Group creation in a collaborative P2P channel allocation protocol

Identifying connected groups of access points

Hans Jørgen Furre Nygårdshaug

Thesis submitted for the degree of
Master in Informatikk: programmering og nettverk
60 credits

Institutt for informatikk
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2017

Group creation in a collaborative P2P channel allocation protocol

Identifying connected groups of access points

Hans Jørgen Furre Nygårdshaug

Group creation in a collaborative P2P channel allocation protocol

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

List of acronyms

AP	Access Point
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
MAC	Medium Access Control
CCA	Clear Channel Assessment
MAC	Mandatory Access Control
PHY	Physical Layer
PLCP	Physical Layer Convergence Procedure
PPDU	PLCP Protocol Data Unit

Contents

List of Figures

Chapter 1

Introduction

- Expansion of WiFI - Densely populated Areas - Increased usage - Expectations of universal wifi - Congested networks because of intereference

1.1 Problem definition

The problem is that devices operating on the same channel in the same area interferes with one another.

1.2 Motivation

Wifi is deployed in almost all residencies, and the problem is especially prominent in densely populated areas.

1.3 Method

Defining groups for running a channel allocation algorithm on

1.4 Channel allocation

To deal with the problem of channel allocation we will think of an AP as a vertex in a graph. When an AP scans its radio it can hear the strength of all nearby wireless networks measured in dBm (decibel milliwatts). This decibel value will be the value of the edge between one AP to another. With a graph expressing the wireless network topology, the problem of optimally distributing channels between APs boils down to a graph coloring problem.

The number of colors in the color problem, represents the number of non-overlapping channels in 802.11. Exactly how an algorithm can be designed to optimally distribute channels within the interfering topology is out of the scope of this thesis. However we can define some invariants that has to be true for such an algorithm to work:

1. All APs has to run the same algorithm

2. All APs must run the algorithm on the same connected group

3. Because of the complexity of the problem the algorthm must solve, the number of APs in the connected group can not be too big

Point 1 is trivial to solve or mitigate, as only APs running the algorithm will actively participate in the channel selection. A simple way to make sure that the same algorithm is used, is by having a software version that is consistently checked with the other APs in the connected group.

Point 2 and point 3 is will be the main focus of the rest of the master thesis, as these are not so easily solved.

We can define a wireless topology graph as a set of wireless APs that are grouped together and share information about their neighbours and interference levels. This set is what will now on be referred to as a *connected group.* All members of the connected group will be considered when running the channel assignment algorithm. For the connected group to have an actual impact on the quality of a network connection, it has to consist of nodes that normally disturbs each other substantially.

An ideal example of a connected group is an apartment builiding. The channel allocation protocol lets APs share information about who-disturbs-who the most in the building. Then each AP can run the channel allocation algorithm. Because they run it on the same graph, every AP will find the same optimal channel distribution throughout the building, and then switch to the correct channel.

Even though an apartment building is most likely an optimal delimination of a connected group, in reality creating such a group is a bigger challenge. As the whole channel allocation protocol is based on decentralized peer-to-peer technology, and no centralized server with access to demographical and geographical divisions exists, the protocol will have to discover suitable connected groups on its own. Moreover, when the group is created the protocol will have to replicate data so that all participants of the group has all the data required to perform channel allocation. It will also need a way to make sure that the image of the current group is consistent within all APs in the connected group.

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 Wireless technology

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

Collisions can also prevent the signal from being received correctly. 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.

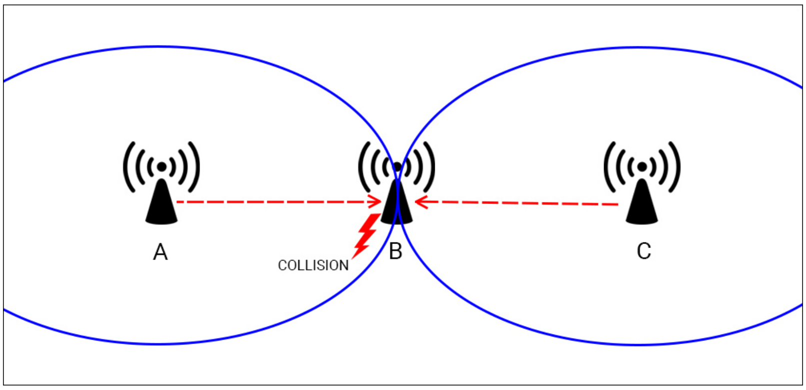

Figure 2.1: A and C sending to node B unknowingly at the same time, resulting in a collision

