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, Spring 2018

Group creation in a collaborative P2P channel allocation protocol

Hans Jørgen Furre Nygårdshaug

18th April 2018

Abstract

Contents

List of Figures

Chapter 1

Introduction

The amount of Internet connected devices has for the past few years been rapidly increasing, and is still increasing this day. The latest forecasts esitmates that there will be about 27 billion devices connected to the Internet by 2021 [9], while by the end of the 2017 the number connected devices will be close to 8.4 billion [15]. In 2020 households alone will be responsible for over 10 billion devices able to wirelessly connect to the home router [2], and wireless traffic will account for 63 percent of all IP-traffic in the world [9]. Traditional devices connected to the Internet like computers, phones, watches, smart TVs, audio systems, and lately also private network storage systems is responsible for much of the traffic. However, 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 in the basement couch and the gaming laptop in the bedroom. All demands coverage and expects mobility at the same time. Many of these devices also have something else in common which reflects the change that has happened over the last few 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 [9].

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

streams have high Quality of Service (QoS) demands. Naturally, buffering of videos and/or reduction of video quality will negatively impact the experience of the service. Secondly, it is not unusual that different video flows occur simultaneously in the same household. 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 presents a largely unresolved challenge of mapping channels to access points in a way that there are only a few, or better yet no, devices on the same channels adjacent to each other.

While the physical layer 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 radio frequency intereference which can not be solved by increasing the physical datarate capacity.

There has been done research on how centralized controllers can benefit the deployment of large enterprise networks ([24], [23] and [29]), but in this thesis we will address the emerging issue of deployment of Wi-Fi in residential areas. More specifically, we will consider possible clustering methods to enable routers and access points to organize themselves in self managing groups. A synchronous, distributed group across access points in different service sets would allow for planned, cooperative channel allocation. This would transform the channel allocation problem from finding a local optimal channel for each router, to a problem of finding an optimal channel plan for the entire group.

1.1 Motivation

Wireless LAN, commonly referred to as Wi-Fi, is deployed in almost all corporate buildings and residencies in the modern world. The use of these networks used to be limited to laptops or phones that generated small amounts of data, but now the range of devices includes smart-phones, network storage devices, and IoT appliances. The deployment of Wi-Fi and the infrastructure has not changed much over the years to match the new demands and increased traffic. Actually, in most places coverage is still the main concern, while QoS comes second.

Customers 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. But as it happens, often interfering networks are the perpetrators, and an increase of bandwith will have no effect. This can lead customers to frustration with their Internet service providers, even though it is the underlying

technology and not the service provider which is at fault.

There exists a large amount of different algorithms that deal with channel allocation to prevent and limit interference. Some even consider the QoS observed by the client - not only the access points. However, many efficient algorithms are deployed in centralized systems and does not focus on decentralized instances, such as residential networks. A decentralized, distributed solution to optimize channel distribution would be helpful in apartment buildings and other residential zones where the density of wireless networks is high, as it would not require the wireless networks to be under the administration of the same controller (e.g. an ISP). The access points could communicate and organize channel allocation on their own. Of all channel allocation algorithms only a few addresses the issue of optimizing the channel distribution, but the algorithms does not have a way to limit the amount of nodes to consider. The main motivation behind this thesis is to explore ways to create a distributed group (clustering) scheme. Once a group is established, many problems can be solved with technologies previously only used in centralized systems.

1.2 Problem definition

There are 3 non-overlapping channels on the 2.4GHz spectrum utilized by 802.11 Wi-Fi. Today, one of the more 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 this selfish manner, where the only available information about the surrounding networks are obtained via local radio observations, it is highly unlikely that the channel distribution in a confined area (e.g. an apartment block) becomes optimal.

Ideally we want all access points to be configured so the channel distribution in an area leads to as few equal channels as possible being next to one another. The task of optimizing channel distribution is essentially what is called a graph-coloring problem, where the problem is finding a distribution where no adjacent nodes should operate on the same channels. This is NP-hard and can be solved with heuristics [5], but is not the focus of this thesis.

Since the problem is NP-hard, it is easier to determine a good channel distribution if fewer access points are considered. In other words, if the groups are too large, the channel distribution problem becomes too complicated to solve. This bring us to the punch line of the problem definition: finding a way to confine access points to a high-impact group. We have already established the motivation behind aiming for a decentralized solution - but to achieve this, access points have to find these groups on their own. It is easy to intuitively say that, for instance, an apartment building should be one group. It is less obvious to answer how all access points in an

apartment building would identify the boundaries of the building and create a group on their own, entirely independent of an Internet service provide (and ideally also router brand). It boils down to a distributed clustering problem, where no access point has a global view of the landscape of neighbouring routers. So we must find a way to make the clustering algorithm able to be run in a way that clusters can be built relying only on information that can be obtained from radio scans of its membering access points.

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

In this chapter the relevant aspects of wireless technology and a selection of important concepts from the 802.11 standard will be introduced.

2.1 Basic radio challenges

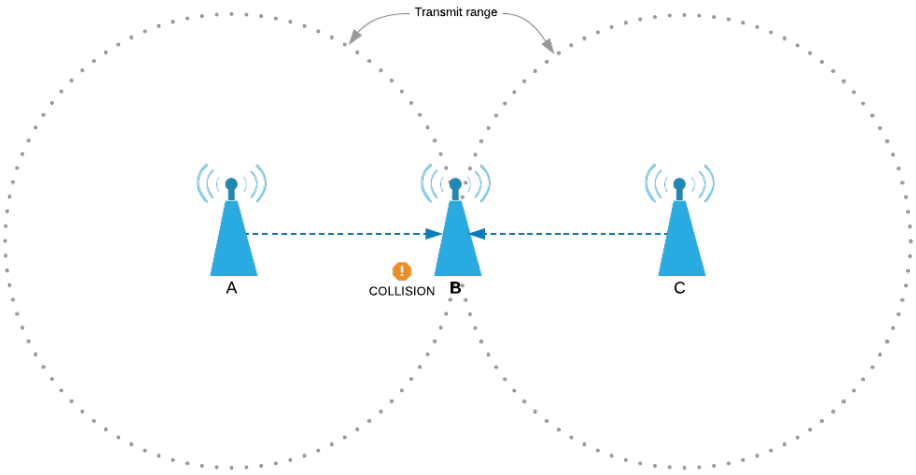
There are some challenges with wireless technologies that are harder to overcome than in wired transmission technologies like Ethernet.

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 only air, the phenomenon is referred to as free space path loss. 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

Given two radio devices A and B, if there are any 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 all other nodes on a wired medium, and listening



(a) A and C transmitting unknowingly at the same time, resulting in a collision at B

Figure 2.1: The hidden terminal problem illustrated

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, and a brief explanation of the problem will be offered here. 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 either of the two, as they have been corrupted during transmisison due to the collision.

2.2 Network infrastructure

In this section there will be a brief introduction to network infrastructures, to get an understanding of the differences between the mainly two types of infrastructures used today.

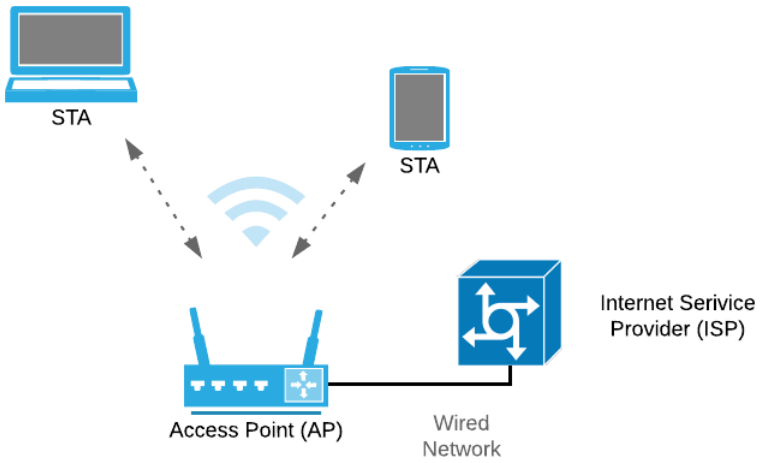


Figure 2.2: Minimalist Basic Service Set in infrastructure mode

2.2.1 Basic Service Set

A basic service set (BSS) in infrastruture mode consists of a single AP connected by wire to a distribution system (in residential deployment the distribution system is typically the ISP). Stations within the basic service area, which is the area physically serveable by the AP, can connect to the BSS using dynamic access point association, described in [10]. A minimalist BSS infrastructure is illustrated in 2.2. It also worth mentioning that a BSS can also be configured to operate as an independent basic service set, where there is no infrastructure in place (no APs), and communication is direct between stations. This is more commonly known as ad-hoc mode.

One basic service set per household is the typical infrastructure in residential networks, but it is not uncommon to have multiple basic service sets in larger homes, where APs are placed on different floors to have a better signal strength. Usually these APs have different service set identifiers (SSIDs), and each requires its own authentication. This is because the APs are not under the same extended service set.

2.2.2 Extended Service Set

An extended service set (ESS) can consist of multiple basic service sets, and thereby APs. The extended service set is a logical entity which can extend station authentication and association to all APs under the same extended service set. In an ESS the SSID of all APs are also identical. This means an STA can be in the basic service

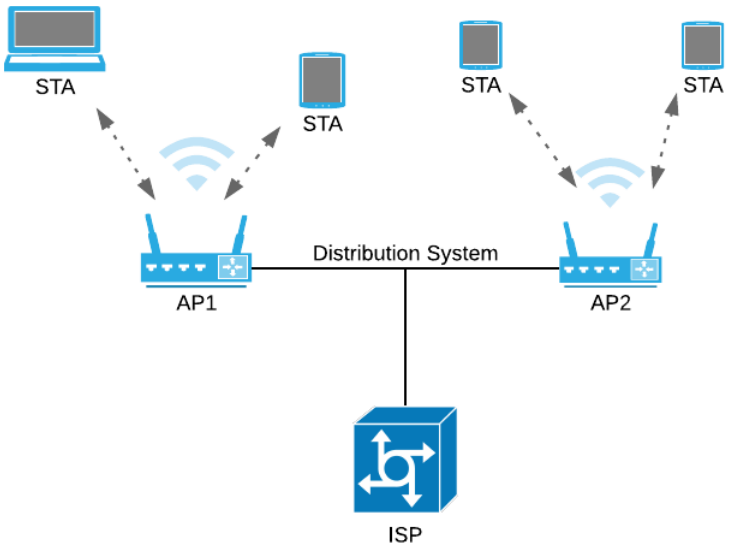


Figure 2.3: Extended Service Set

area of multiple APs at the same time, and is given the opportunity to select which AP to be serviced by. APs in an ESS can operate on different channels, and the medium access control mechanisms are performed as usual. Figure 2.3 illustrates the infrastructure of an extended service set.

2.3 MAC Layer

In this section parts of MAC layer of the 802.11 protocol is accounted for. 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 basic service set operates 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)**, both of which will be briefly covered.

2.3.1 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 request CCA from the physical layer. When a valid 802.11 frame is decoded on a listening wireless network interface, 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 refrain from transmitting and also refrain from rechecking the channel for the duration of the NAV.

2.3.2 Collision Avoidance

CSMA/CA attempts to avoid collisions and is helpful 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 $(0, 2^n * CW_{\min})$, where n is the number of times the transmission has failed, beginning at $n = 0$, and $CW_{\min} < CW_{\max}$.

2.3.3 Distributed Coordination Function

The distributed coordination function is the main mode of operation in the 802.11 MAC layer, and is supposed to provide fair and reliable transmission for all nodes

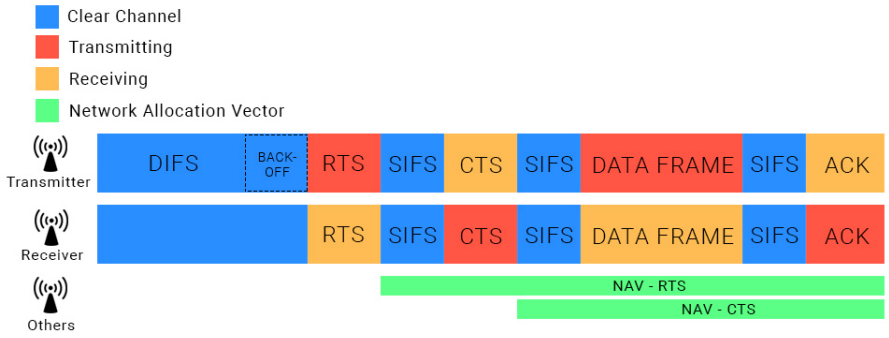


Figure 2.4: The timeline one frame transmission cycle in DCF mode

on the same network. Figure 2.4 shows the frame exchange that happens. Following the figure, 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 [4].

2.4 PHY Layer

The physical layer in 802.11 is also divided in two sublayers. The upper sublayer is the Physical Layer Convergence Procedure (PLCP), responsible for clear channel assessment 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, as well as receiving and decoding incoming frames.

2.4.1 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 2.5. 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.

- **Sync** bits are used to acquire the signal and synchronize timing.

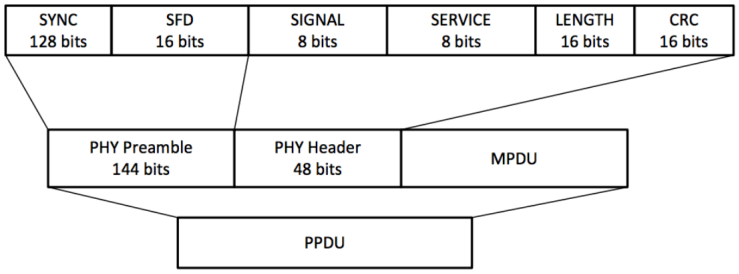


Figure 2.5: DSSS PHY PPDU format from IEEE Std 802.11-2016

- **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.

2.4.2 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 802.11 frame, but is nontheless 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 802.11 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.

2.5 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 Industrial, Scientific and Medical (ISM) band, channel noise or interference can come from sources such as microwaves, bluetooth devices and radio controllers, in addition to other Wi-Fi devices.

2.5.1 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.

Interference can thus be severly damaging for QoS on wireless networks.

2.5.2 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 together at the same time, once 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 can be selectively acknowledged. This means that only a few frames has to be retransmitted in case of corruption, and not the entire aggregation.

Additionally on the physical layer there are a couple of other suggested countermeasures:

- Changing the transmission power levels
- Lowering data rates

- Adjusting CCA threshold
- Forward error correction
- Changing packet sizes

The mentioned countermeasures only have limited impact, while switching to a channel with less interference (if available) remains the most effective in most cases [16].

2.6 Channels

802.11 b/g/n uses the range of frequencies from 2.400-2.500 GHz on the ISM band. The increasingly popular 802.11n/ac also uses a range of 5 GHz frequencies on the Unlicensed National Information Infrastructure (UNII) band, which offers more frequencies [17]. 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 2.6. 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.

Figure 2.6: Channel 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

This chapter is dedicated to give a brief account of works related to solving some of the challenges posed so far in this thesis. It is worth noting that some of the works mentioned here are actual implementations and live products, but only deployed in corporate environments.

3.1 Cisco RRM

CISCO offers a solution directed at enterprise networks [8], 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 and shares it with all APs in the same RF-group. 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 are in range of 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.

3.2 DenseAP

DenseAP described in [23] 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 a client should 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 can only use signal strength as the metric deciding which AP to associate with. This is emphasized as 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.

3.3 HiveOS

HiveOs [25], 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. The penalty per AP diminishes 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

3.4 ResFi

ResFi [33] is the only significant related work that directly aims to enable self-organized management in residential deployments of wireless LAN. Works such as [23], [8] and [25] 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. The figures of ResFi includes a Radio Resource Management Unit (RRMU). This is simply the device that interfaces with the antenna, and in most residential homes it will be a router that controls the channel and the power levels of the antenna. In short ResFi enables communication between access points under different basic service sets without imposing a central controller on the access points or being a part of an extended service set. More importantly, ResFi enables all of this without doing any modifications to hardware and drivers (like modifying standards or requiring propriatary equipment).

