in the following way

```
./GenerateTopology.py -n 500 -w 100 -h 100 --space 10 --dbi 85
```

Which instructs the program to create a topology with 500 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 appear in the neighbour list. The output from this specific run is a 23.2 MB large file containing the resulting topology-data in JSON.

By writing a simple HTML and JavaScript browser application, we can parse the JSON and visually represent the nodes on a grid. The result will look like what can be seen in figure 4.3

4.4 WiGLE

WiGLE (Wireless Geographical Logging Engine) [20] is a project started in 2001 which purpose is to gather information about wireless networks. All the information they collect is user submitted stored in their database. Anyone can download an Android app developed and published by WiGLE, and use

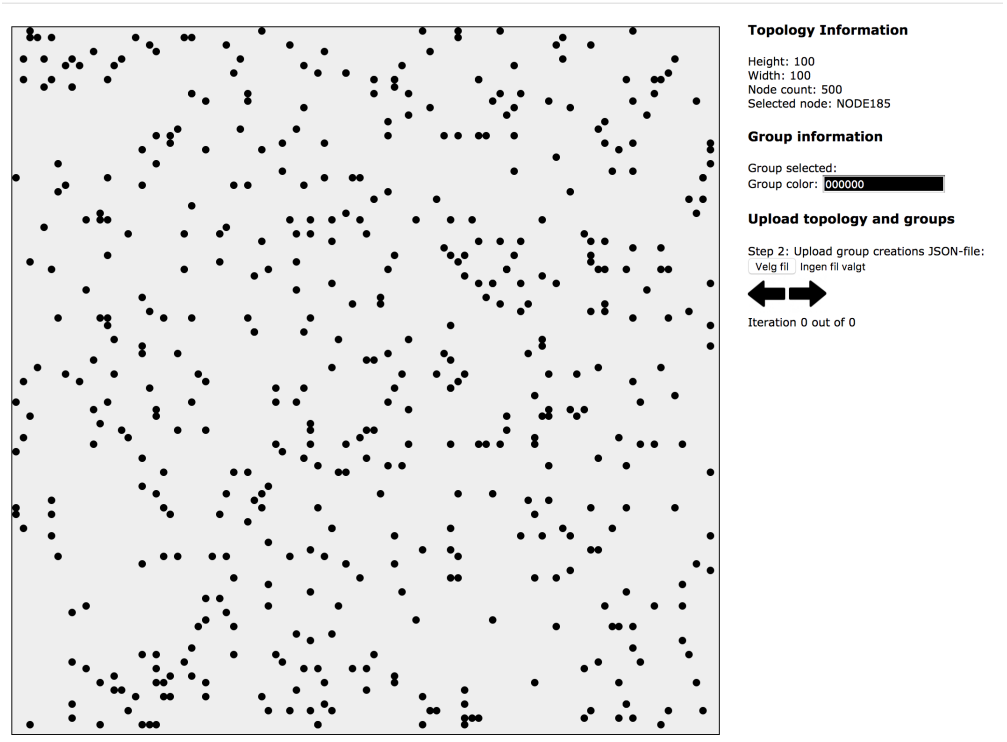

Figure 4.3: Generated topology with random, uniform distribution and the interface for viewing topologies

the app for wardriving, [2], then submit the data to WiGLE's centralized database. The APs discovered can be viewed on an interactive map provided on WiGLE's website. All the data can also be accessed through a public API which is especially interesting for us. Using their service is entirely free, but the amount of data that can be requested is throttled on a day-to-day basis. In the FAQ section on the website its written that the project openly supports research projects, so after contacting them they upgraded the account we will be using for the data requests to a higher daily data quota.

As the physical location of each access point is estimated using weighted-centroid trilateration, we can fetch the estimated position of each access point. However, the location data is not necessarily always accurate: if the measurements of an access point signal strength is done in multiple places (in a way that a line between the measurement locations surrounds the access point), the access point location becomes very precise. On the other hand, if the measurements are taken on only one side of the access point, the

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

```
https://api.wigle.net/api/v2/network/search?first=0&latrange1
    =37.80846&latrange2=37.7467&longrange1=-122.5392&longrange2
    =-122.3813&start=0
```

Figure 4.4: Example of a Wigle API request

measurements are heavily shifted towards that side. figure?

Even though the data is not perfect, it is more suited to impersonate the layout of real-life network topologies and give us a better understanding of how what access point clustering would look like in real-life.

4.4.1 REST API

WiGLE has a REST API that can be used to retrieve information from their database. The API responds to requests with JSON data. It gives access to a number of services such as user profile management, large-scale statistical information about the collected data and more specific network searches. An interactive guide to the public API is available at [19].

We are going to mine data for the topology generation program, hence the network search service is all we are interested in. There are several ways an access point can be retrieved using the network search. We can query specific SSIDs, a range of coordinates, only APs with a minimum signal level, or only query networks that are free to use. As we are only rebuilding the network topology to fit our tailored data structure, we will consider all access points and not filter on any parameters except for coordinates. By issuing an API a request for nodes between two latitudes and two longitudes, the response is an array with AP objects within that area. A request will look something like what can be seen in figure 4.4.

