UiO : Department of Informatics
University of Oslo

Developing a distributed clustering algorithm to enable cooperation between 802.11 access points

Hans Jørgen Furre Nygårdshaug

Master's Thesis, Spring 2018

Developing a distributed clustering algorithm to enable cooperation between 802.11 access points

Hans Jørgen Furre Nygårdshaug

1st May 2018

Acknowledgements

There are many people I want to thank.
First I would like to express my thanks and gratitude to my main supervisor Torleiv Maseng for the opportunity to work with this master thesis and his supervision the last two years. Without your ideas, thoughts and feedback this thesis would not have been possible.
Additionally I would like to thank Madeleine Kongshavn and Magnus Skjegstad for their participation in many phone meetings with me and Torleiv, and providing their thoughts and input on difficult matters.
I want to thank my co-supervisors Tor Skeie and Yan Zhang for agreeing to co-supervising this thesis and putting their knowledge at my disposal.
Finally I would like to thank my girlfriend Martina Langseth Knutsen for her daily support and for taking the time to proofread this thesis.

Abstract

Co-channel interference is the most significant degrader of quality of service in 802.11 Wi-Fi in dense residential areas. The amount of observed neighbouring networks can grow as large as 20-30 networks in a modern apartment building. This makes interference from neighbouring networks a big issue. It can degrade the perceived quality of service to the pointwhere private consumer networks can no longer provide anywhere near the throughput guaranteed by the ISPs' service level agreements. This problem would be easier to solve if access points in residential areas had the opportunity to cooperate and coordinate the distribution of channels in a way similar to what some centralized solutions for enterprise networks offer today. This thesis is focused on developing a clustering algorithm that can define clusters of access points in a distributed and chaotic network topology. These clusters could provide a future framework for communication and coordination between access points. We specify a set of requirements for the distributed clustering algorithm and then go through the development stages of the algorithm, beginning with a minimal working algorithm, and iteratively progress until a satisfactory solution is found that meets the requirements. Lastly, we suggest technologies that could facilitate the deployment of this clustering algorithm in a real world implementation on access points and consider how these technologies could interface with each other to provide the required services.

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 doing so. The latest forecasts estimates that there will be about 27 billion devices connected to the Internet by 2021 [9], and the estimated amount of connected devices in 2017 was close to 8.4 billion [18]. 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 Internet 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 less obvious utilities are also being connected to the Internet. These devices may span everything from lights and air condition systems to coffee machines, fridges and even toasters.

It is not only the numbers of devices that has changed, but 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 internet 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 continuous streams of high bitrate video poses a challenge that can not simply be solved by providing larger bandwidth. Video streams have high Quality of Service (QoS) demands, as buffering of videos and/or reduction of video quality will negatively impact the experience of the service. Additionally, it is

not unusual that different video flows occur simultaneously in the same household. If such traffic is transmitted wirelessly, it can result in a congested wireless environment. Thus, 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 a home is constantly underperforming, it is not unusual for the customer to upgrade the data rate of their subscription or even be advised to buy a new, more powerful router.

Alas, interfering networks are often the perpetrators, and an increase of bandwidth will have no effect. A new router with more transmission power can help mitigating the issue for one customer, but resulting in even worse conditions for a neighbouring network. 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.

Because of the few transmission frequencies available for Wi-Fi, the 802.11 protocol specifies a set of rules which only allows one device in the vicinity to transmit on a channel at the same time. This presents the challenge of managing the transmission channel and transmission power level of access points.

Large enterprise networks can deploy access points throughout a corporate environment which are centrally managed by dedicated controllers. These controllers can run algorithms and heuristics that based on information acquired from all access points, can control and optimize channel allocation for every access point under their control.

This is not the case for the chaotic deployment of access points in residential areas. If for instance an optimal channel distribution in e.g. an apartment building could be computed, the experienced QoS would undoubtedly increase. As of today this can not be done automatically: all decisions have to be made relying only on the local observations at each access point, as no access point knows the condition of other access points in its surrounding area. A suggested channel allocation algorithm from 2004 [26] introduced the idea of using a flooding mechanism to build a graph of all surrounding access points at each access point, and use this list compute an optimal channel distribution. The idea is promising, and might have worked with the limited deployment of Wi-Fi in 2004. Today this mechanism could result in tens, if not hundreds of thousand access points in the list. Not only could the flooding mechanism cause network congestion, but it would be computationally infeasible to attempt to let all access points in e.g. an entire city cooperate on channel allocation.

This thesis is focused around the problem of designing, developing and simulating a distributed clustering algorithm that when deployed in a chaotic landscape of wireless networks, can identify clusters of access points and run continuously to update the clusters when new access points appear.

1.1 Motivation

To prevent co-channel interference on the wireless spectrum, some sort of coordination and control between access points would be beneficial. This can be done by identifying smaller clusters of access points that will exchange information between each other. A central controller could possibly do this, but the deployment of Wi-Fi in residential areas is inherently chaotic. Centralized controllers can be hard to scale, are single points of failures, and would be difficult to realize when customers are largely subscribed to different Internet service providers and use their own router brands.

A distributed and decentralized method could provide a better solution. For this a distributed clustering algorithm is needed to identify collaborative groups of access points. These groups should contain the access points that would benefit the most of being grouped together, hence minimizing the interference between the groups without exceeding a feasible maximum amount of nodes inside each group. When such groups have been identified, information about channel interference and QoS could freely flow within the group to enable better channel allocation.