3.4.1 Operation

This section contains a short introduction of the way ResFi is initiated and operates.

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 establish secure point-to-point communicate with all other APs using the wired backbone, the globally routable IP-address and the cryptographic keys

3.4.2 Implementation

This section is dedicated to a brief overview of how ResFi can be implemented and deployed.

Originally ResFi was implemented 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 [30]. 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, but the north-bound API could fascilitate applications that provides services like these.

3.5 Channel allocation using DSATUR and SCIFI

This section is dedicated to previous work done using the DSATUR heuristic. A little note to this section: it would also fit well in the background section. It accounts for the fundamentally important priniciple that channel allocation can be treated as a graph coloring problem once a network graph is constructed.

3.5.1 DSATUR

DSATUR (from degree of saturation) is a heuristic created by Daniel Brélaz [5] to find solutions to the NP-complete problem of coloring the vertices of a graph so that no adjacent vertices share the same color. Channel allocation schemes relying on the DSATUR algorithm has been proposed before. In 2004 a paper was published, called "Automatic channel allocation for small wireless local area networks using graph colouring algorithm approach" [21], where the idea is to listen for neighboring AP beacon frames to create a list of all neighbours. This list would be broadcast to all the neighbours. For multi-hop support, all receiving nodes will rebroadcast the list in a fashion equal to the flooding routing mechanism. This enables routers to create a graph of access points, and the DSATUR algorithm can then be used to compute the channel distribution.

3.5.2 SCIFI

SCIFI [3] is a centralized channel allocation protocol for infrastructure Wireless LANs that improves the traditional graph-coloring algorithm DSATUR. While it shows that their central coordinator in fact improves the throughput compared to Wireless LANs that are not configured by SCIFI, the it is an algorithm for setting a channel in a preconfigured adminstrative domain. It does not deal with how the administrative domains or router clusters are defined. If the method of creating clusters of collaborating routers proposed in this thesis has merit, SCIFI could plausibly be a supporting

technology to compute channel distribution, where a group acts as the administrative domain required by SCIFI.

Chapter 4

Data acquisition and data structure

This chapter is dedicated to the creation of a program that generates and visualizes the layout of Wi-Fi network topologies based on generated data and mined data. The data will be used in chapter 5 where there will be demand for data to feed clustering algorithms. Storing the data, the data structure type, and finally how it can be visualized in a web application are also matters which will be addressed here.

4.1 Motivation

There is an undoubtful need for data when developing a distributed clustering algorithm to facilitate group creation. Surely a testbed would also be beneficial for development, but it would require a large amount of routers. Maybe 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 vast amount of equipment, but there are some unsolved logical challenges as well. Like the communication protocol between routers, and maybe distributed consensus or some way to perform synchronization to communicate group memberships. These problems are addressed in chapter 6.

Even if such challenges could be overcome, going directly from a concept idea to a large testbed in a master thesis - without having any empirical indication of which approaches have merit, may not be a good use of time. Hence, the data is gathered with the motivation of being able to perform realistic calculations and do algorithm developement on a money-, and time budget. Lastly, the motivation behind mining real data, instead of generating everything, is to enhance the quality and believeability of the results produced.

4.2 Requirements and assumptions

This section will cover the requirements and assumptions for a data generation-, and visualization program.

Background

As presented in the background chapter: SSID, channel frequency, radio power, physical data rate and supported 802.11 standards are just a few of the properties that makes up a wireless access point. Before deciding how to represent an access point in a network topology dataset, it is useful to consider the state an access point would be in when group creation is performed.

An access point is not in the transmission (CSMA/CA) state when performing channel allocation. This is simply due to the toughness of performing channel sensing without having a channel to sense. So, which sate is it in then? Well, it is hard to say without having a bird's eye view of the protocol and algorithm architecture. A fully developed protocol for distributed group-creation in residential deployed Wi-Fi is at the time of writing (and to the author's knowledge) non-existient. Nor will a full-fledged architecture be presented in this thesis (even though suggestions will be made at the end), so a couple of assumptions has to be made.

Assumptions

The 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. The major motivation for doing group creation in the first place is to be able to cleverly and collaboratively allocate channels within the group, so this state would have to occur before channel allocation is performed.

Hence, to simulate group creation and discovery, many of the access point properties does not have to be considered. Actually it is only necessary to store two things:

- Each access point's SSID, as a convenient way to provide a unique identification handle [1]
- The list of neighboring access points that can be seen through a Wi-Fi interface scan on each access point, along with the observed signal strength of these APs.

Channel frequency, radio power and data rate are properties that impacts data transmission between clients and access points, and does not need to a part of this model.

[1] In the real world this is of course not the case, but we will enforce it in the computations. Actually it could just as well be called an unique id.

In all succeeding simulations it is assumed that all nodes are transmitting with equal strength, and that the environment is flat and obstacle free.

Requirements

Unfortunately there is at the time of writing no publicly available data source that contains radio scans of a large amount of access points located in the same area. This subsection describes the requirements for the simulation program that will both generate and represent Wi-Fi network topologies. Having a clear specificiation of program requirements simplifies the programming stage and reduces the risk of feature creep.

As a basis for the simulation, access points, hereupon also referred to as nodes, should be placed on a two-dimensional grid. Each dataset has to contain a set of nodes which has two coordinates x and y, giving them a position on the grid. When all desired nodes are placed on a grid, the grid represents a network topology. It will then be possible to compute an estimation of which nodes can hear each other over the Wi-Fi radio. In other words, a virtual network interface scan. All observed *neighbour nodes* will be added to each node's neighbour list. This neighbour list contains the names of all the nodes it can hear, and the signal strength levels measured in $-dBi$.

Additionally the following parameters should be variable in the topology generation program, 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).

4.3 Program design

4.3.1 Primary functionality

The resulting 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 topology size, node

```
#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

count and minimum distance between nodes, are properties taken as input arguments to the program.

The second functionality is performing the calculation of which nodes can actually observe each other over the radio, and is described below.

Signal strength calculation

All neighbouring nodes observed by a node is added to its list of neighbours. The neighbour list contains the signal strength in $-dBi$ and the SSID of each neighbour. The interference levels between access points are calculated by iterating through every access point. For each node N its x and y position is recorded, and then a second iteration through the nodes is initiated, resulting in a complexity of $O(N^2)$. For each node n in the second iteration the distance d in meters between N and n using Euclidean distance is calculated.

To compute the signal strengths of which the nodes can observe each other, the formula for isotropic antennas, described by Friis [14], can be used to derive the formula for the Free Space Path Loss (FSPL) in dB.

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

Where $d = distance$, $f = frequency$ and $c = constant$. The constant c is used to account for different units. Meters is used to denominate distance in the program,

and megahertz for the frequency. The resulting formula implemented in the program is then:

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

Pseudocode for the interference calculation can be seen in figure 4.1.

Program result

The resulting program, written in Python 3 [13], contains an importable *topology class*. This way, for further testing different data sources can be used to get the positions of nodes, and only let the topology class compute the list of neighbours and standarize the data structure of a topology.

The program can be run in the following way

```
./GenerateTopology.py -n 500 -w 100 -h 100 --space 10 --dbi 80
```

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 -80 or larger should appear in the neighbour list.

The entire program can be viewed on GitHub:
`https://github.com/hansjny/GroupSimulations/blob/master/GenerateTopology.py`

4.3.2 Data output and visual representation

The result of the topology generation is stored in a JSON [18] data file. The data file contains the height and width of the the generated topology, as well as the number of nodes. A `node` 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.

The output from the program execution described in the previous subsection is a 23.2 MB large file containing the resulting topology-data in JSON.

By writing a simple HTML and JavaScript browser application, the JSON can easily be parsed and visually represent the nodes on a grid. The result after creating a topology and visualizing in the web application can be seen in figure 4.3.

4.4 WiGLE as a data source

In the previous section we looked at the data generation, representation and visualization of nodes in a network topology with uniform distribution. This section is dedicated to using an open data source called WiGLE to create the network topology, while reusing the code and data structure for representation and visualization.

```
{
  "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

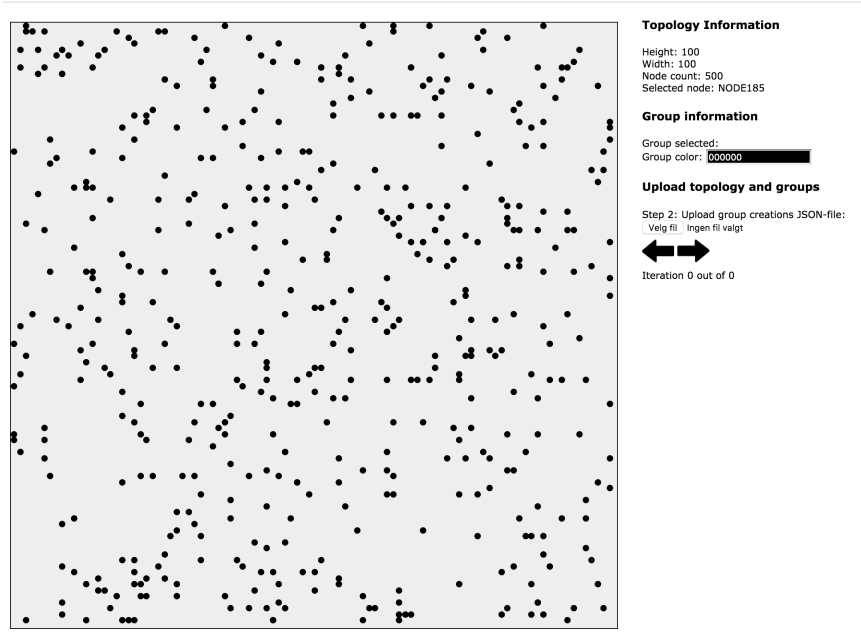


Figure 4.3: Generated topology with random, uniform distribution and the interface for viewing topologies

4.4.1 Introduction to WiGLE

The Wireless Geographical Logging Engine (WiGLE) [32] is a project started in 2001 which purpose is to gather information about wireless networks. All information collected by WiGLE is user submitted. Anyone can download an Android app developed and published by WiGLE, and use 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 as seen in figure 4.4. All the data can also be accessed through a public API. 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 which will be used to gather data for this thesis to a higher daily data limit.

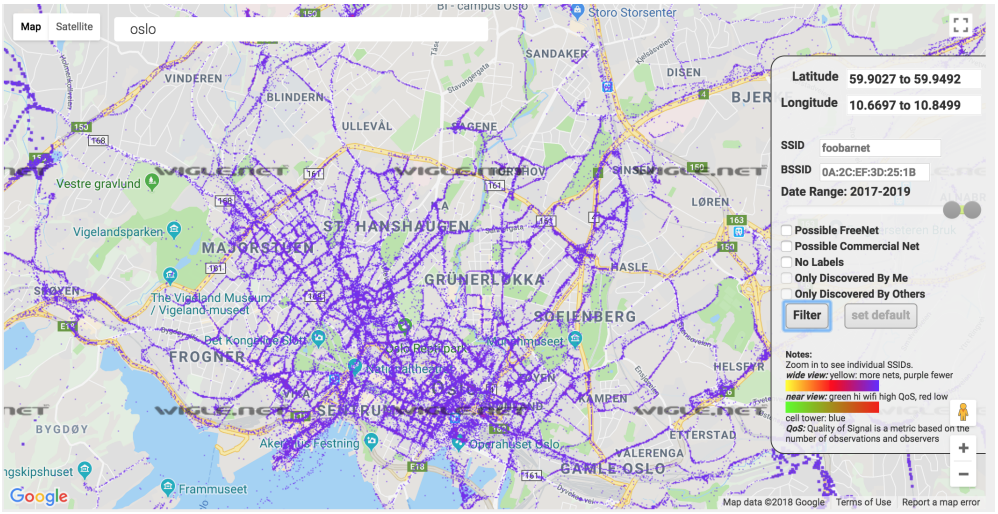
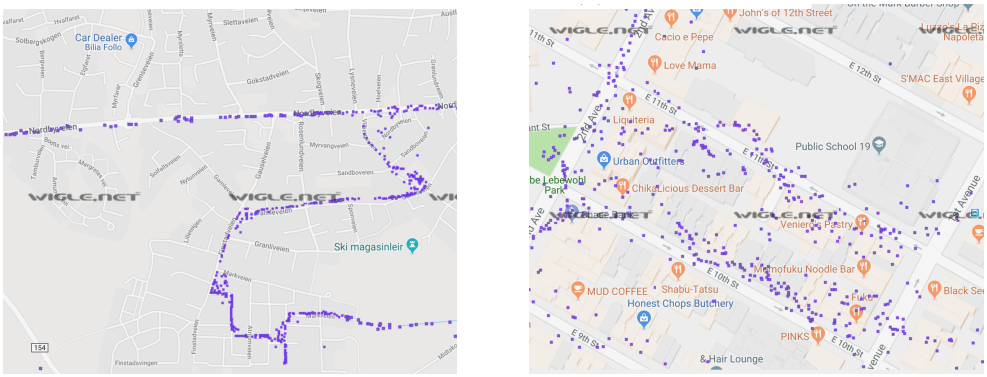


Figure 4.4: WiGLE map of Wi-Fi networks observed in Oslo from 2017-2018

4.4.2 Data quality

WiGLE determines the geographical coordinates of the position of each access point using weighted-centroid trilateration [27]. This means the AP location is not necessarily 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,

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



(a) Few observation points in Ski, Norway

(b) Several observations points in Manhattan, New York

Figure 4.5: WiGLE accuracy

```
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 4.6: Example of a Wigle API request

if the measurements are taken on only one side of the access point, the measurements are heavily shifted towards that side. This tendency can be observed when zooming in on many Norwegian towns where most access points seem to be positioned on the road. This is obviously not right, but as most observations are most likely are done from the road when wardriving, this happens frequently. Figure 4.5 illustrates the big difference multiple observation points can make.

Even though the data is not perfect, it is safe to say it is better suited to impersonate the layout of real-life network topologies than the generated data.

4.4.3 REST API

Overview

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 [31].

```
{
        "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 4.7: REST API response with AP data

Data gathering

To retrieve data for the topology generation progam, the only required part of the API is the network search service. There are several ways an access point can be retrieved using the network search. For instance specific SSIDs can be queried, a range of coordinates, APs with a minimum signal level, or only query networks that are free to use. As the purpose is to rebuild the network topology to fit the previously tailored data structure, all access points will be considered, and no filters applied except for a range of coordinates. By issuing an API request for nodes between two latitudes and two longitudes, the response is an array with AP objects within that area. An API request will look something like what can be seen in figure 4.6. The parameters $latrange1$ and $longrange1$ are the coordinates that marks the beginning of the area of interest, while $latrange2$ and $longrange2$ marks the end. To be meaningfully create a network topology based on the API request, information about *all* access points within the specified range is desired. Alas, as it happens WiGLE returns at most 100 results per query. This means if there are more than 100 access points, the *start* parameter has to be changed. This parameter tells WiGLE at which offset to begin fetching data from. For instance a start value of 0 instructs it to fetch the first 100 access points, with indexes 0-99. A value of 100 means that the next 100 APs in the range $100 - 199$ is fetched, and so on. The JSON response for a succesful request for one AP can be seen in figure 4.7.

The JSON-object in the response contains quite a lot of information about all of

the APs retrieved. Most of this data is redundant information, but it is worth noting that some of it could be used to perfect the search. For instance the "last updated"-parameter could be a way to refine the access points retrieval so access points which have been long gone is not fetched. The most valueable properties are *trilong* and *trilat*. As the names suggest these contain the estimated coordinates of the access points.