The parameters $latrange1$ and $longrange1$ are the coordinates that marks the beginning of the area we are interested in and $latrange2$ and $longrange2$ marks the end. We want information about the position of all access points within this range, however WiGLE returns at most 100 results per query. If there are more than 100 access points, we need to change the *start* parameter. This parameter tells WiGLE at which offset we want to begin fetching data from. For instance a start value of 0 means we fetch the first 100 access points, with indexes 0-99. A value of 100 means that we fetch the next 100 APs in the range $100 - 199$ and so on. The JSON response for a succesful request for one AP can be seen in figure 4.5.

The JSON-object in the response contains quite a lot of information about all of the APs retrieved. We wont be using most of this data, but it is worth noting that some of it could be used to perfect the search. For instance the

```
{
\subsection{Graph history}
        "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 4.5: REST API response with AP data

last updated parameter could be a way to refine the access points retrieval, so that access points that has been long gone is not taken into consideration. The most valueable properties for us is *trilong* and *trilat*. As the names suggest these contain the position calculated by the weighted-centroid trilliteration.

4.4.2 The Haversine Formula

As seen in the previous section, WiGLE provides data about the location of access points and a way for us to retrieve this data in a usable format. At this point we could create a program that operates directly on the longitudes and latitudes retrieved. This program would also need to be able to use the FPSL formula on global coordinates to compute the radio scans of each AP, and to properly visualize the data we would have to reinstall the nodes on a globe. This implementation requires a lot of extra labour, and seems especially redundant when we already have a working program that already has the aforementioned functionality. The problem is that our previous work relies on a two dimensional Cartesian coordinate system. To be able to reuse what we have have already built, the geographic coordinates has to be translated into coordinates in Euclidean space.

The haversine formula [16] is used to accurately compute the great-circle distance between two global coordinates.

$$d = 2Rsin^{-1}\left(\sqrt{\left(sin\left(\frac{\varphi_2 - \varphi_1}{2}\right)\right)^2 + cos(\varphi_1) * cos(\varphi_2) * sin\left(\left(\frac{\lambda_2 - \lambda_1}{2}\right)\right)^2}\right)$$

Where d is the distance between two latitudes φ_1 and φ_2 and two longitudes λ_1 and λ_2. R is the radius of the sphere, which in this context is the earth's radius.

This can also be expressed with a two-parameter inverse tangent function [4], as long as neither of the parameters are zero.

$$\begin{aligned} a &= \left(sin\left(\frac{\varphi_2 - \varphi_1}{2}\right)\right)^2 + cos(\varphi_1) * cos(\varphi_2) * sin\left(\left(\frac{\lambda_2 - \lambda_1}{2}\right)\right)^2 \\ c &= 2 * atan2(\sqrt{a}, \sqrt{(1-a)}) \\ d &= c * R \end{aligned}$$

The python implementation of the formula can be seen in figure 4.6. It is imortant to note that the degrees taken as input has to been converted to radians before inserting it in the formula.