As each node can hear everyone on else on a cabled medium, and 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

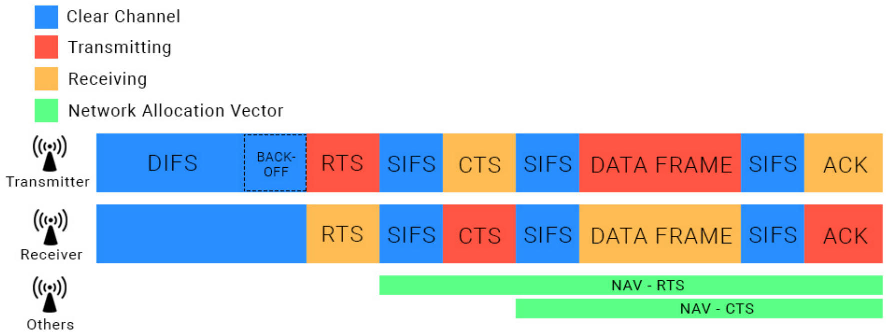

Figure 2.2: The timeline one frame transmission cycle in DCF mode

$(0, 2^n * CW_{\min})$, 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 [1].

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.

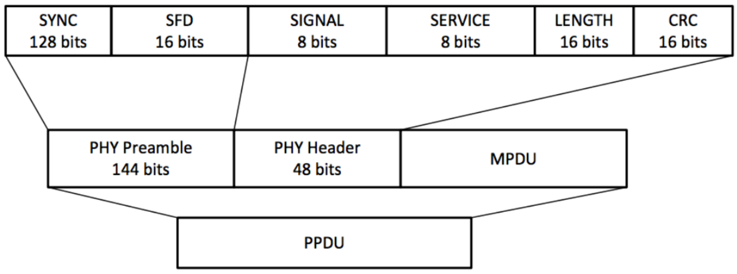

Figure 2.3: DSSS PHY PPDU format from IEEE Std 802.11-2016

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 [3].

2.5 Channels

802.11 b/g/n uses the range of frequencies from 2.400-2.500 GHz on the ISM band. The increasingly popular 801.11n/ac also uses the 5 GHz band which offers more frequencies and has higher attenuation, thus making it shorter reaching. Other than that the properties and challenges 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.

Figure 2.4: Channel/frequency distribution

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

Chapter 3

Related work

Chapter 4

Connected groups

To enable collaborative channel allocation, it is important for every AP to know which APs it is collaborating with. One way to share information about who collaborates with who, is to let the access points group together. Information relevant for channel allocation can then be shared freely within the group, between APs.

We will proceed by looking at some of the requirements for a group creation algorithm. It should work decentralized in a distributed fashion. Hence, not only does the APs have to be imposed group membership, but they also have to be able to create and definine meaningful groups on their own. Later we will propose an algorithm to create groups, and then evaluate computed groups based on the algorithm.

4.1 Definition

A connected group is a set of access points that are within close geographical range. The group should consist of APs that interfere with each other when their channels are overlapping, so that overlap can be avoided with a channel allocation algorithm run within the group. Not all APs in the connected group will necessarily be able to hear each other on the radio directly, but all nodes should be able to hear each other through a transitive relation (neighbour of a neighbour, etc.).

4.2 Requirements and assumptions

We are going to be looking at how APs can organize themselves without a central coordinator, but it should be noted that the creation of groups could potentially be easier if a centralized entity had the full picture of APs and their radio readings. It could then combine geographical information along with AP readings to create consistent groups. We will consider the two options against each other later on, but proceeding ahead with P2P group creation as the main concern. Essentially this means that an AP has no other knowledge of the world than the other APs it can hear on its radio. For the sake of creating and evaluationg a group creation algorithm, we will assume a number of things that will have to be in solved at a later point:

1. All APs visible on any AP's radio also runs the group creation algorithm.
2. If an AP can see another AP on its radio, it also knows how to directly contact the AP (e.g. via TCP).
3. All APs in a group are completely synchronous, and always have an equal image of the state of the group.