The Haversine Formula

As seen in the previous section, WiGLE provides data about the location of access points and a way to retrieve this data in a usable format. At this point a program that operates directly on the retrieved longitudes and latitudes could be made, but the problem is that the previous work relies on a two dimensional Cartesian coordinate system. To be able to reuse what we have have already built, the global coordinates has to be translated into distances.

The haversine formula [28] can be 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 [6], 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 4.8. 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 4.8: Implementation of haversine distance

This function is used to compute the distance in meters between the boundaries of the area of interest. When data is retrieved from WiGLE, the specified coordinate range is used to compute the size of the cartesian coordinate system, and it is done in the following way:

- Compute the distance between the first latitude φ_{start} and the second latitude φ_{stop} 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 the distance between λ_{start} and λ_{stop}.
- The same approach is 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.

4.4.4 Data output

The resulting python program[3] 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 4.3. The APs will be placed correctly relative to each other, with real world distance between them, based on the coordinates of each AP.

[3]GitHub URL: https://github.com/hansjny/GroupSimulations/blob/master/wigleData.py

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 signal strength measured between access points in two separated buildings should be lower than readings between access points in adjacent apartments. In the following chapter we will consider how we can utilize the signal strength measured between access points to build a clustering algorithm that creates reasonable groups of access points.

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.

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 most solutions presented in the related work section is usually only deployed under the same extended service set (ESS).

Just like a distributed version, a centralized controller is also restricted by the NP-complete nature of all graph colouring algorithms. To be able to compute a channel plan using heuristics, 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

We will continue considering a distributed approach as the main topic of the thesis. In reality this idea has some major challenges that we have to overcome. Here are the most prominent ones:

1. The group creation itself. Based on the signal strength 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.

2. 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.

3. State synchronization. As all nodes have to compute a channel plan for the group, they all need to have synchronized information about the members of the groups and possibly also every other node's signal strength measurements.

4. Channel plan computation. Even though the centralized approach would have a similar problem, it will be more difficult to solve in a distributed fashion as it is reasonable to assume that an access point has limited computational power. This point extends past the scope of the thesis, but algorithms like DSATUR (insert ref) and SCIFI are examples of algorithms that computes channel plans as a graph colouring problems.

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. 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 neighbouring

nodes. This neighbour list contains the appropriate computed signal strength measurements for each node. Based on the signal strength of their neighbours, the nodes should be able to create groups that resembles clusters. 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. To simplify the computation procedure and to reduce the workload for building the simulation program we are going to do a number of assumptions.

1. Assumption 1: All nodes involved in the simulation also run the group creation algorithm.

2. Assumption 2: When a node is observed over radio, it is also known 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 signal strength measurements and group membership. For now we will assume this is true, but in chapter (ref) we will address the issue.

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. This is also an issue that will be addressed in chapter 3.

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 [13], and is designed to parse the output from the data generation program and use it directly. 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. The program is not parallelized to keep results consistent and to easier debug and locate program errors.

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 signal strength measurements, 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

The results of the group computations 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. 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. This allows us to either implement a parser in python for the data and use `matplotlib` to visualize it, 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.

```
1  "iterations": {
2    "0": {
3      "0": {
4        "groupName": "GROUP0",
5        "members": {
6          "0": "NODE0"
7        },
8        "memberCount": 1
9    },
10     "1": {
11       "groupName": "GROUP1",
12       "members": {
13         "0": "NODE1"
14       },
15       "memberCount": 1
16     }
17 },
18   "1": {
19     "0": {
20         "groupName": "GROUP1",
21         "members": {
22           "0": "NODE0",
23           "1": "NODE1"
24       },
25       "memberCount": 2
26    }
27 }
28 }
```

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 Creating groups with clustering

In this section we will consider and evaluate a way to divide access points in groups by assessing an exisiting clustering algorithm, and consider different modifications to this algorithm to make it better suited for a distributed clustering of access points.

5.5.1 Agglomerative Clustering

There are a couple of reasons for choosing hierarchical agglomerative clustering as a starting point. First of all it does not need a pre-specified amount of clusters to be able to run, like for instance partitioning clustering needs. The second reason is that agglomerative clustering has a bottom-up approach, where the clustering begins by treating nodes as clusters and iteratively merges clusters to create a dendrogram that represents the current cluster and all previous merges. The bottom-up approach intuitively seems to fit well in a ditributed model, where all nodes know only of themselves and their neighbours.

Description

Agglomerative clustering [22] is a variant of hierarchical clustering where clusters are pairwise greedily merged. The starting number of clusters is the same as the number of points in the data set. Based on a specified distance metric, the two clusters that are closest to one another is merged into combined cluster. The clustering is complete when there is only one cluster left. It falls under the category of hierarchical clustering, and a dendrogram is iteratively being built for each merge, containing all past merges of the clusters. An example of agglomorative clustering can be seen in figure**??**.

Assessment

Agglomerative clustering provides us with the basic idea for a an algorithm we can use to create access point groups. However, there are two points where agglomerative clustering fails to meet our requirements:

- There is no upper limit to the amount of nodes in each cluster
- As the clustering algorithm will be running distributed on each cluster, a decision has to be made with only the local distance observations at hand. Assuming that there exists two groups that finds each other the closest (most disturbing), might prove a fallacy. Signal strengths are observed locally, and while one node observes another with a given signal strength, the mututal observation might be a different one. This could create a deadlock where there

exists no combination of group pairs where both observes the other as the closest group.

The first point has the trivial solution of checking the number of nodes before accepting a merge, and would only be a slight implemention change of agglomorative clustering. The second point forces us to change the paradigm of always finding two mutually closest groups.

5.5.2 K-Nearest Neighbour Clustering

Description

To resolve the problems discussed in the previous section, we will create a slighly different algorithm and refer to it as K-Nearest Neighbour Clustering (KNNC). In this method, similar to agglomerative clustering, in the beggining there are equally many clusters as there are nodes. Instead of looking for pairs that are mutually close, each cluster seeks to merge with the cluster that is closest. The distance is defined by the distance metric. The merge will always happen as long as the resulting cluster does not contain a higher node count than K.

Distance metric

The distance metric lets us specify exactly what the distance between two clusters signifies. As each cluster usually consists of several nodes, there are multiple options for what the distance between two clusters could be defined as. The distance metrics used in hierarchical clustering are typically either:

- The average distance between the nodes of each cluster
- The minimum distance between the nodes of each cluster
- The maximum distance between the nodes of each cluster

The metric we will use is the -dBi value of neighbouring nodes. Every node in a cluster tracks the observed -dBi value of all other nodes that are not a member of the cluster. The cluster containing the node with the highest -dBi value will be merged together with original one. This is similar to the minimum distance metric.

Implementation

Each node begins by identifying itself as a member of a group that only contains itself. Let us call this group a. Group a loops through the radio readings of every member of the group, and picks the node with the highest observed -dBi value to contact.

In the beginning there is only one node in group a. Hence 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. 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. A pseudocode sample of an implementation can be seen in 5.2. In a real world implementation there would have to be a supporting protocol to enable the flow of information and synchronization of data.

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 members a group can have, referred to as K. If the sum of members in two groups that wants to merge exceeds K, the merge is aborted and no changes is reported to have happened for either group. This means that the simulation algorithm converges when no groups remain that are small enough to merge with another.

5.5.3 Results on different topologies

Figure 5.3a shows the result of running the K-Nearest Neighbour Clustering algorithm on a uniform distribution topology, while figures 5.3b, 5.3c, 5.3d shows the resulting clusters after running it on topologies on citites and towns collected by WiGLE. The different node colors indicate different group memberships. The large scale images of each topology can be seen in appendix A.1.

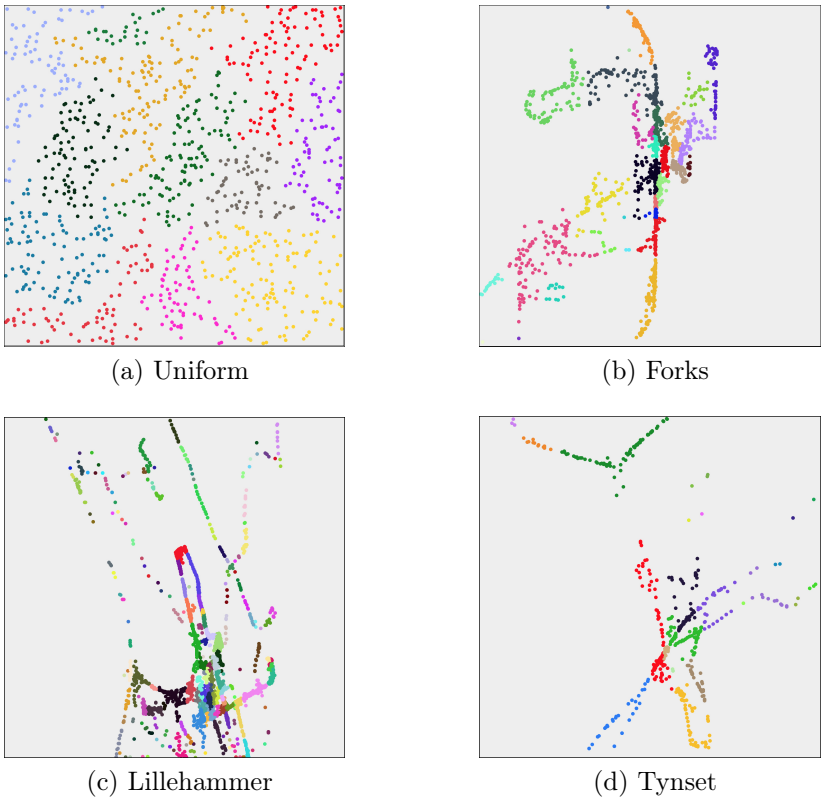
5.5.4 Evaluation

By looking at the results of the simulations of the K-Nearest Neighbour Clustering it is obvious that clusters are created. Even in the more chaotic topologies as Lillehammer, there are distinct clusters that encompasses the closest nodes. However, there are two obvious problems with this algorithm:

- In contrast to agglomerative clustering, our K-Nearest Neighbour Clustering algorithm does not care if the closest distance between the groups is pairwise mutual. This might lead to one group a merging into another group b, even while b has another neighbouring group c that lies closer. If the maximum size K is reached during the merge of a and b, the resulting group will never be able to merge with c even though that would have given a better cluster.

```
allGroups = [];
K = 120;

for node in topology: //Initialize groups
  g = new Group()
  g.members.append(node)
  allGroups.append(g);

while True: //Run as long as there are changes
  changes = 0;
  for group in allGroups:
    groupMaxDbi = -INFINITE
    groupClosestNode = None

    for node in group:
      nodeMaxDbi = -INFINITE
      nodeClosestNode = None

      for neighbour in node.neighbours:
        if neighbour.dbi > nodeMaxDbi:
          nodeMaxDbi = neighbour.dbi
          nodeClosestNode = neigbour

      if (nodeMaxdbi > groupMaxDbi):
        groupMaxDbi = groupMaxDbi
        groupClosestNOde = nodeClosestNode:

    var groupToMergeWith = groupClosestNode.group
    if (length(group.members) + length(groupToMergeWith.members)) < K:
      var newGroup = new Group()
      newgroup.members = group.members + groupToMergeWith.members
      allGroups.append(newgroup)
      allGroups.remove(groupToMergeWith)
      allGroups.remove(group)
      changes = 1
  if (changes == 0):
    break
```

Figure 5.2: Pseudocode sample of how the K-Nearest Neighbour Clustering runs in a simulated environment



(a) Uniform (b) Forks

(c) Lillehammer (d) Tynset

Figure 5.3: K-Nearest Neighbour Clustering on different topologies

- Once a cluster reaches the maximum size K, there is no possibilty for any other nodes to join the cluster. In our simulation of static topologies that might work, but in a real world scenario access points may turn on randomly, and be located in the middle of an already existing cluster. With the current algorithm there is no way to handle this event, and the node could not become a member of the cluster that surrounds it.

In the next section we will look at how we can use group splitting to create a solution for these problems.

5.6 K-means splitting

In this section the the concept of group splitting will be introduced. Group splitting is thought to improve the K-Nearest Neighbour Clustering algorithm suggested in the previous section. Moreover, this section is dedicated to the introduction, simulation and evaluation of one of the two split approaches that will be taken in the thesis.

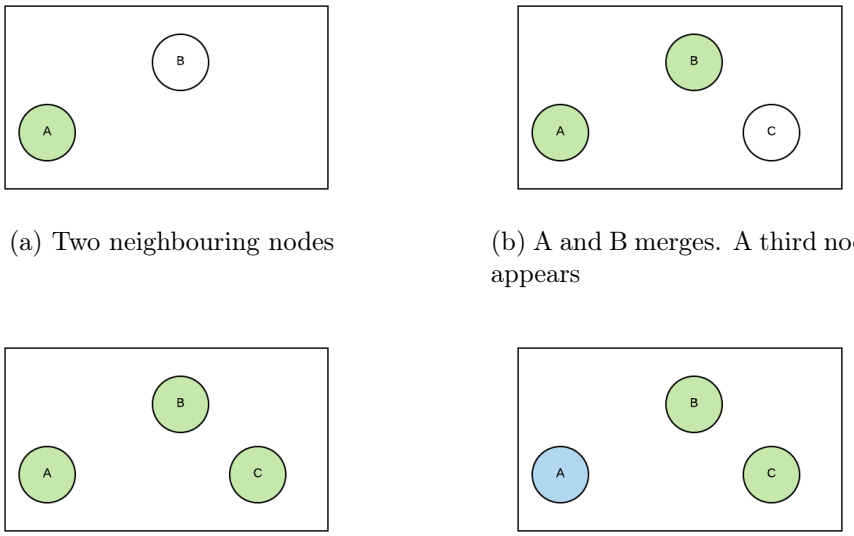
5.6.1 Group splitting

An implication created by letting one cluster merge with another without knowing if the merge is mutually beneficial, is that not all merges will be optimal. A way to increase the likelihood of a group ending up with the neighbouring nodes that impacts each other severely would be to accept merges that result in groups larger than K, the maximum size. The group would be too large, but more likely to have the opportunity to merge with the nodes of highest impact. To reduce the group size back to K, the least impactful nodes would have to be kicked out of the group. This is what we call group splitting. A simplified illustration of the basic concept is shown in figure 5.4, where the maximum group size is 2 and two neighbouring nodes join to reach the maximum group size. A third, more impactful node appears, and is included in a transient group before the least disturbing node is kicked out to get the group size back to K.

5.6.2 Introduction to K-means

K-means is a clustering algorithm commonly 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).

K centroids are randomly placed somewhere in the data set, then the distance between each data point and each centroid is computed. All data points closest to



(a) Two neighbouring nodes

(b) A and B merges. A third node appears

(c) All nodes joined in transient group

(d) Splitting to reduce group size

Figure 5.4: Simplified illustration of the idea behind splitting. K = 2

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. The algorithm continues until the position of all centroids remains unchanged after an iteration. When the position of the centroids are the same two iterations in a row, it means that the solution has converged and all K-clusters has been identified [20].

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

5.6.3 K-means splitting

K-means clustering is not directly applicable to solve the clustering challenge in this thesis. The simple reason being the distributed nature of the nodes and groups. Indeed, a vanilla implementation of K-means requires an extensive overview of the surronding, if not entire, network topology. The groups are so far 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. By extending the K-Nearest Neighbour Clustering 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 the K-means variable K, to $K = 2$, K-means can be used to identify where the transient group should be split to create two more connected clusters.