```
def distanceBetweenCoordinates(lat1, lat2, long1, long2):
  deltaLon = math.radians(long2 - long1)
  deltaLat = math.radians(lat2 - lat1)
  a = (math.sin(deltaLat / 2))**2 + math.cos(math.radians(lat1)) * math.cos(
      math.radians(lat2)) * (math.sin(deltaLon/2))**2
  c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
  R = 6371000
  d = round(c * R)
  return d
```

Figure 4.6: Implementation of haversine distance

This can be used to get the distance in meters between the boundaries of our area of interest. When the data is retrieved from WiGLE we can use the specified coordinates to compute the size of the cartesian coordinate system. The distance between the first latitude φ_{start} and the second latitude φ_{stop} is calculated. This is done by using the same longitude $\lambda_1 = \lambda_2$, and returns the size of the x-axis in meters. The opposite has to be done to get the size of the y-axis, using λ_{start} and λ_{stop} and equal longitudes. The same approach can be used to place each node on the coordinate system, where the first set of the coordinates is φ_{start} and λ_{start}, and the second set is the coordinates of the AP.

4.4.3 Data output

The resulting python program **??**takes an output filename and two latitudes and two longitudes as input. It queries WiGLE's API for all the APs between these coordinates. As long as the number of APs in the coordinate range does not exceed the daily data limit, all the data will be stored in a topology datastructure imported from the topology generation program in section 4.2. The APs will be placed correctly relative to each other, with real world distance between them, based on the coordinates of each AP.

wigle data figure, overlaid map

Chapter 5

Access point clustering

5.1 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.

5.2 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.

5.2.1 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, 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.

5.2.2 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 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.

5.3 Computation assumptions and requirements

Moving forward in the next chapters we will be looking at possible ways for nodes to organize themselves into groups, only depending on peer-to-peer information sharing. We will be using the data we fetched in chapter 4. The data provides a topology of nodes where all nodes have a list of neighbours. This neighbour list contains the appropriate computed RSSI-readings for how loud each neighbour 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 create groups that resembles clusters of nodes. It is important that the border between two different groups are placed in such a way that interference between the two groups are as minimal as possible, and this also a metric for how we evaluate the performance of different algorithms. To simplify the computation procedure and to reduce the workload for building the simulation program we are going to to a number of assumptions.

1. Assumption 1: All nodes involved in the simulation also runs the group creation algorithm. This is an assumption needed to be able to see how well the group performs in an optimal condition. Note that 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 environment.

2. Assumption 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). This assumption also implies that there is implemented a protocol that lets nodes communicate and exchange information about their RSSI-readings and group membership.

3. Assumption 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. Sharing information with the group is instant.

5.4 Program design

5.4.1 Design choices

The program is designed to be modified so it can accomodate different algorithm types without changing the fundamental framework. Everything regarding group computation is implemented in Python 3 [7], and is designed parse the output from the data generation program and use it directly. Seperating these programs is important, as group computation should be a 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 the computation should be run in parallel or in sequence.

If the computation runs as parallel processes, it could assign one thread to each node and then let the nodes interact through a public interface. This is in many ways similar to simulating a peer-to-peer protocol interface. Comparing it with a sequential implementation, a parallel computation could arguably be a better representation of the real world, as the nodes act independently of each other and the order of events is to some degree random. It is exactly because of this chaotic nature of parallel computation the program was designed to run in sequence. The benefit of a sequential implementation is consistent and recreateable results, which outweighs the benefits of a parallel implementation. It also reduces the complexity of doing operations on groups memberships, since we don't have any race conditions. This particular point is also accounted for in computation assumption 3, where we assume that all nodes have the exact same state. Given a perfect algorithm that always finds the best division of groups, we could argue that 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.

5.4.2 Group framework

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 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 nodes, removing

nodes or merging the group with another. This is the part of the code that has to be changed when implementing and changing algorithms. If an action was performed the method returns 1, else it returns 0.

- **GroupCollection**. An object of this class contains all the groups used in a simulation. Its main functional responsibility is looping through all the groups and calling the `iteration` method of each group 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.

flowchart

5.4.3 Output file structure

As previously mentioned the results are writen to file. The results does not only contain the resulting divison of groups, but to be able to recreate the simulation visually, the results contains the topology of all nodes and their group membership for each iteration in the simulation process. This means that we can step-by-step recreate the simulation. This can of course also be achieved in realtime by the computation framework using the Python library `matplotlib`, but there is a value to being able to separate the processes of creating groups and visualizing them. The data is stored as JSON, and the structure can been seen in figure 5.1. Having the data stored as JSON means that the data is language independent and this allows us to either implement a python parser for the data and use matplotlib as suggested, or we can use another applicaton to visualize the data. Since we already have the topology visualizer written in HTML and JavaScript from chapter 4, we can extend this program to let us upload a group creation output file.