4.3 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.

4.4 Simulation data

4.4.1 Requirements for data creation

Before implementing and testing the group creation algorithm, we need to gather usable data to perform testing on. The data consists of the topology size, and where on the topology a node is located. There are two ways we can get this knowledge.

The first way is creating our own data, and either assume the location of nodes, or randomly distribute them on a topology.

The second way is getting real world data from a data source, where the topology would represent a section of a map, and the nodes would be APs, most likely located in the households of the map. Either way, the represenation of the topology should be the same.

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 the grid the node topology, or just the topology. When the nodes are placed on the grid, we can compute which nodes it can hear on the radio, and add these to the nodes neighbour 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

4.4.2 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[2], 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 [5] to compute the $-dBI$ values.

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.

4.4.3 Data output and visual representation

The 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
    }
                        }
        }
```

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

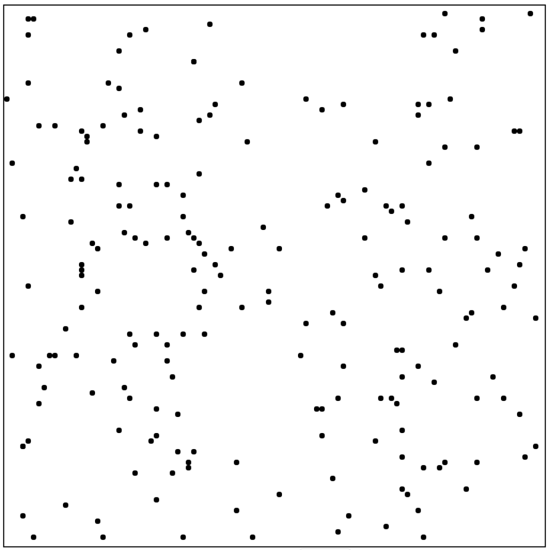

Figure 4.1: Generated topology with random, uniform distribution

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 4.1

4.5 Algorithm implementation

The algorithm was implemented in Python 3 [2], designed to run on the output from the data generation program.

For this computation, we are assuming that all members of a group are precicely synchronized and the group members are consistent across all nodes. With that assumption, we can be sure all nodes will make the same decisions.

A group object can then be responsible for keeping track of current members, represented as node objects. The group object will also look through the radio readings of all its member nodes to find new member candidates, handle merges, locking the group and kicking out extraneous members, all according to the algorithm description in section 4.3.