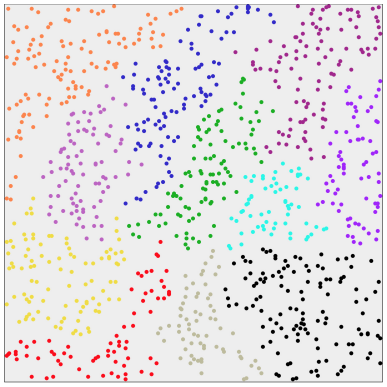
Randomly picking two nodes to be centroids could lead to undesirable results where the two resulting groups are fundamentally different from the original ones. Recall that the purpose of the split is to reevaluate the divison line between the groups and see if K-means can achieve a better split, not create two entirely new clusters. There already exists two independent clusters before the split. The centroids of these clusters can be precalculated before merging, and be used to bootstrap K-means with two non-randomly chosen centroids. 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. As the bootstrapped position of the centroids is planned and not random, the result is deterministic and the algorithm does not have to be run more than one time to get the result.

5.6.4 Results

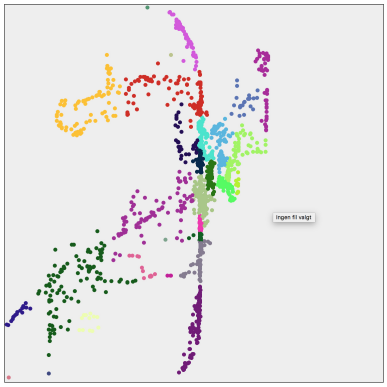
The results of simulating the group creation with K-means splitting implemented can be seen in figure 5.5 (see appendix A.2 for full size images). The groups clearly look very similar to those we computed in 5.3, but in some places there are some major differenes. This is where splits have happened. A comparison of the two algorithms executed on the same section of Tynset's town centre can be seen in figure 5.6, and another comparison of a section of the uniform distribution can be seen in figure 5.7.

5.6.5 Evaluation

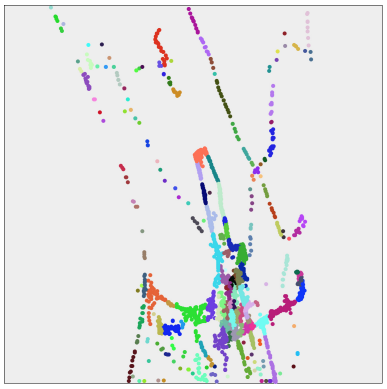
There is little doubt that K-means splitting improves upon the original algorithm, especially when splits only are accepted when the maximum interference of a group is reduced after the split. Even though the clusters in a static environment is slightly improved, the most important use-case for K-means splitting is under the introduction of new nodes to an environment where all groups have converged. Where in the unmodified K-Nearest Neighbour Clustering algorithm there was no specific way to handle a new node in an environment of saturated groups with K-means splitting eventually new groups will be formed on the basis of the updated topology. The weakness of this approach is how ill suited it is to run in a distributed environment. It is trivial to simulate splits using K-means, as during a simulation the absolute



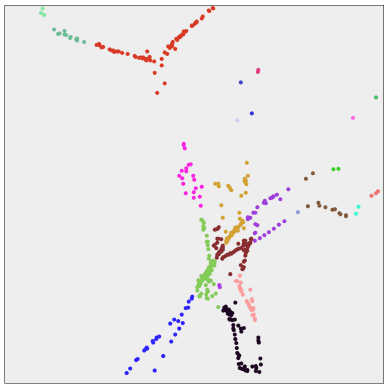
(a) Uniform



(b) Forks

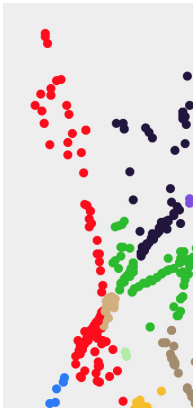


(c) Lillehammer

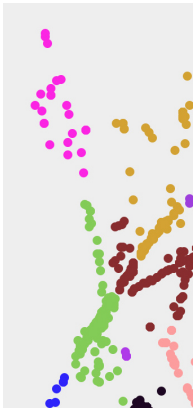


(d) Tynset

Figure 5.5: K-means splitting on different topologies

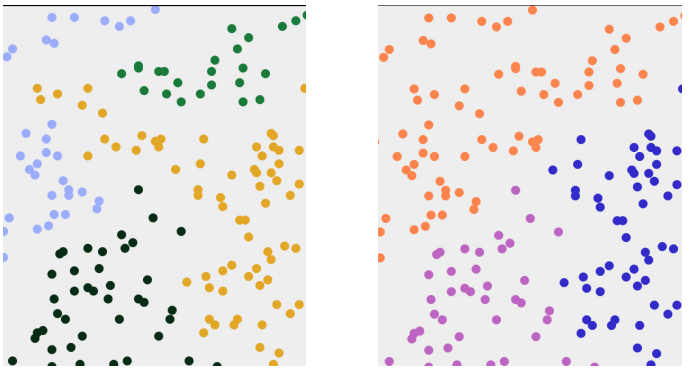


(a) Without K-means splitting



(b) With K-means splitting

Figure 5.6: Comparison with and without K-means splitting on Tynset



(a) Without K-means splitting

(b) With K-means splitting

Figure 5.7: Comparison with and without K-means splitting on uniform distribution

coordinates of the nodes in the group are accesible. This is information unavailable for nodes under real circumstances - the only available information is the approximiate point-to-point distance inferred from the signal strength scans. Hence some major adaptions would have to be made to get this to work in an implementation. One way to realize the algorithm would be to select a physical node as centroid instead of a virtual point. All nodes in sensing range of the surrounding area of the centroid node would report the distance to the centroid to their n-hop neighbour using an overlay network to communicate internally in the group. The other nodes could then figure out their approximate distance to the centroid. It is unknown if such an approach would be accurate enough to work, or have a complexity that is conceivable to implement.

5.7 Minimum Cut Splitting

In this section we will consider another approach to group splitting, namely the graph theory algorithm for finding a minimum cut in a directed graph.

5.7.1 Minimum cut

The core concept of the minimum cut algorithm is to find a way to cut a directed graph so that a source node is completely separated from a sink node, while cutting over edges whose weights adds up to a sum which is as low, or minimal, as possible [7].

The minimum cut is often referenced in the context of the max-flow min-cut theorem, which states that the maximum flow of a directed graph is equal to the

capacity of the minimum cut. The capacity of the minimum cut is equal to the sum of the weight of the edges the cut runs through. In 1956 Delbert R. Fuelkerson and Lester R. Ford proposed a method for finding the minimum-cut [12] in a flow network, now called the Ford-Fuelkerson method. The method was extended by the Edmonds-Karp algorithm [11] which is identical, except that is specifies that a breadth-first search should be used for the augmented-path stages in the algorithm, whereas in Ford-Fuelkerson this is unspecified (hence being called a method, and not an algorithm).

The implementation of minimum cut that will be used in this section is a part of the NetworkX [1] python package for manipulation of complex networks.

5.7.2 Using minimum cut for group splitting

Minimum cut can be used for group splitting by identifying the least connected partitions of a group. If a node, or a partition of the group, is connected to the rest of the group through one or more links which are weaker than the strongest link to a neighbouring group, the partition can be excluded from the group to allow for a merge with the neighbouring group.

To utilize minimum cut as a mechanism for splitting, the groups has to be treated as directed graphs. While the data structure of the networks in the implementation is not originally designed for graph operations, the nature of the nodes with its neighbour lists and signal strength metrics certainly can be treated as a graph. Meaning that each node's neighbour list contains its outgoing edges, while the belonging signal strength value is the weight of the edge. In contrast to the K-means splitting implementation, the minimum cut implementation does not require a larger transient group. Instead the groups have to negotiate to know when a split can be confirmed. A flowchart illustration an abstract flow of the simulation implementation can be seen in figure 5.8, showing the steps which are more closely described below.

1. Two groups A and B wants to merge, their combined size is larger than the specified group $maxSize$. The measured maximum value of signal strength between nodes in the two different groups A and B is called $rssiThresh$.

2. The groups independently build a graph of their current state, using the signal strength values as weights on edges to neighbouring nodes. In the simulation implementation this is done using the NetworkX graph.

3. Each edge in the graph where the weight is larger than $rssiThresh$, gets a new weight = `INFINITE`.

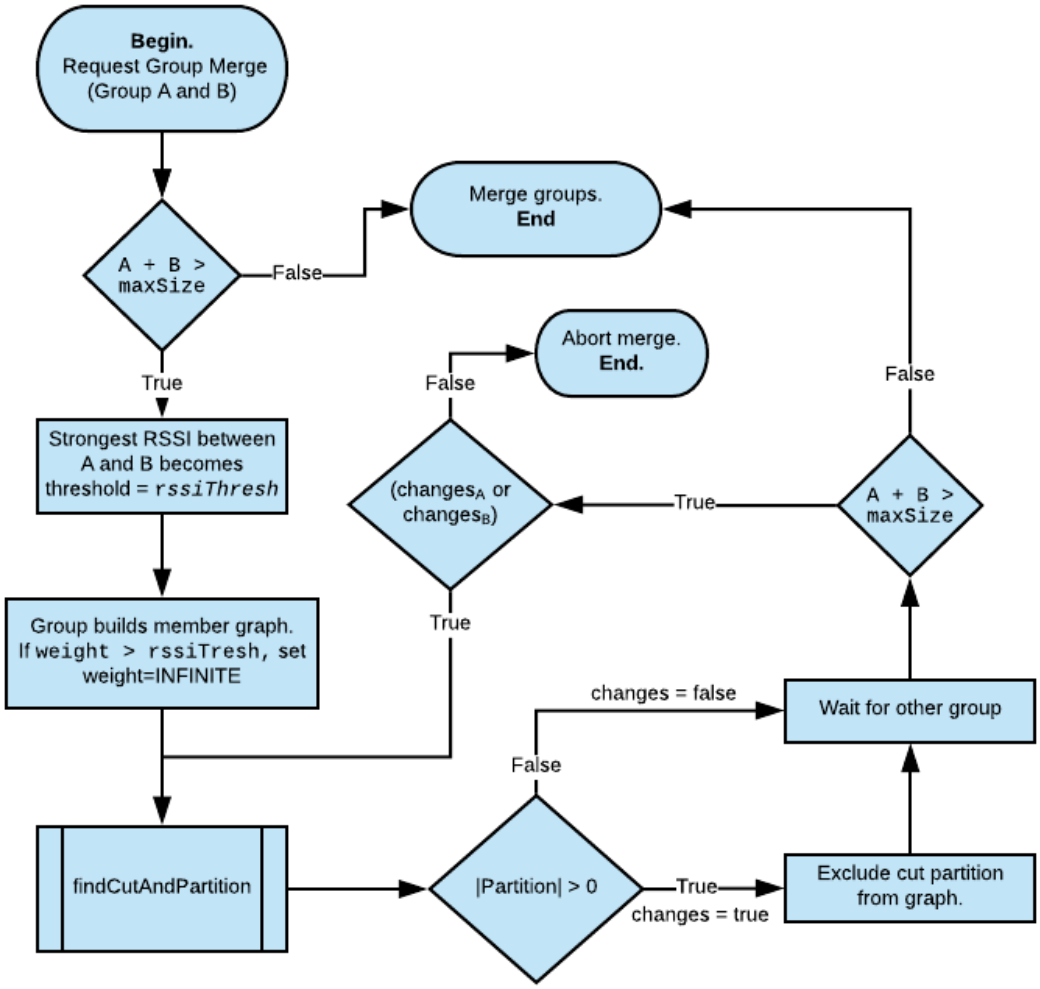


Figure 5.8: Flowchart of the minimum cut implementation for splitting

```
def findCutAndPartition(G, sink):
minCutCapacity = 99999 #Initiated to infinite (or a very high number)
minCutPartition = [] #Partition initially empty
for source in G.nodes_iter():
  if source == sink: #Source and sink should not be the same
      continue
  cut, partition = nx.minimum_cut(G, source, sink)
  if (cut >= 99999): #If cut is too high (infinite), dont keep
      continue
  if cut < minCutCapacity:
      minCutCapacity = cut;
      if sink in partition[0]:
          minCutPartition = partition[1]
      elif sink in partition[1]:
          minCutParttition = partition[0]

for n in minCutPartition: #Remove nodes from graph
      G.remove_node(n)
return list(minCutPartition);
```

Figure 5.9: Pseudocode of the minimum cut stage of the splitting

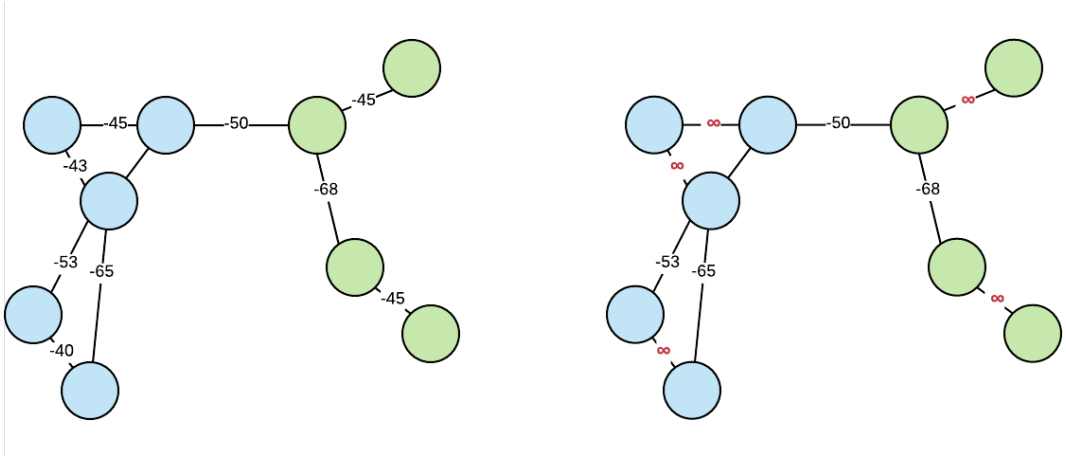
4. The groups independently perform minimum cut multiple times. In the simulation implementation this is done with a call to the library function `minimum_cut(G, s, t)`, where G is the graph, s source and t is the sink. To find the best partition to exclude from the graph, the sink node is always the node which lies closest to the other group, while the source node changes for each call to minimum cut. When all nodes, except for the sink, have been sources in the minimum cut algorithm, this step is finished. The cut which has the lowest capacity of all performed cuts is stored along with the partition of this cut.

5. The partition which does not contain the sink node is excluded from the graph, however if this partition is empty it means there exists no suitable cuts in the group which are less beneficial than the addition of another group.

6. The groups A and B check if the combined number of nodes exceeds $maxSize$. If the node number is still too high, repeat step 4 provided that there were any exclusions in the graphs in the previous run. If there are no exclusions in either group, there exists no cut that can bring the combined group size down to desired size, and the merge has to be aborted.

The pseudocode for step 4-5, referred to as the `findCutAndPartition` procedure in the flowchart of 5.8, can be seen in 5.9.

For a detailed step by step illustration describing how a split happens in a topology where a split would be beneficial, see figure 5.10.