```
{
  "iterations": {
    "0": {
      "0": {
        "groupName": "GROUP0",
        "members": {
          "0": "NODE0"
        },
        "memberCount": 1
    },
      "1": {
        "groupName": "GROUP1",
        "members": {
          "0": "NODE1"
        },
        "memberCount": 1
      }
  },
    "1": {
      "0": {
          "groupName": "GROUP1",
          "members": {
            "0": "NODE0",
            "1": "NODE1"
        },
        "memberCount": 2
     }
  }
}
```

Figure 5.1: Group simulation file structure

This particular simulation had two iterations. In the first iteration there were two groups, each with one member node. In the second iteration the two groups has merged to one group, now containing both nodes.

5.5 Basic Neighbour Clustering

The basic approach we will consider first is a very intuitive approach, and is a natural starting point. A short way to describe it is that each group always seeks to merge with the neighbouring group that contains the node which disturbs the group the most.

5.5.1 Procedure

As a way of boostrapping the group creation procedure, each node begins by identifying itself as a member of a group that only contains itself. Let us call this group a. Group a looks at all the radio readings of every member of the group, 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 in the first iteration this node's radio readings alone will decide which group to merge with. The

neighbour node, which we will call B, is the node that disturbs group a the most. B is a member of group b. As this neighbour is the most disturbing neighbour of the entire group a, it means that this is the most crucial node to collaborate with. In other words, group a wants to merge with group b to create a larger group that contains node B.

A merge happens in the following way: the members of the two groups exchange information about all their member nodes and their radio readings and combine the information. As the data is now identical for all the members of both groups and they can make identical choices it means that they are part of the same group.

In our simulation this is as easy as combining two Group objects into one. In an implemented scenario or in a test bed the procedure would be a little more complicated, and we will cover that later.

We can not always accept merges, else we would end up with a group that spanned the entire topology. That is why we define a maximum threshold for the amount of member a group can have. If the sum of members in two groups that wants to merge exceeds a predefined threshold, the merge is aborted and no changes is reported to have happened for either group. This means that the algorithm converges after a while, when all groups are too large to merge.

5.5.2 Results

The results of the computations on the different topologies using this appraoch can be seen in figures 5.2 and 5.3.

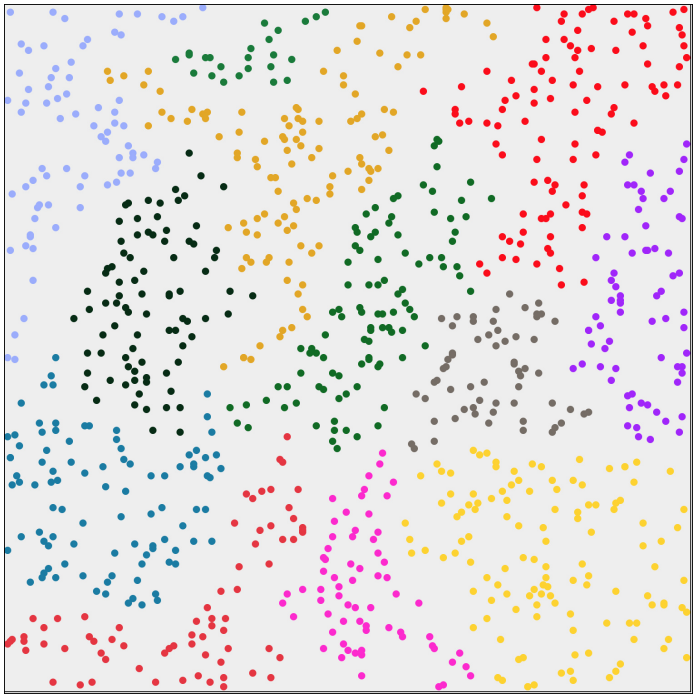

Figure 5.2: Groups in a uniform distribution. Basic Appraoch

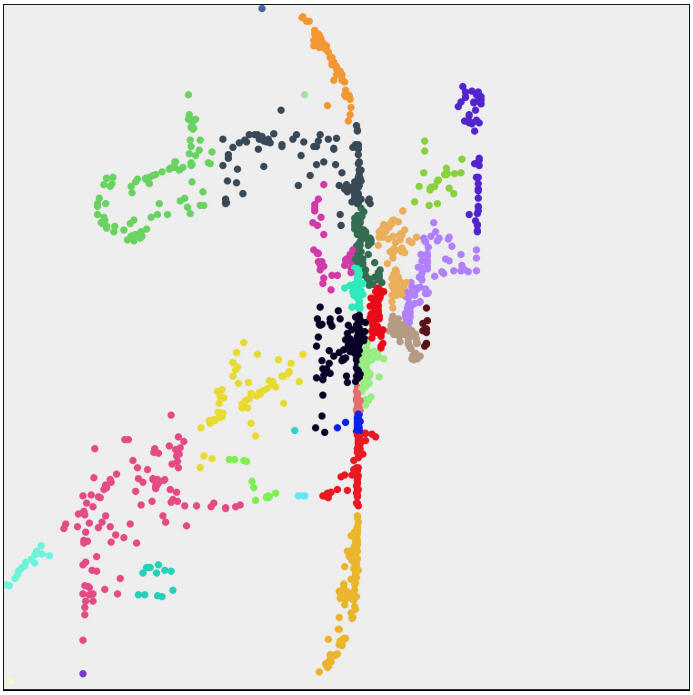

Figure 5.3: Basic approach. Groups in Forks, Washington

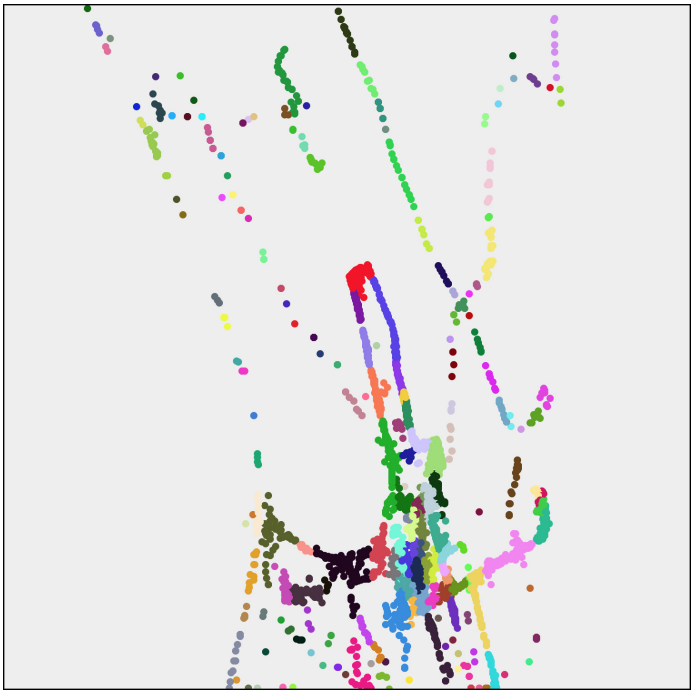

Figure 5.4: Basic approach. Groups in Lillehammer, Norway

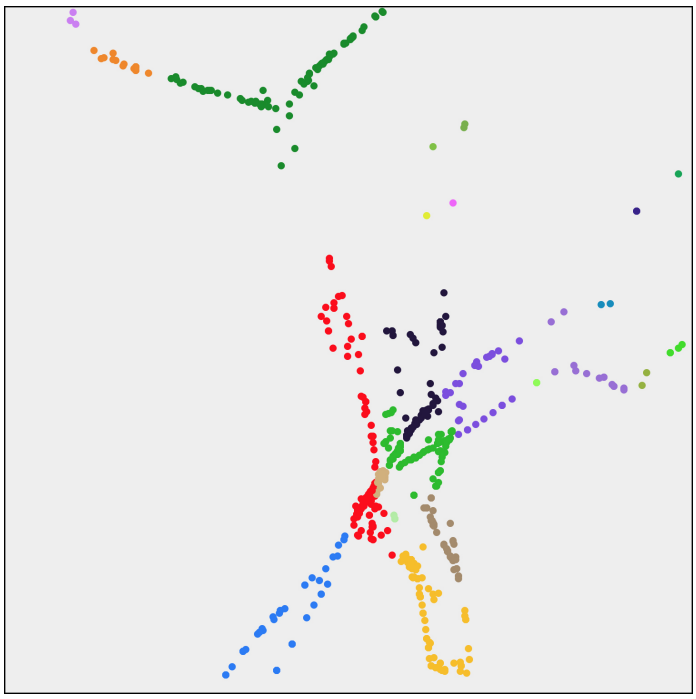

Figure 5.5: Basic approach. Groups in Lillehammer, Norway

5.6 K-means clustering

5.6.1 Introduction to K-means

K-means is a well known clustering algorithm often used for classification purposes in machine learning, usually in the context of unsupervised learning. K-means is an iterative algorithm designed to find a predefined amount of clusters in a data-set with unclassified data points. The amount of clusters to find is denoted by K (hence K-means).

The principle of K-means is rather simple. K centroids are randomly placed somewhere in the data set, then the euclidean distance between each data point and each centroid is computed. All data points closest to the same centroid becomes part of the same cluster set. When all data points are associated with a cluster, each cluster computes the mean position of all its data points. The value of the mean position becomes the location of the new centroids. When all the new centroids are computed one iteration is over, and the algorithm continues until the position of all centroids remains unchanged after an iteration. This means that the solution has converged and all K-clusters has been identified.