To make it easier to evaluate the results, the computation is done sequentially and in iterations. This is a simple way to make sure the same groups are created for the same topology every time, and it will make the evaluation process easier.

The group creation program outputs a JSON-file containing the groups and their members for every iteration. The first iteration being every node having a group for their own, and the last iteration is the resulting groups.

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 4.2 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.

The result of this computation, seen in figure 4.3, 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

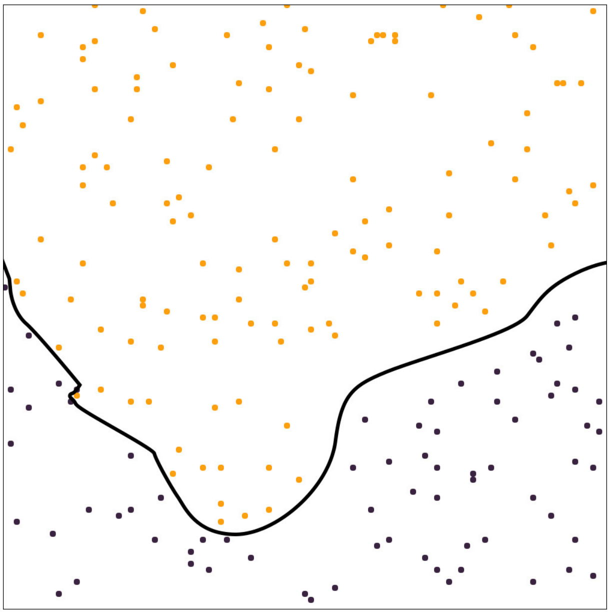

Figure 4.2: 200 nodes, $memberThresh = 128$, $size = 200x200$

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 4.4 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 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 4.4. Its member nodes has a thicker black line around them.

4.7 WiGLE

WiGLE (Wireless Geographical Logging Engine) [4] 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

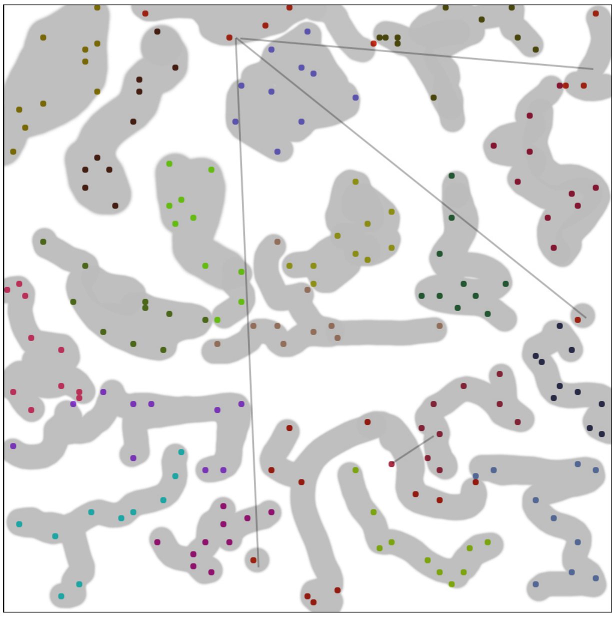

Figure 4.3: 200 nodes, $memberThresh = 10$, $size = 200x200$

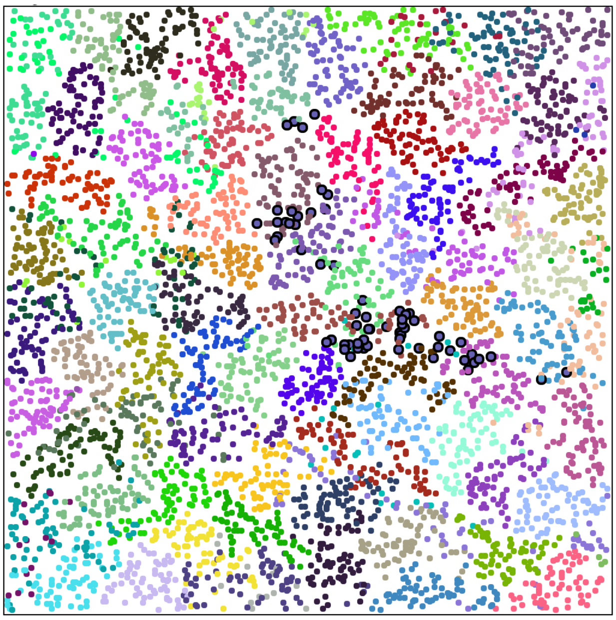

Figure 4.4: 5000 nodes, $memberThresh = 64$, $size = 2000x2000$

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.

4.7.1 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 this:

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

The parameters *latrange*1 and *longrange*1 are the coordinates that marks the beginning of the area we are interested in, where *latrange*2 and *longrange*2 marks the end. As WiGLE at most returns 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 **??**.

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

4.7.2 Using the 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

[1]Wardriving means tracking wireless networks using a laptop or a phone, and then store the information about each network [6].

```
{
        "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

a topology with the same format as the generated topologies we created in section 4.4. 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 4.4.3 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.