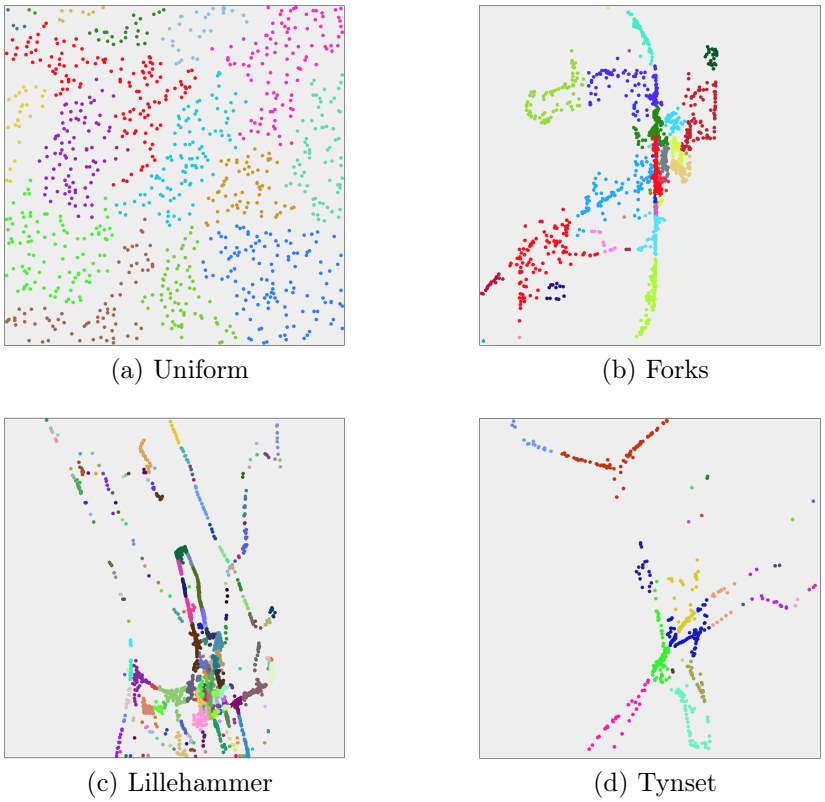
(a) Two groups are initially too large to merge. A split is going to be attempted.

(b) Both groups identify edges which should not be cut, marked by an infinite edge weight

(c) The groups excludes the partitions with the lowest capacity cuts

(d) The number of members of the groups combined is now lower than maximum size, and the merge is succesful

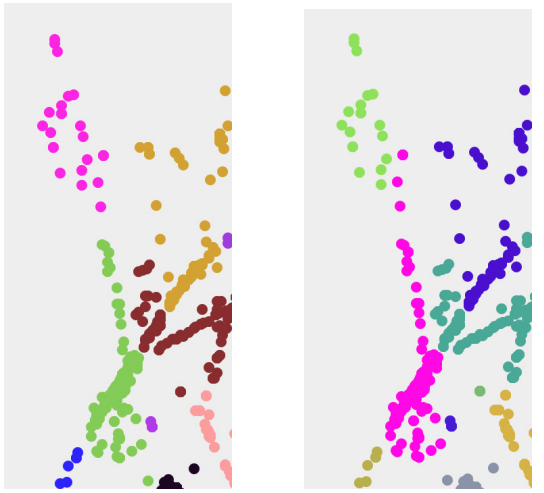
Figure 5.10: Minimum cut illustration with group maximum size set to 5



(a) Uniform (b) Forks

(c) Lillehammer (d) Tynset

Figure 5.11: Minimum cut algorithm splitting on different topologies



(a) With K-means splitting

(b) With minimum cut splitting

Figure 5.12: Comparison between K-means splitting and minimum cut splitting on Tynset

5.7.3 Simulation results

The results of the simulation, using the same four topologies used for previous simulations, with the implementation of minimum cut splitting can be seen in figure 5.11 (full scale images can been seen in A.3). Different groups are distinguished by color, and the group max size is set to 128.

Comparison

In the result section of K-means splitting there was a close up examination and comparison of the group division in Tynset's city centre using no splitting and K-means splitting. In figure 5.12 a close up of the same area has been made between K-means splitting and minimum cut splitting. The group division seems to be very similar, except for a major difference in a cluster on the top left area of the topology (pink color on 5.12a). While the K-means splitting algorithm identifies a neatly separated cluster, the same cluster in the minimum cut method does not include three nodes at the bottom right edge of the cluster. This is imperfect behaviour. By looking at the minimum cut, figure 5.12b, one can tell that the distance from the three pink nodes to the pink cluster is large. Actually, it is larger than the distance to the green cluster. The green cluster is far from being the max size of 128, so upon a merge, shouldn't a split happen which would redefine the groups in a better way? What could possibly be the cause of this less optimal behaviour?

Algorithm inspection

The application created earlier to step through each iteration of the algorithm can help illuminate the problem. The entire solution converges in 11 iterations, but iteration 8 and 9 seems to be the iterations which spefically impacts the result of the cluster in question. These are illustrated in figure 5.13. In 5.13a, the green group seeks to merge with the pink. They lie very close with a signal strength between them being -45 dBi. During the split several nodes are kicked out, which can be seen in 5.13b. As expected, the three nodes in question have not been kicked out. By watching the figure it is easier to understand why. These 3 nodes are obviously less tighter connected than -45dbi, which would the threshold for minimum cuts over this graph, but they are not kicked out because sometime during the split the size of the group becomes lower than the max size of 128 - before they have the chance to be thrown out. The merge is then immediately accepted.

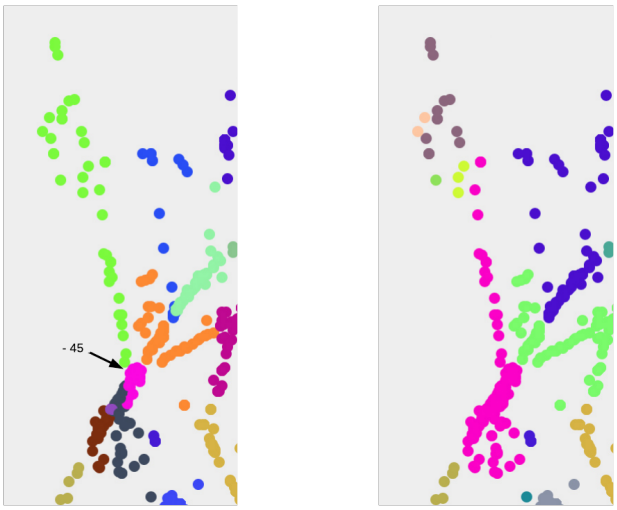
An improvement to the algorithm would be to change step 6, so that the minimum cut partitioning would run until no more nodes could be kicked out. Even if the group size reached a tolerable size before that. A change like that would prevent leaving nodes in a group when their neighbours have been kicked out, and they would also be better off in another.

Results after re-evaluation

The algoritm results after the modification is certainly more satisfactory than before. The final comparison between K-means and the new minimum cut can be seen in figure 5.13. For the full size version of all four topologies on this simulation as well, see appendix A.4.

5.7.4 Evaluation

While being slightly more complex to implement in a simulation than the K-means splitting, the results are more reliable and less dependent on initial group selection. In all simulations the groups have been merged iteratively, meaning that the merge sequence depends on the order of merges. With this minimum cut implementation the resulting groups will become the same, no matter the order of group merging. This is a property which can be considered of importance in a real world scenario. The major benefit the minimum cut group splitting has over K-means splitting in a real-world implementation scenario is the need for only link weights. As mentioned, K-means would be hard to realize, while the minimum cut method should in theory be little different from simulation to real world implementation.



(a) Iteration 8: The green group seeks to merge with the pink, the signal strength between them for instance being -45dBi

(b) Iteration 9: After the minimum cut split, some nodes have been thrown out to allow for the merge

Figure 5.13: Illustrating two algorithm iterations to find the problem source of the minimum cut

5.8 Evaluation of the clustering study

In this chapter four methods of clustering has been considered: regular agglomerative clustering, K-Nearest Neighbour Clustering (KNN Clustering), KNN Clustering with K-means splitting and KNN Clustering with minimum cut splitting. Which of the first is a well known clustering algorithm, and the last three has been proposed in this thesis. This section is dedicated to the evaluation of the work done throughout this chapter, with regards to gained knowledge and simulation weaknesses.

5.8.1 Knowledge acquired from simulations

Before beginning the work on the thesis it was hard to envision how groups could be formed in a distributed manner, without any central controller having a top down view of the landscape of wireless access points. While many questions still remain, some of which will be addressed in the next chapter, many has also been answered. The main issue addressed is how groups can be formed, and how they can be continously updated to keep up with changes in the surrounding network.

Unsurprisingly, the small modification to the agglomerative clustering algoritm, which made KNN Clustering, yielded promising results. This algorithm could provide a natural starting point for a real-world implementation in a static network topology. However, this specific algorithm does not handle changes in the network environment very well, as once a group's maximum size is reached, it can not be modified. Still it could possibly provide useful for testing scenarios, maybe in an early implementation when a distributed channel allocation algorithm is to be tested across the group, without wanting changes to happen. By considering ways to improve this algorithm, the notion of splitting was introduced, and tested through the methods of K-means and minimum cut.

K-means was an interesting clustering algorithm to work with. The promising splitting results observed on the simulation topologies was especially fun to see, as the K-means method was adapted to fit the clustering purpose, and not implemented as a textbook K-means clustering. However, when comparing the K-means and the minimum cut algorithm to decide which one is the best bet for a real-world implementation of a splitting algorithm, the minimum cut algorithm comes off as the easiest to implement given the limited knowledge of the surrounding nodes each access point has access to. Minimum cut also has the benefit of being more predictable in its splitting. The layout of a network is so similar to a graph, that in hindsight a version of the minimum cut algorithm seems like such an obvious choice.

5.8.2 Simulation weaknesses and data bias

As seen the simulations have taught us about what kind of clustering could work under the assumptions made early in the thesis. The obtained results are also a product of the data that was gathered, and the simulation program which was made. This subsection will consider all the different aspects that has not been considered when producing the simulated results, but might have an impact in a real life implementation.

Volatility

All the simulations 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. Actually, it is more like a computation than a simulation in its rightful meaning. While it is true that most access points are relatively stationary and constant, it is 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. Any reader of this thesis will know that ways the algorithm can handle volatily has been considered, enabled by group splitting, but the simulation framework does not support inserting or removing nodes during the computation, hence only static environments that quickly converges have been tested.

Signal strength reporting bias

The group computations has been based on the signal strength of all nearby access points, and the dBi values that denotes all signal strengths has been calculated using the free space path loss formula. As distance is the only variable that affects the result of the signal strength calculation, it means that the signal strength levels becomes a symmetric binary relation between nodes. In the real world this is not the case, and also the reason allgomerative clustering was deemed less ideal. Access points may interfere with each other differently, and the signal strength values may change when there are variations in the environment. This is the reasons deadlocks could occur if implementing regular agglomerative hierarchical clustering and the motivation behind creating K-Nearest Neighbour Clustering, even though the topologies actually could work well with agglomerative clustering. We also have no simulations of how groups would look if the perceived signal strength between two nodes were not mutual.

Dimensionality and node location reporting bias

Both the data that has been randomly generated, and the data fetched from `wigle.net` takes the x and y axes in account. Obtaining data in three dimensions is harder. Wigle places the access points using longitude and latitude as all their data is acquired using ground-level triangulation By creating clusters based on information in two dimensions there is no way to know how the algorithm behaves in a 3-dimensional world. Of course, it is reasonable to think that an algorithm which is based on distance two dimensions should also work well in three dimensions, but there has been done no simulations in three dimensions. As mentioned in the data-mining chapter, Wigle has a tendency to place nodes closer to roads, which means that the positioning of nodes may not be strictly accurate in two dimensions either.

Chapter 6

Communication protocol, state synchronization and future work

This chapter is dedicated to the specification, design and implementation criteria of what will be referred to as the Distributed Group Creation Protocol (DGCP). While in the previous chapters it has been suggested an algorithm for how groups can be formed, in this chapter the focus shall be on the technology and protocols required to enable communication between access points, - and ultimately groups. There has already been done some work on the subject of access point communication over IP, and previous work in the field of distributed consensus can be used to ensure synchronized group states. In this chapter these technologies will be covered and considered, and finally a protocol sketch that stitches all required components together will be suggested.

6.1 Problem overview

In chapter 5 we looked at an algorithm that enables cluster formation in a distributed environment where no node knows the complete layout of the surrounding networks. A number of assumptions were made before we suggested an algorithm. Two of the assumptions encapsulated the three following problems:

- Direct contact between access points is possible
- An underlying group communication protocol is in place
- The state of a group is synchronized throughout all its members

In this chapter we look at technologies that can handle these issues to suggest an abstract architecture for a protocol that enables group creation in a distributed environment.

6.2 Enabling technologies

When designing a protocol that enables group communication, there is need for some some already well-researched theory and technology as a foundation. There are already enough considerations to take when designing an architecture like this, and there is no need to reinvent the wheel where there is already done a good amount of research. Raft will be used for state synchronzation. Raft is a distributed consensus protocol and provides the equivalent degree of fault-tolerance as the Paxos [19] protocol family. ResFi is a protocol used to enable communication over IP between adjacent access points. This section is dedicated to briefly introduce these protocols.

6.2.1 Distributed consensus with Raft

Raft [26] is a distributed consensus protocol designed with simplicity in mind. The creators of Raft designed it to be used for educational purposes. Traditionally, Paxos has been used to explain distributed consensus, but Paxos has a complex design with extremely many different variations. Raft is now taught in courses all over the U.S, and their GitHub page includes to links of implementations in a large amount of languages. As Raft has such a wide range of implementations and is an easy protocol to understand relative to its quite complicated relative, Paxos, we will suggest that Raft will be used to handle distributed consensus.

6.2.2 Why distributed consensus?

Distributed consesus is required for the distributed group creation protocol for a couple of reasons, the most important two being:

- Leader election for decision making
- Synchronous data replication (makes sure all nodes have a consistent picture of the group, in-case leader goes down)

These reason are derived from the simple fact that there is no centralized controller to take decisions, so equally replicated data across all members node is required for all access points to reach the same decisions about group membership.

6.2.3 What Raft can not help with

Raft is not originally intended to run in a flexible environment where the amount of servers changes rapidly. Thus, Raft is not able to handle the group membership changes after two groups merge, within the same Raft session. This is because there

can be no consensus on past logs between nodes from different groups, as all groups will have different logs from before the merge. There is also no native Raft method to invoke native leader handover, so in the event of merge there would be two leaders.

Another aspect that would be different from a traditional distributed consensus scenario, is the assumption that there are two main actors with different roles: servers and clients. In distributed consensus, each server is supposed to have their own data, identical via data replication across across all the servers in the Raft session. The clients are the actor that requests changes on the data.

A traditional scenario is the banking example: a client could be a mini-bank issuing a withdrawal from a bank account. The servers would be all the banking servers making sure the new, updated account balance is consistent no matter where money is withdrawn from.

In the distributed group creation protocol all nodes running Raft are both servers and clients. They have to report new changes in the form of neighbours and signal strength values, while also keeping a local, replicated copy of the state of the group.

6.2.4 Access point communication with ResFi

This subsection is dedicated to briefly describe how ResFi operates, and to cover why and how it is a protocol that can be taken advantage of in the Distributed Group Creation Protocol.

ResFi is a protocol framework that supports creation of radio resource management in legacy residential networks. ResFi is intented to be used in a chaotically deployed landscape of acccess points, which is similar to the intentions of the group creation in this thesis. It enables the creation of secure point-to-point communication channel over IP through wired backhaul network. Point-to-point in this context means APs that are directly adjacant (or neighbouring APs as it has been referred to in the group creation chapter). It also supports secure broadcast via n-hop communication. A brief account of the sequential steps of the ResFi standard mode of operation follows, the more thorough explanation can be find in the ResFi paper [33] under chapter *IV. Detailed Specification.*

1. When an AP (a) is booted, a symmetric group key is created, along with an RSA key-pair.

2. a scans all 802.11 channels for neighbouring APs. For each AP it finds, a sends out a probe request that contains its public IP and public RSA key. It also includes the symmetric group key. As a response to the probe request it receives a probe response containing the equivalent information for each neighbour.

3. When the exhchange has happened, *a* subscribes to the publish sockets of all the neighbouring nodes using the IP received in step 2. Each neighbour in turn subscribes to *a*'s publish sockets as well. This makes it possible for each AP to broadcast messages to all subscribed neighbours, or create a unicast session key between one specific AP to enable secure and bidirectional unicast communication.