A well known problem with K-means is that the result strongly depends on how the algorithm is initiatied. This problem is usually dealt with by running K-means several times and choosing the best result of all the runs.

5.6.2 K-means splitting

K-means clustering is not directly applicable to solve our clustering challenge. We will consider whether it can be used in combination with the previous algorithm to create more interesting results or not. Indeed, a vanilla implementation of K-means requires an extensive overview of the surrounding network topology. Our groups are limited to knowing about their members and their neighbours. It would also be hard to choose a suitable K. Nontheless, K-means could still potentially help us create better groups. Let us consider the following. We extend the basic approach so groups are allowed to merge into a transient group if it exceeds the given maxsize. The purpose of the large transient group is to identify the biggest gap between nodes, and split the group in two new groups with $count(nodes) < maxsize$. If we set $K = 2$, K-means can be used to identify where the transient group should be split to create two more connected clusters.

Randomly picking two nodes to be centroids could lead to undesirable results where the two resulting groups are fundamentally different from the original ones. Recall that all we are trying to do is to reevaluate the divison

line between the groups and see if K-means can achieve a better split. We can consider using the two negotiating nodes as centroids. The negotiating nodes belong to different groups, and are the nodes that has observed eachother as the strongest interferer and are the nodes that are closest to the division lines between the group. If the distance between the most interfering nodes is lower after applying the K-means split, the new groups are applied. On the other hand, if the distance is shorter (meaning a less optimal cut is found) the original groups are restored. Because the bootstrapped position of the centroids is planned and not random, the result is deterministic and we don't have to run the algorithm more than one time to get the result.

5.6.3 Results

Figure 5.6: K-means splitting. Groups in a uniform distribution

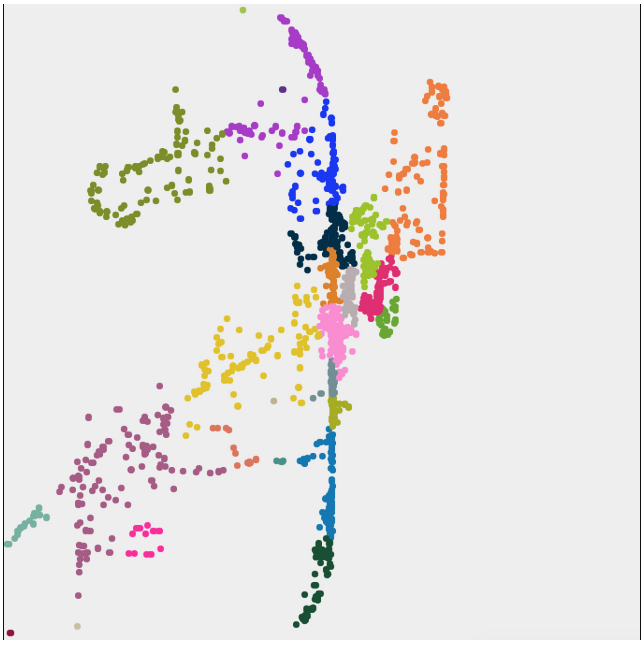

Figure 5.7: K-means splitting. Groups in Forks, Washington

Evaluation

5.6.4 Minimum Cut

5.7 Analysis

5.8 Evaluation program

5.8.1 Acquired knowledge

5.8.2 Weaknesses

The computations have taught us about what kind of clustering could work with the assumptions we have set. Even so, there are several questions that still remains, and without considering them we can not expect to get any concluding answers.

Dimensionality

Both the data we have created and the data fetched from `wigle.net` takes the x and y axes in account. Obtaining data in three dimensions is harder. Wigle positions the access points using longitude and latitude, and the simple

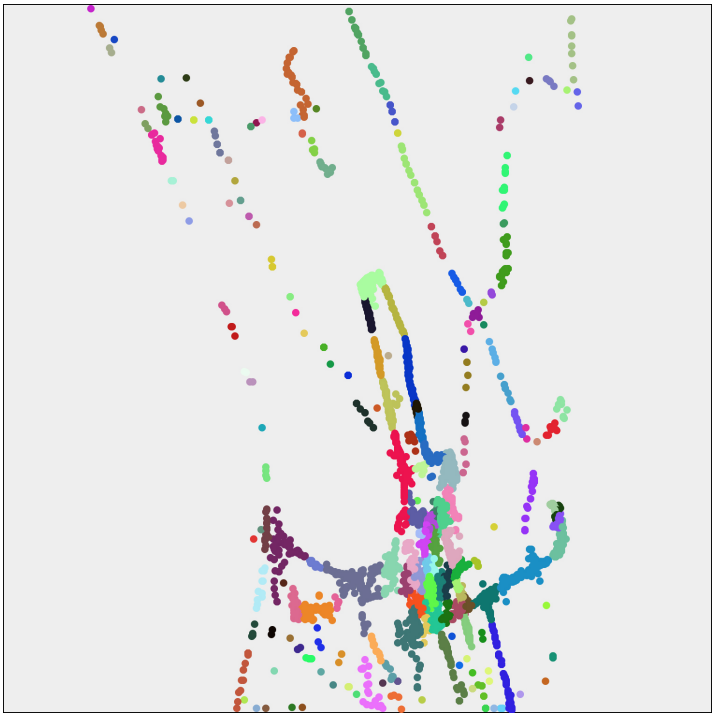

Figure 5.8: K-means splitting. Groups in Lillehammer, Norway

reason for this is that all their data is based on ground-level triangulation. As we are are only creating clusters based on information in two dimensions, we can not immediately assume that our results is applicable to 3-dimensional world. We could argue that it is resonable to think that a group creation algorithm that works well in two dimensions should also work well in three dimensions. Especially since our computations has only used a virtual dBi value, and as we recall from the beginning this is a property calculated using the free space path loss formula. This formula takes only the distance between each node as a variable. Nonetheless, other challenges may occur in 3 dimensions that we have not accounted for. For instance, the positioning of the nodes may we much denser than anticipated and its hard to estimate how that will impact the formation of groups.

Volatility