4.8 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 4.6. The topology is of medium density, consisting of 4990 APs, and is 2572 meters 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

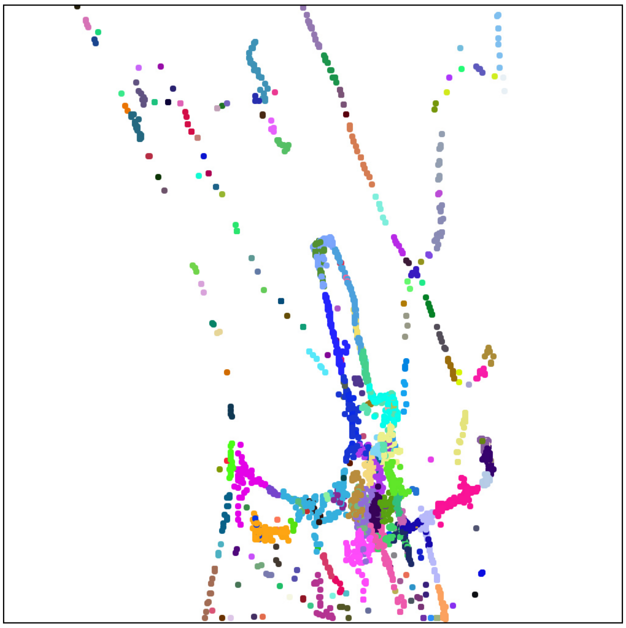

Figure 4.6: Lillehammer

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 4.7. 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.

Forks

The computation results of Forks can be seen in figure 4.8. The topology consists of 1715 nodes, and is 2122 meters high and 4495 meters wide. It is

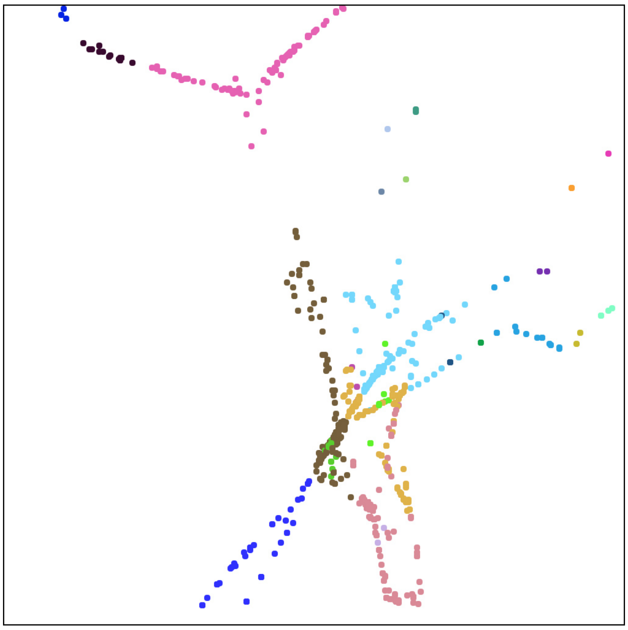

Figure 4.7: Tynset

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.

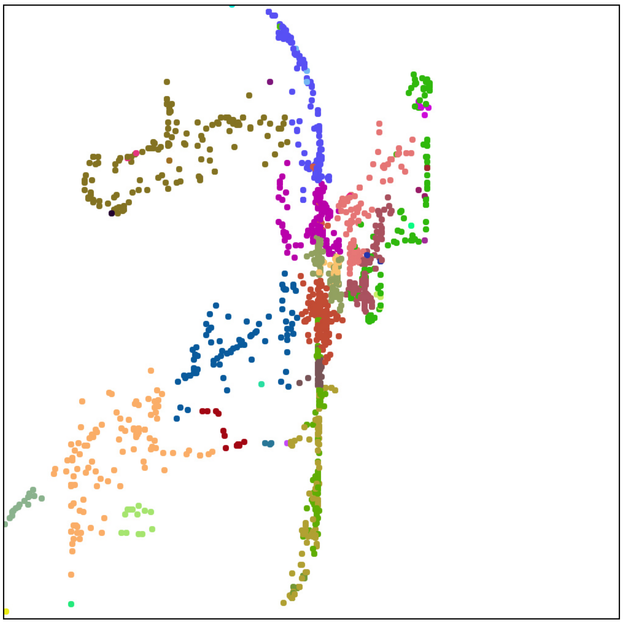

Figure 4.8: Forks

Bibliography

[1] 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.

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

[3] 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.

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

[5] 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.

[6] 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.