ResFi has a north-bound framework API that lets application running on the AP use ResFi's features through an API, without doing direct modifications to the implementation. All communication happens in the JSON-format. Table 6.1 shows which functions are available in the north-bound API, original table also including the south-bound API functions can be found in [33] table 1.

sendToNeighbor(ap_id, message)	Sends a message to an ap with id ap_id. Message is in JSON format. The message is encrypted using the symmetric unciast session key.
sendToNeighbors(msg, TTL)	Sends a message to all neighbours. Will be flooded out to n-hop neighbours, where n = TTL.
getNeighbour()	List all neighbour AP IDs
regCallbacks(newMessage, newNode, nodeDC)	Registers callback functions for the events new message, new neighbour node, and node disconnected.
registerNewApplication(name)	Registers a new application with ResFi. Names are used to separate different applications.
getResFiCredentials(param)	If param is 1, it return the public IP of the AP, and if param is 2 it returns the public RSA key
usePrivateRSAKey(data, mode)	Uses RSA key on tjhe data. If the mode is 1, it computes the signature of the data, if mode is 2 it assumed the data is encrypted and decrypts it with the key.

Table 6.1: ResFi north-bound API

6.3 Protocol design

This section outlines the architecture of a protocol that could facilitate group creation.

6.3.1 Archictectural overview

The protocol architecture we suggest here relies on 4 main components which can be seen in figure 6.1. In the figure the blue boxes signifies logic that has yet to be implemented, while grey boxes signifies the components that needs to pre-exist. The arrows represents which of the components that has to communicate with each other. In the next few subsections we address the two blue compontents individually.

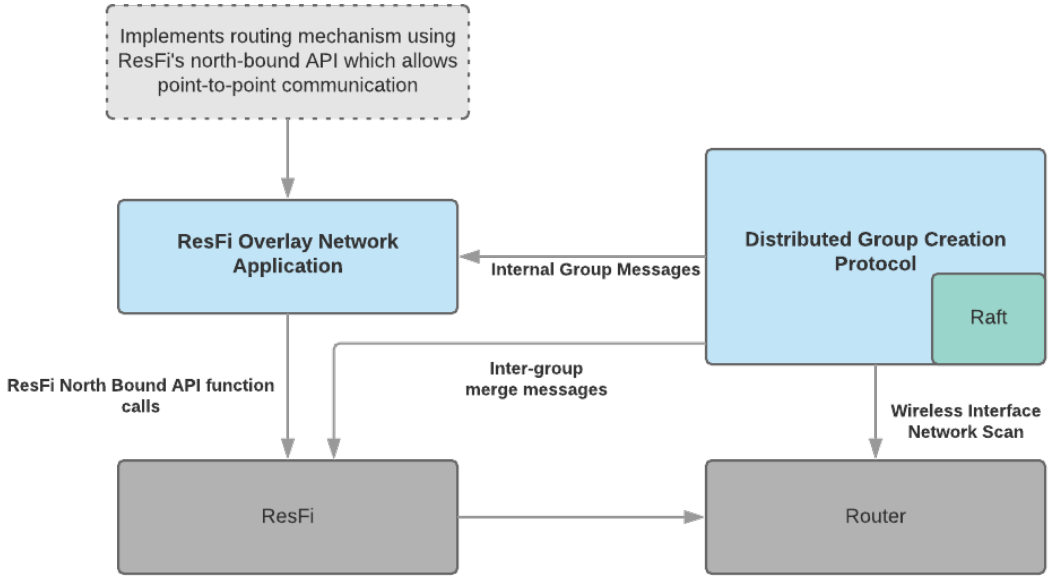


Figure 6.1: Architectural overview of protocol components

6.3.2 ResFi Overlay Network Application

As made clear earlier, ResFi enables secure two-way communication between 1-hop neighbours, and broadcast messaging via n-hop neighbours. To enable secure two-way unicast communication throughout the group, an overlay network application can be built using the north-bound ResFi API. This means the overlay network application would have to use ResFi for point-to-point communication, and then implement its own routing mechanism to relay messages from node to node, until the message reaches its destination. Let us look at the following suggested criteria for the ResFi Overlay Network Application:

- Messages sent through the ResFi OverLay Network Application can only reach members of the group. If a message is requested for a node not inside the group, the message is not relayed.
- Should be used for all communication between nodes, like control messages, Raft log updates, etc., except for merge messages, which would have to happen across groups.
- Uses IPv6 address to uniquely identify nodes ???

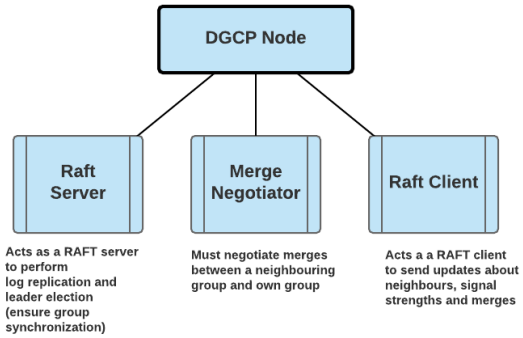


Figure 6.2: The roles of a DGCP Node

6.3.3 Distributed Group Creation Protocol

This subsection is dedication to a suggested architectural overview of the Distributed Group Creation Protocol (DGCP). The protocol relies on the ability to directly interface with the access point radio, to execute and parse the results of the network scan. The protocol also needs to be able to send message through the ResFi Overlay Network Applications to the rest of the group, as well as direct access to the ResFi north-bound API to be able to contact 1-hop neighbours that are not in the group, to negotiate merges.

The roles of a node running the protocol is illustrated in figure 6.2. What follows is an account on the different services and functionalities the protocol has to implement.

- Decide which nodes to merge with.
- Negotiate merges with neighoburs not in the same group
- Incorporate Raft to ensure all nodes in the group know the entire group network topology

Deciding which nodes to merge with

Since the beginning of the thesis, the idea has been to let all nodes have the exact same information about the group state to calculate the same results. Deciding which neighbouring groups to merge the group with is done by checking which nodes in the group has the most impactful neighbours. As all nodes in the group should share the same information, all nodes will find the same result of which neighbour to merge

with. In the event that nodes in the group observe neighbours with exact equal signal strength, there will need to be implemented a deterministic way to decide which of the nodes would be chosen. This could be as simple choosing the neighbours with the highest alphbetical order of the SSID. The important matter is that all nodes reach the same result of which group to merge with.

Negotiating merges via message passing

We established that the protocol has to make sure all nodes in a group conclude which neighbour disturbs the most on their own. For the sake of simplicity, we call this most disturbing neighbour X. X is a member of group G. So now the nodes in the starting group knows it will attempt a merge with group G. The node that lies physically closest to X should be responsible for negotiating the merge. We can be sure this node will be able to observe X over its radio, and can then use direct ResFi uni-cast communication with this node.

For the handling of merges themselves, there are probably several ways this can be executed. In this thesis we suggest the following way: Node N contacts is a part of a group G that has identified node X as the most disturbing node. Node N will then contact the node issuing a `merge request` message directly over ResFi. The merge request contains the entire network topology of G. X will then locally merge the nodes to see if it is valid according to the maximum size. If the merge is accepted, it replies with a `merge accept`.

The Distributed Group Creation Protcol uses the ResFi Overlay Network Application to communicate with the rest of the group. Group communication messages should include (but may not be limited to):

- Neighbour updates, e.g. a new access point appeared on the network scan of one of the routers, or significant signal strength value changes
- Raft log updates, heartbeats, vote requests, and vote messages
- Merge attempts

Preventing duplicate merge attempts

To prevent a group from repeatedely attempting to merge with another group we suggest that a hashing the member datatru

6.4 Future Work

- Assessment of the ResFi overlay network application, the protocol, modification and implementation
- Testing group creation using the clustering algorithms
- Security evaluation

Bibliography

[1] P. Swart A. Hagberg D. Schult. *NetworkX*. Online: accessed 21-March-2018. 2015. URL: https://networkx.github.io/documentation/networkx-1.10/.

[2] 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.

[3] 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.

[4] 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.

[5] Daniel Brélaz. 'New Methods to Color the Vertices of a Graph'. In: *Commun. ACM* 22.4 (Apr. 1979), pp. 251–256. ISSN: 0001-0782. DOI: 10.1145/359094.359101. URL: http://doi.acm.org/10.1145/359094.359101.

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

[7] John W Chinneck. *Practical optimization: a gentle introduction*. 2006. Chap. 9. URL: http://www.sce.carleton.ca/faculty/chinneck/po/Chapter9.pdf.

[8] 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.

[9] 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.

[10] Terry L. Cole and Simon. eds. (2007) Barber. 'Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (IEEE Std 802.11-2007)'. In: *Local and Metropolitan Area Networks, Specific Requirements, IEEE Standard for Information technology— Telecommunications and information exchange between systems.* (2007).

[11] Jack Edmonds and Richard M. Karp. 'Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems'. In: *J. ACM* 19.2 (Apr. 1972), pp. 248–264. ISSN: 0004-5411. DOI: 10.1145/321694.321699. URL: http://doi.acm.org/10.1145/321694.321699.

[12] Lester R Ford and Delbert R Fulkerson. 'Maximal flow through a network'. In: *Canadian journal of Mathematics* 8.3 (1956), pp. 399–404.

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

[14] 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.

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

[16] 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.

[17] '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.

[18] *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.

[19] Leslie Lamport et al. 'Paxos made simple'. In: *ACM Sigact News* 32.4 (2001), pp. 18–25.

[20] David J. C. MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, New York, 2003. Chap. 20, pp. 284–292. ISBN: 0-521-64298-1.

[21] Petri Mahonen, Janne Riihijarvi and Marina Petrova. 'Automatic channel allocation for small wireless local area networks using graph colouring algorithm approach'. In: *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*. Vol. 1. IEEE. 2004, pp. 536–539.

[22] D. Müllner. 'Modern hierarchical, agglomerative clustering algorithms'. In: *ArXiv e-prints* (Sept. 2011). arXiv: 1109.2378 [stat.ML].

[23] 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.

[24] 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.

[25] 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.

[26] Diego Ongaro and John Ousterhout. 'In Search of an Understandable Consensus Algorithm'. In: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX ATC'14. Philadelphia, PA: USENIX Association, 2014, pp. 305–320. ISBN: 978-1-931971-10-2. URL: http://dl.acm.org/citation.cfm?id=2643634.2643666.

[27] Navin Kumar Sharma. 'A Weighted Center of Mass Based Trilateration Approach for Locating Wireless Devices in Indoor Environment'. In: *Proceedings of the 4th ACM International Workshop on Mobility Management and Wireless Access*. MobiWac '06. Terromolinos, Spain: ACM, 2006, pp. 112–115. ISBN: 1-59593-488-X. DOI: 10.1145/1164783.1164804. URL: http://doi.acm.org/10.1145/1164783.1164804.

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

[29] 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.