All the computations we have done takes a static network topology as an input. The static topology is an image of a network topology in a given state, and is not representative of the volatile nature of modern networks. While it is true that most access points are relatively stationary and constant, it is

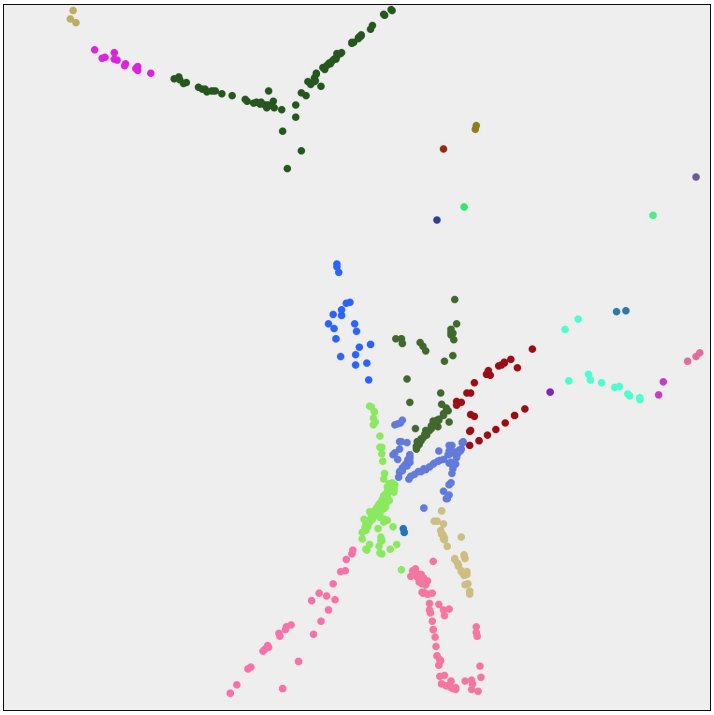

Figure 5.9: K-means splitting. Groups in Tynset, Norway

increasingly popular to create ad-hoc on-demand Wi-Fi networks using cell phones or a computers as access points. Additionally resetting of routers, power outages, etc. means that access points come and go. Hence it is indisputable that network topologies are volatile, and the group clustering algorithm should, when final, run as a continous operation.

Computed interference vs observed interference

Our group computations has been based on the RSSI of all nearby access points, and the dBi values that denotes all RSSIs were calculated using the free space path loss formula. As distance is the only variable that affects the result of our RSSI calculation, it means that all RSSI levels forms a symmetric binary relation between nodes. In the real world this is not the case. Access points may interfere with each other differently, and the RSSI values may change when there are variations in the environment. We have assumed that there is always another access point that disturbs the most. If the RSSI values changes rapidly, so may also the access point that disturbs the most.

Chapter 6

Future Work

6.1 ResFi

6.2 Raft

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] H. Balbi et al. 'Centralized channel allocation algorithm for IEEE 802.11 networks'. In: *2012 Global Information Infrastructure and Networking Symposium (GIIS)*. Dec. 2012, pp. 1–7. DOI: 10.1109/GIIS.2012.6466657.

[3] 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.

[4] Robert G. Chamberlain. *Computing Distances*. Online: accessed 14-Nov-2017. 1999. URL: https://cs.nyu.edu/visual/home/proj/tiger/gisfaq.html.

[5] Cisco. *Radio Resource Management under Unified Wireless Networks*. Accessed 16th Nov, 2017. May 2010. URL: https://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/71113-rrm-new.html.

[6] 2017 Cisco VNI. 'Cisco Visual Networking Index: Forecast and Methodology, 2016–2021'. In: (June 2017). URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html.

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

[8] H. T. Friis. 'A Note on a Simple Transmission Formula'. In: *Proceedings of the Institute of Radio Engineers* 34.5 (May 1946), pp. 254–256. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1946.234568. URL: http://dx.doi.org/10.1109/JRPROC.1946.234568.

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

[10] 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.

[11] '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.

[12] *The JSON Data Interchange Format*. Tech. rep. Standard ECMA-404 1st Edition / October 2013. ECMA, Oct. 2013. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[13] 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.

[14] 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.

[15] Aerohive Networks. 'Radio Resource Management in HiveOS'. In: (2011). Technical report. URL: http://docs.aerohive.com/pdfs/Aerohive-Solution_Brief-RadioResource_Management_in_HiveOS.pdf.

[16] Roger W Sinnott. 'Virtues of the Haversine'. In: *Sky and Telescope* 68.2 (1984), p. 158.

[17] 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.

[18] Sven Zehl (szehl). *ResFi*. https://github.com/resfi/resfi. 2016.

[19] WiGLE. *WiGLE API*. Nov. 2017. URL: https://api.wigle.net/swagger.

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

[21] 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.

[22] Sven Zehl et al. 'ResFi: A Secure Framework for Self Organized Radio Resource Management in Residential WiFi Networks'. In: *17th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2016)*. accepted for publication. Coimbra, Portugal, June 2016. URL: http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/2016/zehl16resfi.pdf.

Appendices

Appendix A

Data generation

A.0.1 GenerateTopology.py

```
class Node:
  _ssid = None
  _channel = None
  _neighbours = None
  gg_neighbourMembers = None
  _freq = 0
  _constant  = -27.55
  _minimumInterference = 0
  group = None
  name = None
  x = 0
  y = 0

  def __init__(self, posx, posy, n, freq, thresh, name = None):
    self._neighbours = []
    #Data structure
    self.edges = {}
    self._minimumInterference = thresh*-1
    self.x = posx
    self.y = posy
    self._freq = freq
    if (name == None):
      self.name = "NODE" + str(n)
    else:
      self.name = name
    self._ssid = self.name

  def __hash__(self):
    return hash(self.name)

  def rssiNeighbour(self, node):
    for n in self._neighbours:
      if n["ssid"] == node.name:
      return n["dbi"]
    return None

  def distanceTo(self, node):
    x = node.x - self.x
    y = node.y - self.y
```

```
    return math.sqrt(x**2+y**2)

  def getNeighbourCount(self):
    return len(self._neighbours)

  def getMostDisturbing(self):
    highest = -100
    nodeinfo = None
    for n in self._neighbours:
      if n["dbi"] > highest and n["obj"].group.name != self.group.name and
          not n["obj"].group.locked:
      nodeinfo = n
      highest = n["dbi"]
    return nodeinfo

  def getLeastDisturbingCompanion(self):
    lowest = 100
    nodeinfo = None
    for n in self._neighbours:
      if n["dbi"] < lowest and n["obj"].group.name == self.group.name:
      lowest = n["dbi"]
      nodeinfo = n
    return nodeinfo

  def getDBSum(self):
    sum = 0;
    for n in self._neighbours:
      sum += n["dbi"]
    return sum

  def calculateInterferenceTo(self, nodeObject):
    if self == nodeObject:
      return
    dist = round(self.distanceTo(nodeObject))
    if (dist == 0):
      dBi = -40
    else:
      dBi  = self.measureDbi(dist)*-1
    if (dBi > self._minimumInterference):
      self._neighbours.append({"ssid": nodeObject._ssid, "dbi": dBi, "obj:":
          nodeObject})