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

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

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

[33] 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

Simulated Group Topologies

A.1 K-Nearest Neighbour Clustering

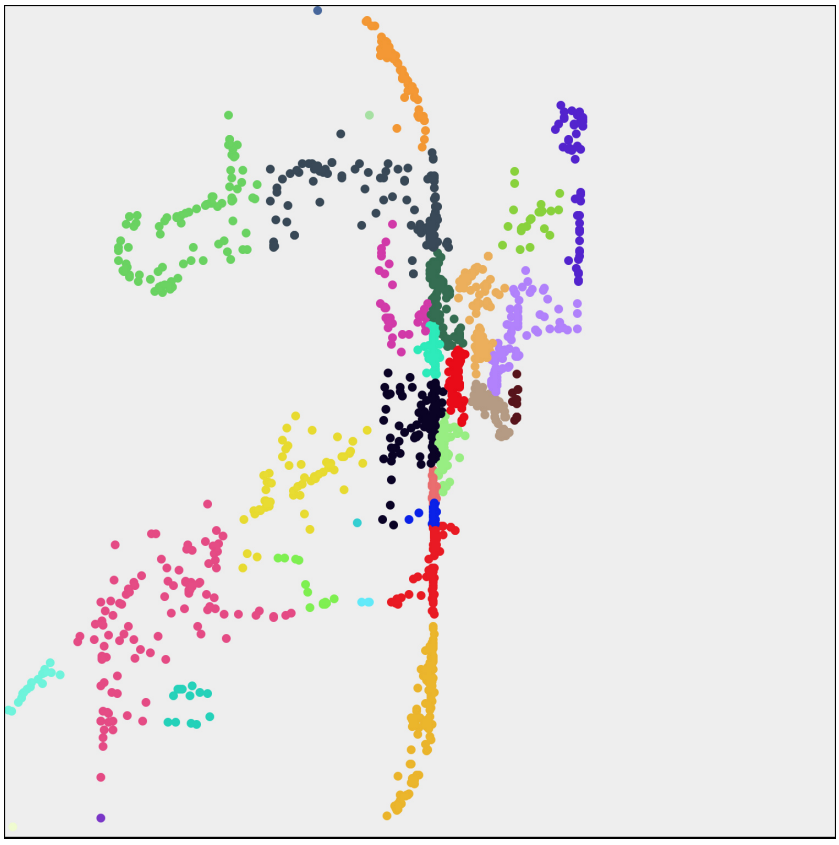


Figure A.2: Forks, Washington, USA

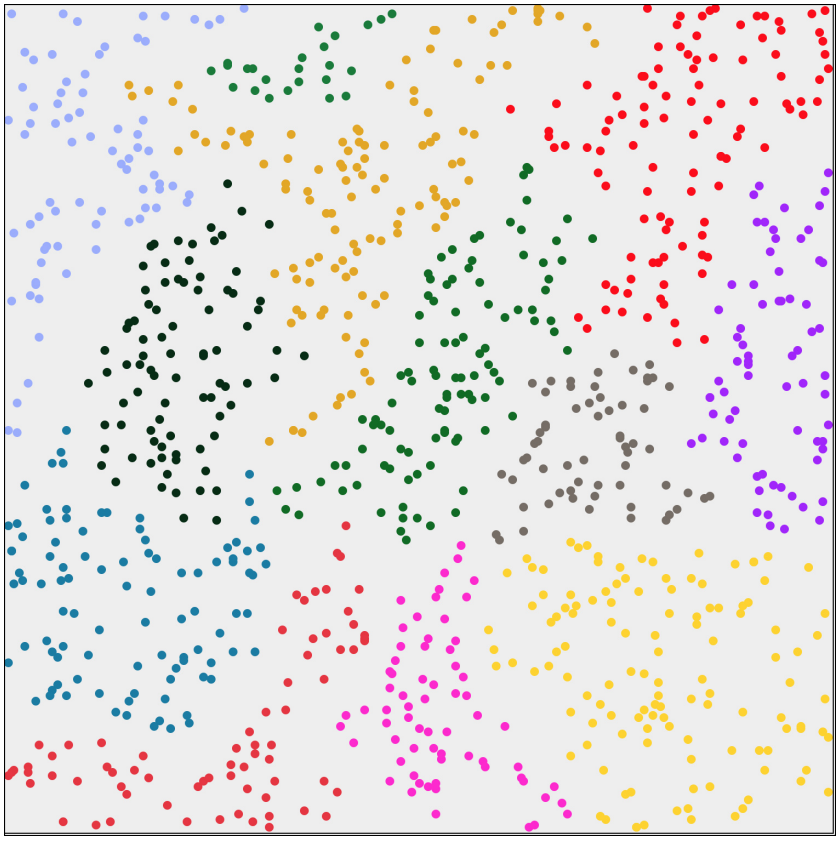


Figure A.1: Uniform distribution, 500x500, 1000 nodes

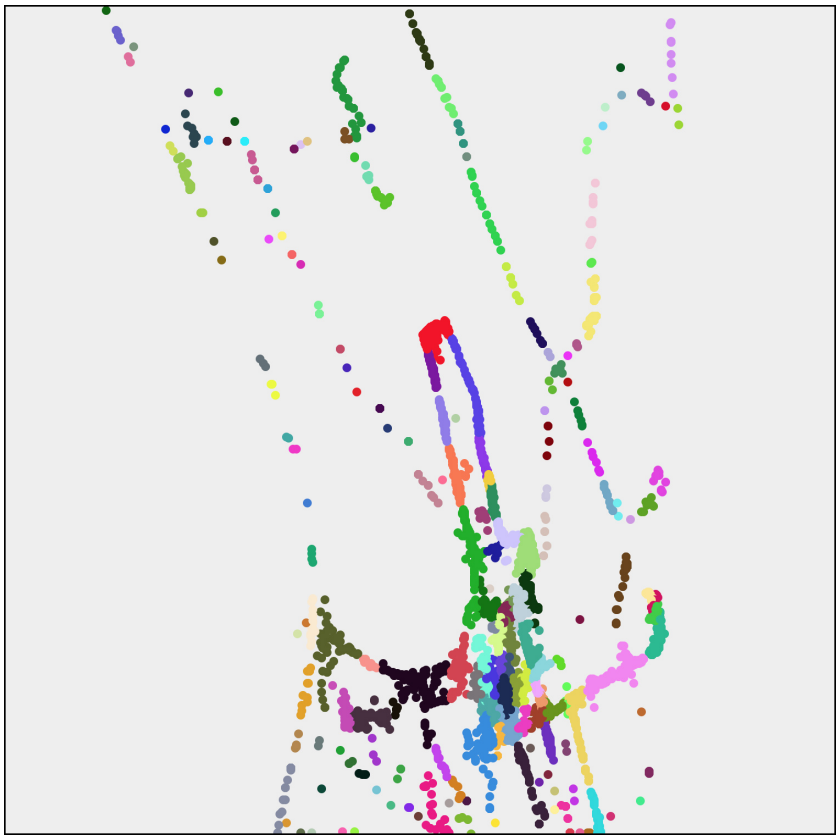


Figure A.3: Lillehammer, Norway

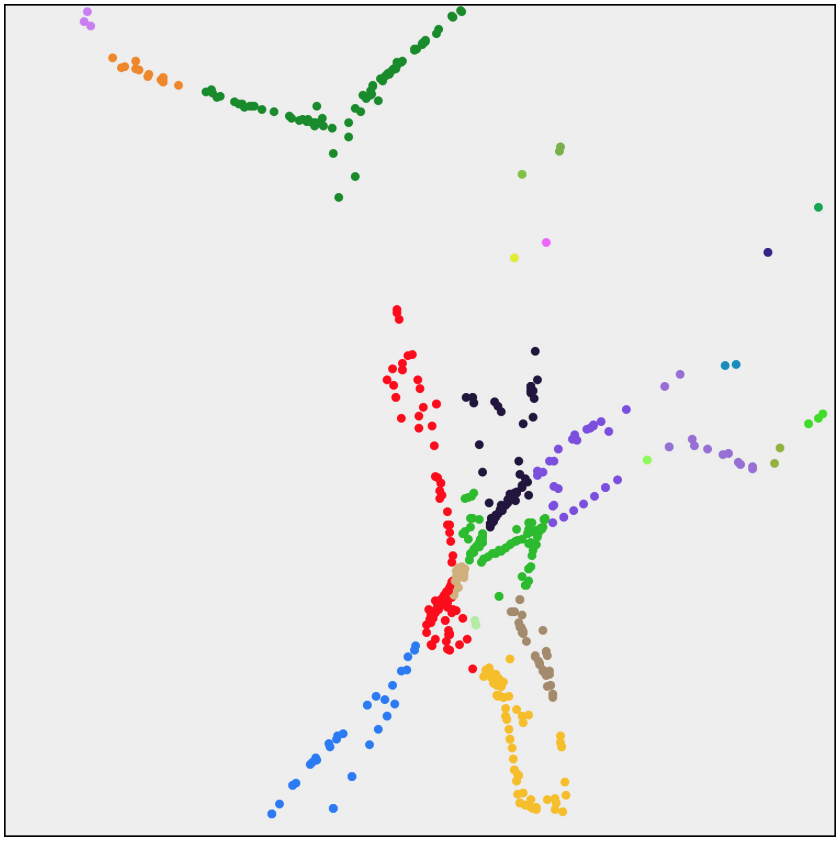


Figure A.4: Tynset, Norway

A.2 K-means Split

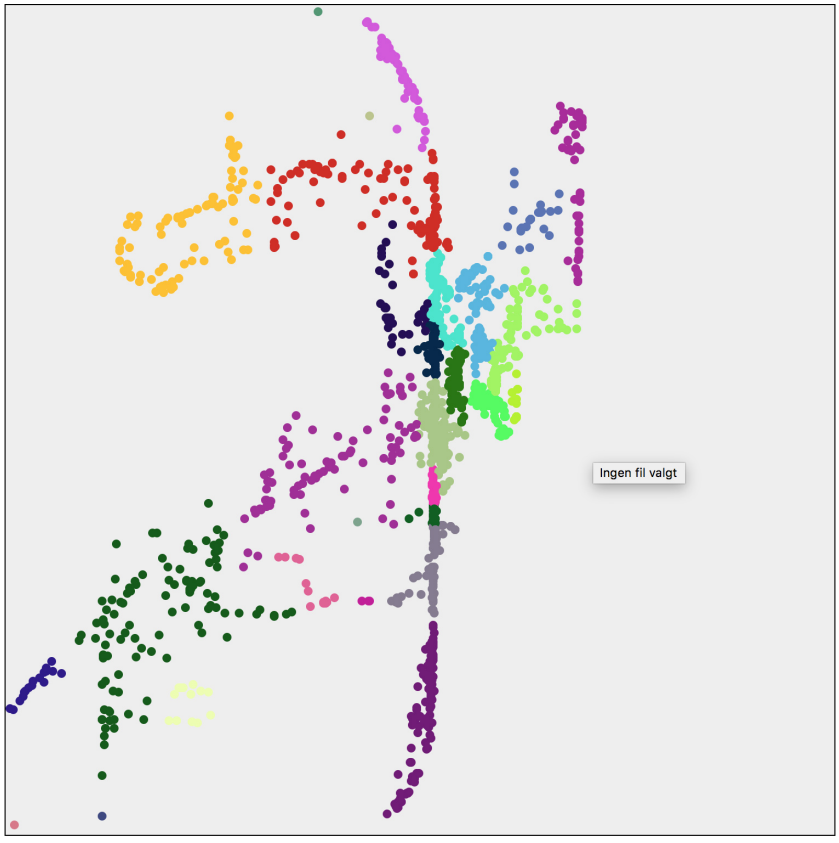


Figure A.6: Forks, Washington, USA

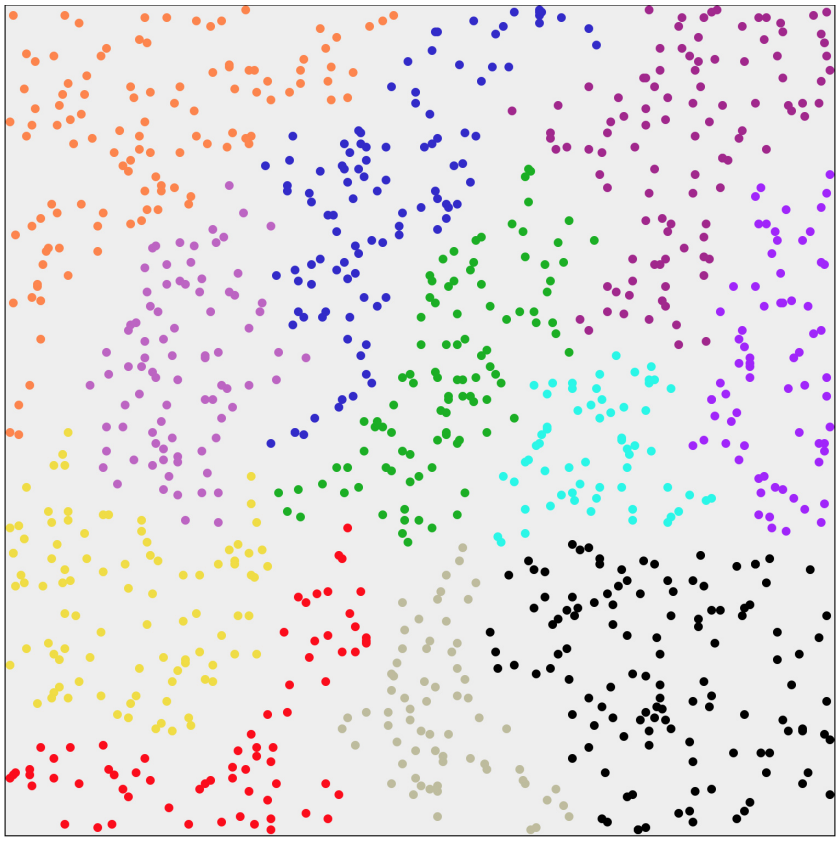


Figure A.5: Uniform distribution, 500x500, 1000 nodes

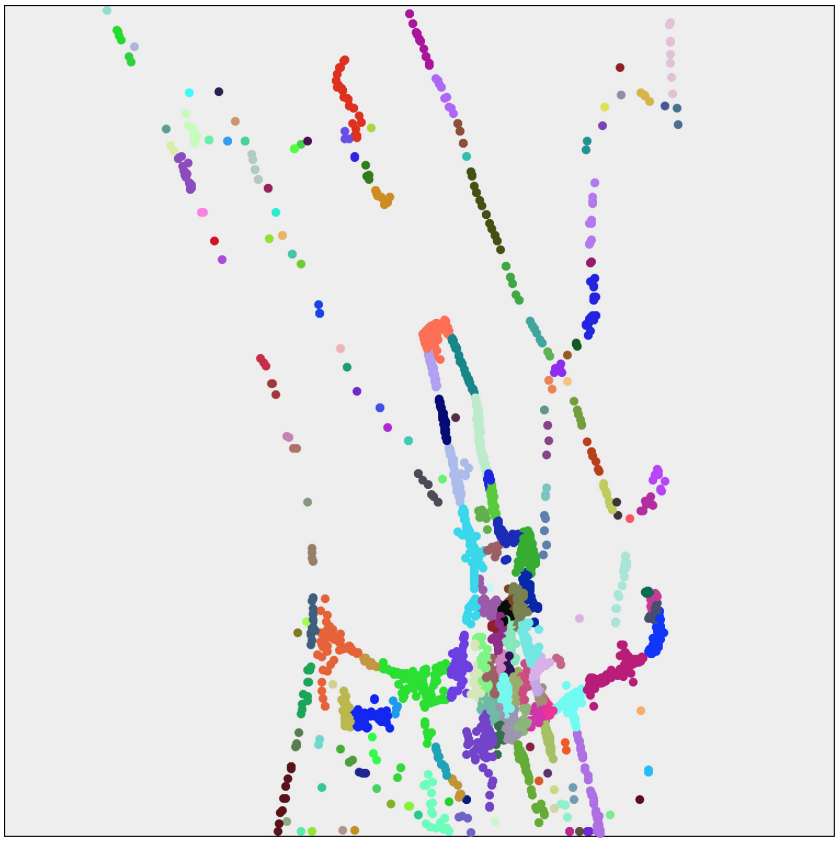