  def getData(self):
    data = OrderedDict()
    data.update(posX = self.x)
    data.update(posY = self.y)
    data.update(frequency = self._freq)
    data.update(ssid = self.name)
    data.update(neighbourCount = len(self._neighbours))
    data.update(neighbours = {})
    for i in range(len(self._neighbours)):
      data["neighbours"].update({i : {"ssid" : self._neighbours[i]["ssid"],
                                      "dbi" : self._neighbours[i]["dbi"]}})
    return data

  def measureDbi(self, dist):
    return (20 * math.log(self._freq, 10)) + (20 * math.log(dist, 10)) +
        self._constant

  def __str__(self):
    return self.name
```

```
  def __repr__(self):
    return self.name

  def __unicode__(self):
    return self.name

class Topology:
 _map = {}
 _width = None
 _height = None
 _spacing = None
 _nodes = []
 _nodesDict = {}
 _nodeCount = 0
 _frequency = 2437
 _thresh = 0
 _minimumRadiusVectors = None
 premadeNodes = None

 def __init__(self, width, height, spacing, nodeCount, dbThresh,
     premadeNodes = None):
  self._thresh = dbThresh
  self._width = width
  self._height = height
  self._spacing = spacing
  self.premadeNodes = premadeNodes
  if (premadeNodes != None):
    self._nodeCount = len(premadeNodes)
  else:
    self._nodeCount = nodeCount

 def newTopology(self):
  self.createMinimumRadiusVectors()
  self.populateMap()
  self.measureInterference()

 def getNodeCount(self):
  return self._nodeCount

 def getNodes(self):
  return self._nodes;

 def initMap(self):
  print("Generated topo")
  print("Generated filled topo with None objects")

 def measureInterference(self):
  print("> Calculating interference between all nodes.")
  i = 0
  printPercentage = 5
  percentageMark = len(self._nodes) / (100 / 5)

  for nodeSubject in self._nodes:
    if (i % percentageMark == 0):
    print(" *", i, "of", len(self._nodes), "nodes done.")

    for nodeObject in self._nodes:
    nodeSubject.calculateInterferenceTo(nodeObject)
    i += 1

 def createNode(self, posx, posy, nodeNumber, nodeFreq, nodeDbiThresh, name=
     None):
```

```
 node = Node(posx, posy, nodeNumber, nodeFreq, nodeDbiThresh, name=name)
 try:
   self._map[posy][posx] = node
 except KeyError:
   self._map[posy] = {}
   self._map[posy][posx] = node

 self._nodes.append(node)

def generateRandomNodes(self):
 nodeCount = 0
 for i in range(self._nodeCount):
   while 1:
   y = random.randint(0, self._height - 1)
   x = random.randint(0, self._width - 1)
   if self.isPositionAvailable(x, y) == True:
     self.createNode(x, y, nodeCount, self._frequency, self._thresh)
     nodeCount += 1
     break

def placeExistingNodes(self):
 nodeCount = 0
 print("> Placing nodes in topology.")
 for n in self.premadeNodes:
   self.createNode(n['x'], n['y'], nodeCount, self._frequency, self._thresh
       )
   nodeCount += 1
 print("> Nodes placed.")

def populateMap(self):
 if self.premadeNodes == None:
   self.generateRandomNodes()
 else:
   self.placeExistingNodes()

def isPositionAvailable(self, testx, testy):
 for pos in self._minimumRadiusVectors:
   x = testx + pos[0]
   y = testy + pos[1]
   node = None
   try:
   node = self._map[y][x]
   except KeyError:
   return True
 return False

def createMinimumRadiusVectors(self):
 """Creates a list of relative positions to a node, where
 no other node can be placed because of the minimum spacing
 between nodes."""
 positions = []
 for i in range(-self._spacing, self._spacing + 1):
   for j in range(-self._spacing, self._spacing + 1):
   dist = math.sqrt(i**2+j**2)
   if dist <=  self._spacing:
     positions.append((i, j))
 self._minimumRadiusVectors = positions

def printTopology(self):
 for y in range(len(self._map)):
   print(self._map[y])
```

```
 def writeData(self, outfile):
  data  = OrderedDict()
  data.update(mapWidth = self._width)
  data.update(mapHeight = self._height)
  data.update(nodeCount = self._nodeCount)

  allnodes = {}
  for i in range(self._nodeCount):
    allnodes.update({i : self._nodes[i].getData()})

  data.update(nodes = allnodes)
  j = json.dumps(data, indent=2)
  f = open(outfile, "w")
  f.write(j)
  f.close()
  print("> Topology data written to file: ", outfile)

def main():
 args = parseOptions()
 print("Width: ", args.width, " height: ", args.height, "spacing: ", args.
     spacing)
 topo = Topology(args.width, args.height, args.spacing, args.nodes, args.
     thresh)
 topo.newTopology()
 topo.writeData(args.output)
```

A.0.2 WigleData.py