Figure A.7: Lillehammer, Norway

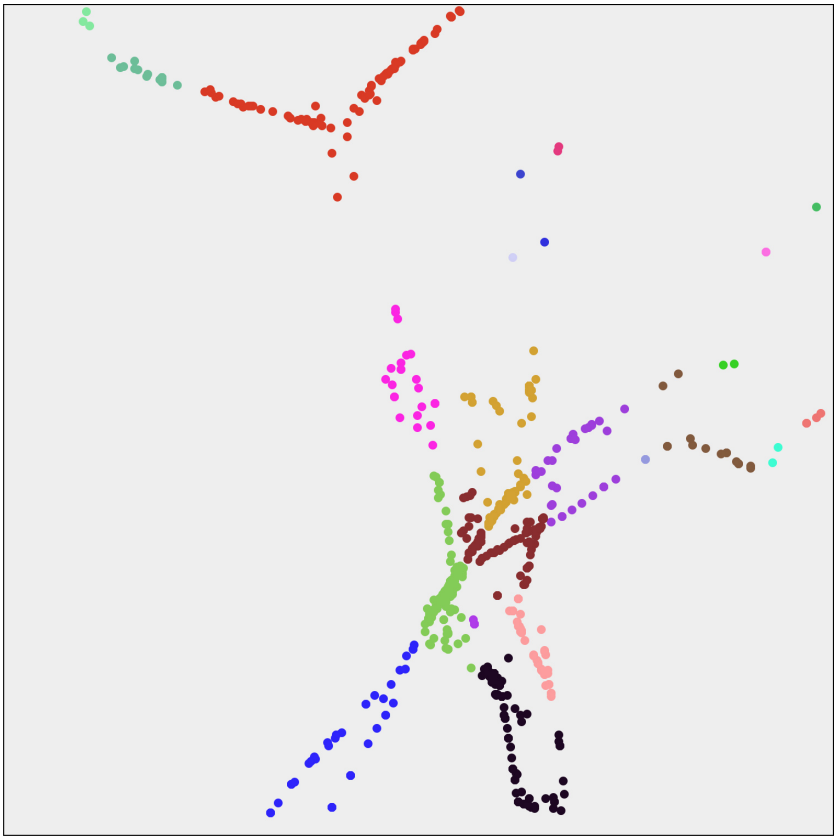


Figure A.8: Tynset, Norway

A.3 Original Minimum Cut Split

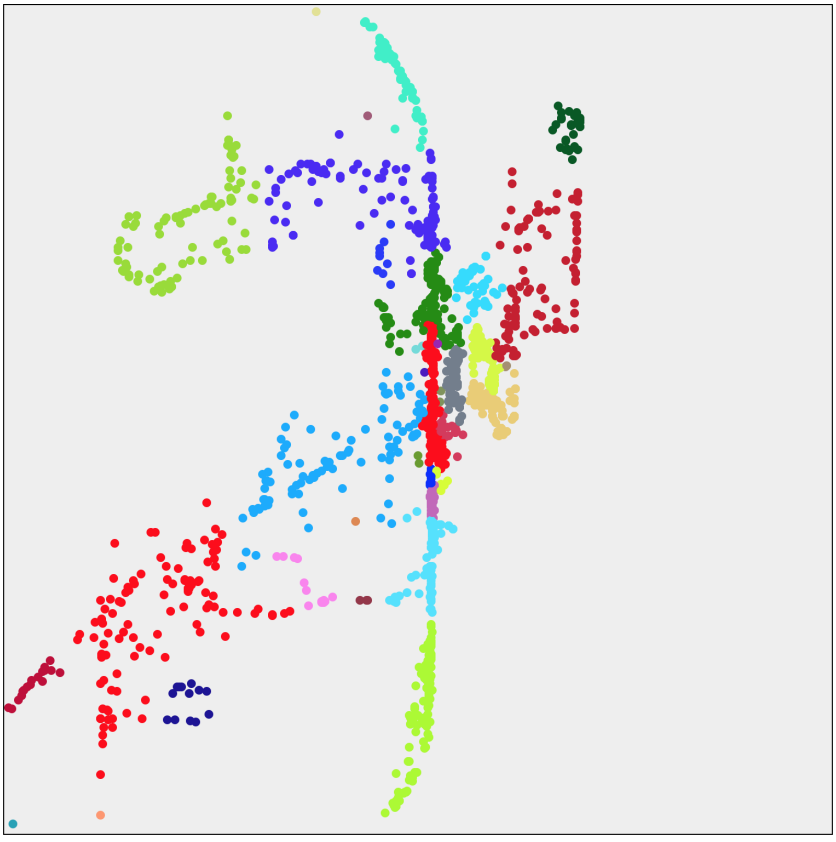


Figure A.10: Forks, Washington, USA

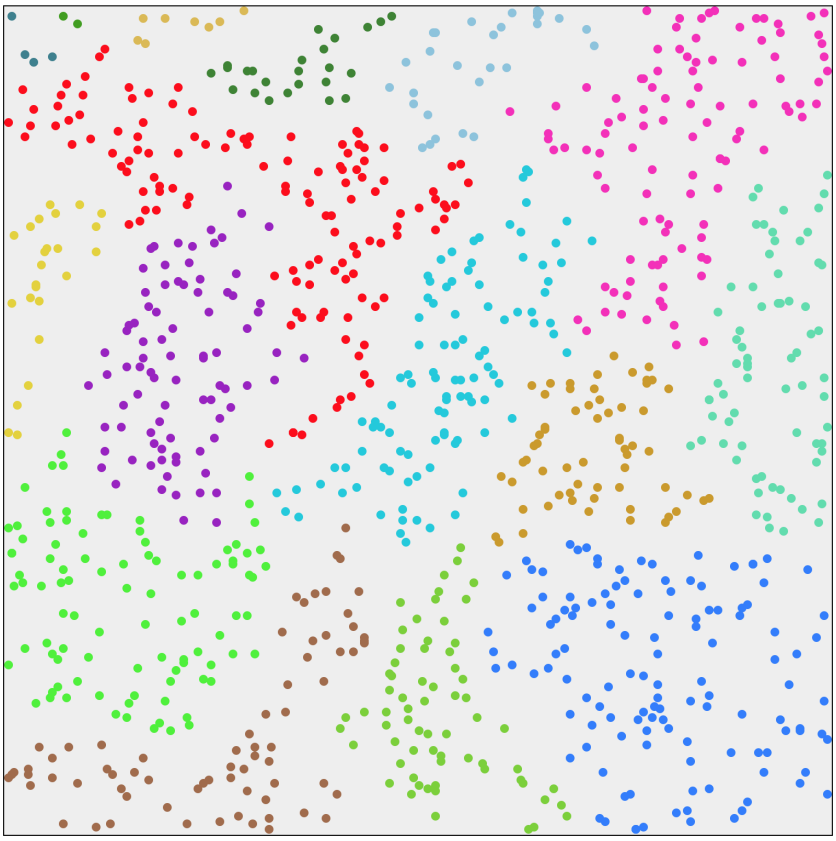


Figure A.9: Uniform distribution, 500x500, 1000 nodes

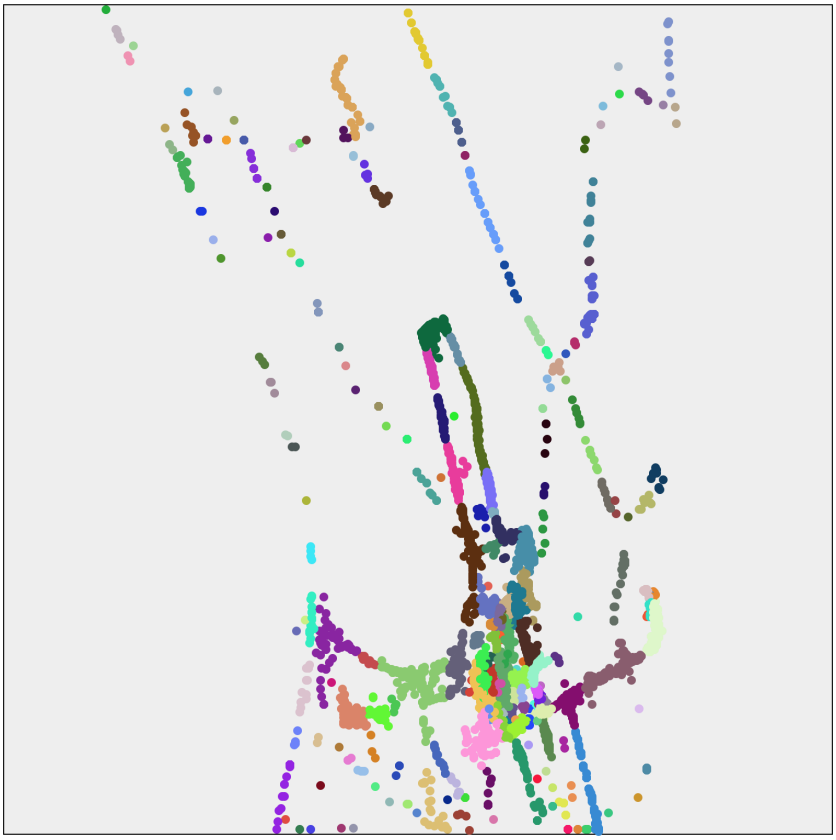


Figure A.11: Lillehammer, Norway

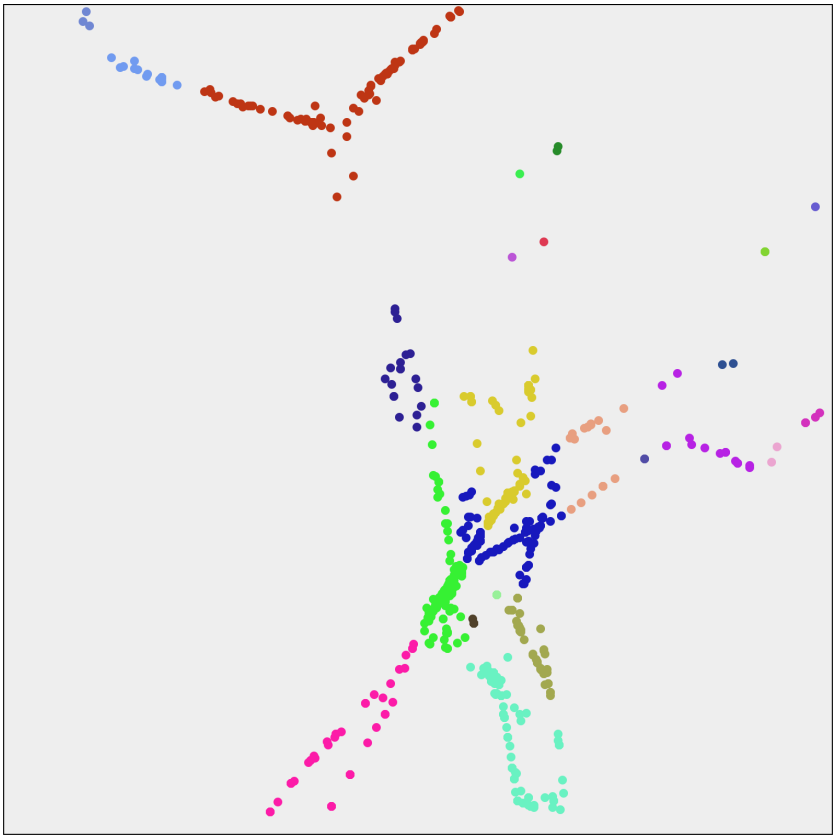


Figure A.12: Tynset, Norway

A.4 Re-evaluated Minimum Cut Split

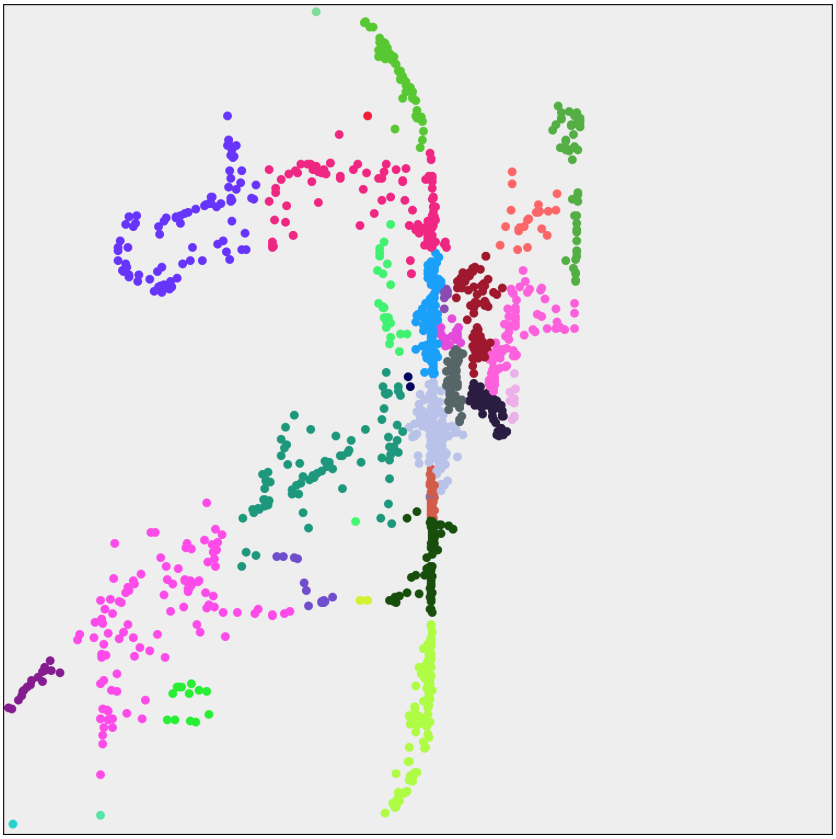


Figure A.14: Forks, Washington, USA

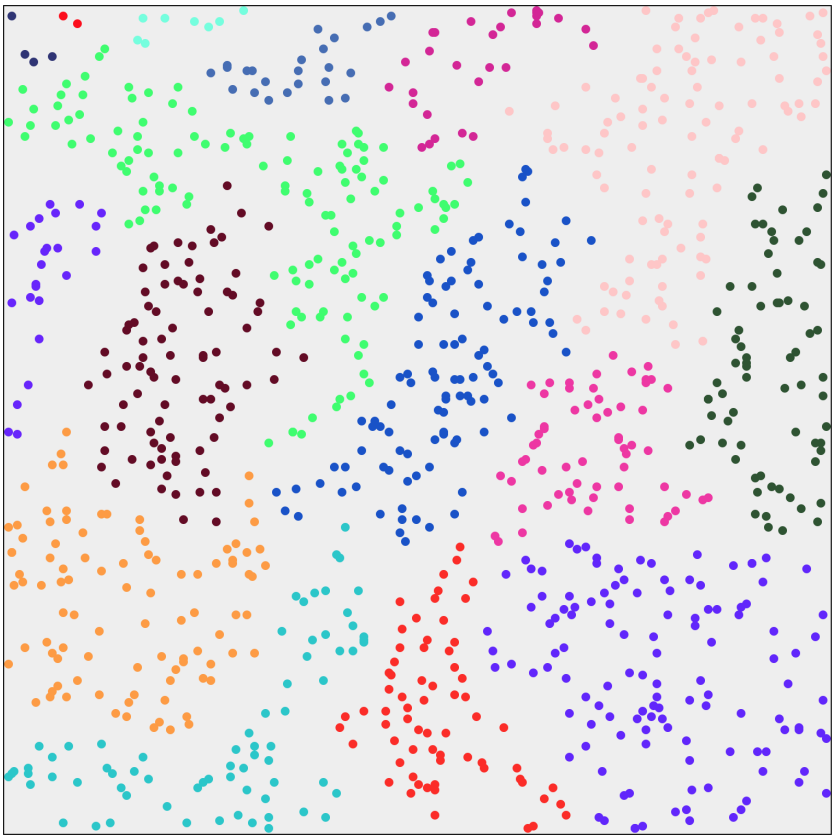


Figure A.13: Uniform distribution, 500x500, 1000 nodes

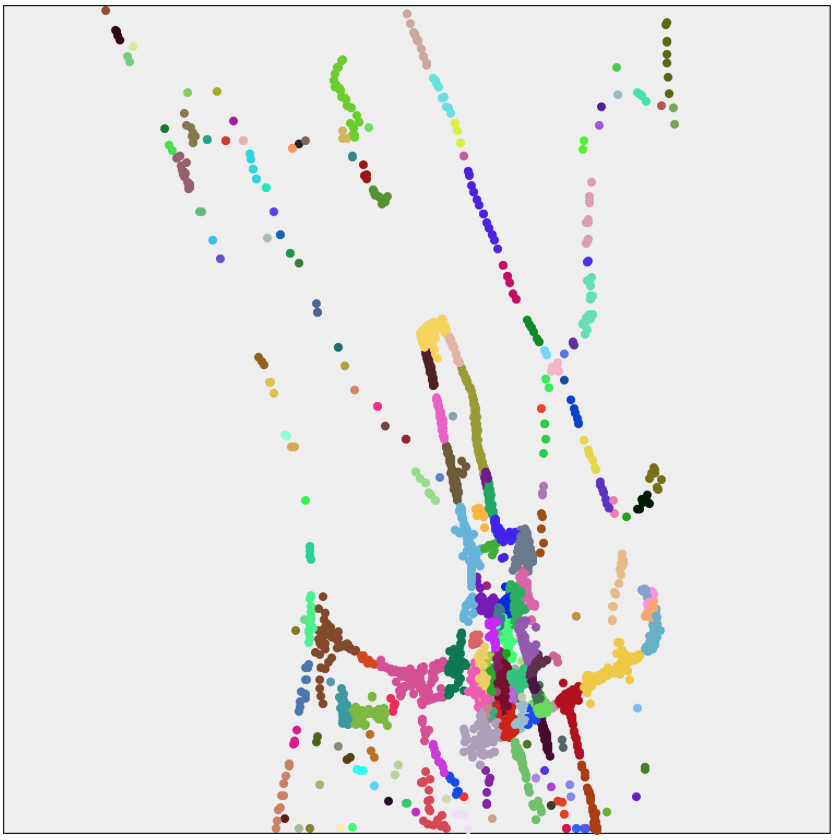


Figure A.15: Lillehammer, Norway

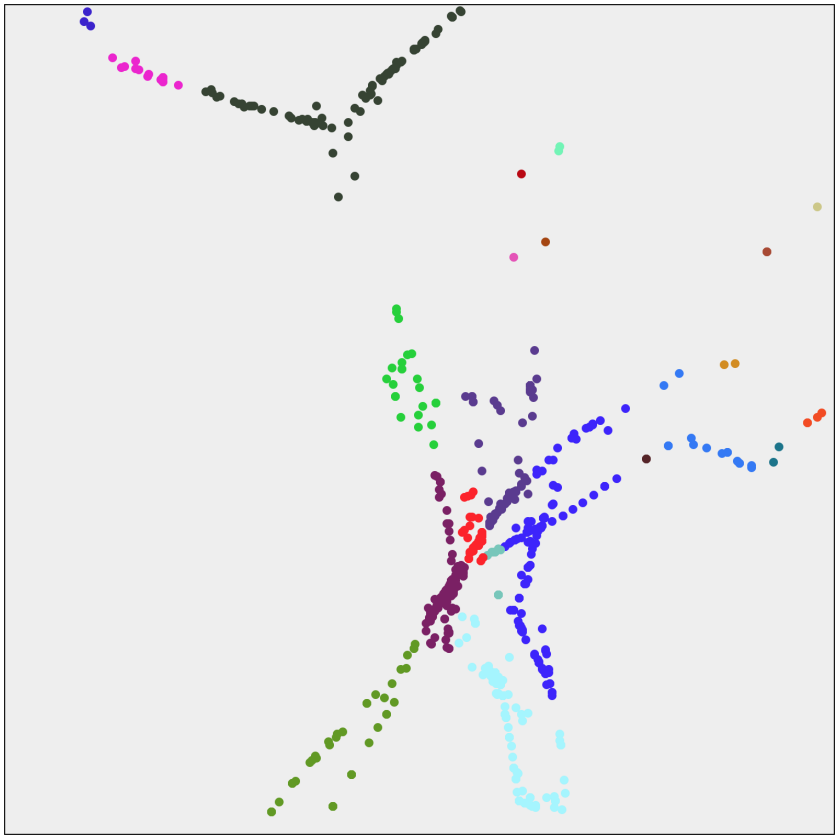


Figure A.16: Tynset, Norway