1.2 Problem statement

As described by the introduction and motivation section, defining clusters of access points could enable distributed cooperation and coordination between access points. In this thesis we will address the following problems related to the creation of such clusters:

1. *Defining requirements for distributed clustering*
   This type of clustering problem will be different from traditional clustering. Access points have limited knowledge of the surrounding environment, and other restrictive properties may apply. Hence we should define what requirements applies to the distributed clustering algorithms.

2. *Design and code a clustering algorithm that meets the requirements*
   After each algorithm iteration, simulations of the algorithm should be done. Then we can observe results and adapt the clustering algorithm. Data about network topologies has to be made to enable such simulations.

3. *Suggest the technology stack needed to enable this algorithm to run in a real world implementation*
   When the clustering algorithm has been developed and evaluated, some considerations for what technology and protocols needs to be in place to enable an implementation of the algorithm to run on real-world access points.

1.3 Method

First we will perform data mining to provide simulation data. The report "Computing as a discipline" by the ACM Task Force [12] introduces 3 paradigms for the software engineering discipline. Throughout this thesis we will follow the design paradigm from the report, to build the programs required to address the problems presented in this thesis. We will introduce the requirements for a distributed channel allocation algorithm, and then pick a simplistic algorithm to evaluate towards the requirements. Based on simulations observations we will iteratively modify the algorithm until all requirements are fulfilled.

1.4 Outline

The structure of the thesis is as follows:

- **Chapter 2: Background**
  In the background chapter we introduce the mechanisms of wireless technologies and the 802.11 standard. We look at why interference impacts Wi-Fi so hard, and what countermeasures exists. Finally we introduce two clustering algorithms that will be used in this thesis.

- **Chapter 3: Related Work**
  Here we consider the work that has been done to minimize the impact of co-channel interference in different centralized solutions. We also look at technology that is aimed to work in a chaotic, distributed environment, that can possibly facilitate clustering in the real world.

- **Chapter 4: Data acquisition and data structure**
  This chapter is dedicated to the creation of artificial network topologies that will be used to test algorithms under development. We present the data structure for how network topologies will be stored, and create a program that can visually represent the network topologies in a browser.

- **Chapter 5: Access point clustering**
  Here the we phrase the requirements and specification of a distributed clustering algorithm fit to run on wireless access points. Then we use the data from chapter 4 to iteratively build and evaluate a clustering algorithm.

- **Chapter 6: Group communication and state synchronization**
  This chapter is dedicated to looking at technologies and protocols that could

facilitate a real world implementation of the algorithm suggested in chapter 5. We also suggest an abstract architecture of how these components could coexist.

- **Chapter 7: Conclusion**
  Finally we conclude by looking at what has been done, and consider if the problems in the problem statement have been addressed. Then we look at possible future work in this area, and what remains to do before the clustering algorithm can be implemented in the real world.

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

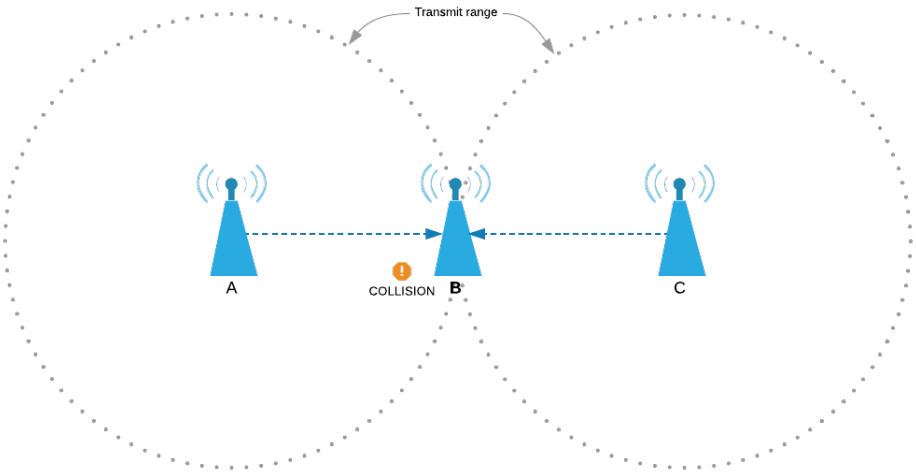
There are some challenges with wireless technologies that are harder to overcome when comparing to wired transmission technologies like Ethernet.

The first step to achieve a successful transmission is making sure the receiving radio is within transmit range of the transmitter. The transmit range is decided by the power of which a 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 more compromised range.

Even if the surrounding environment is mostly open space, the wireless signal becomes subject of attenuation, which is a physical property of electromagnetic 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 at 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

This section provides a brief introduction to network infrastructures, to get an understanding of the differences between the mainly two types of infrastructures used today.

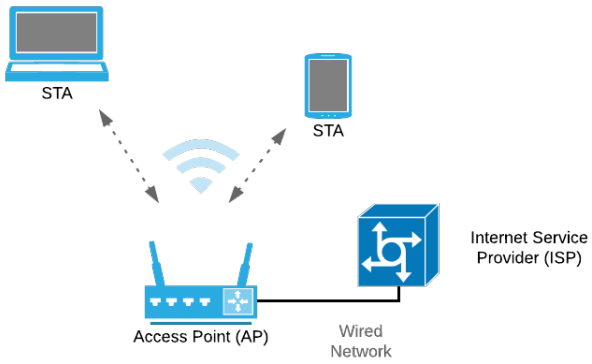


Figure 2.2: Minimalist Basic Service Set in infrastructure mode

2.2.1 Basic Service Set

A basic service set (BSS) in infrastruture mode consists of a single access point (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 [11]. A minimalist BSS infrastructure is illustrated in 2.2. It is 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 achieve 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 a station (STA) can be in the basic service 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

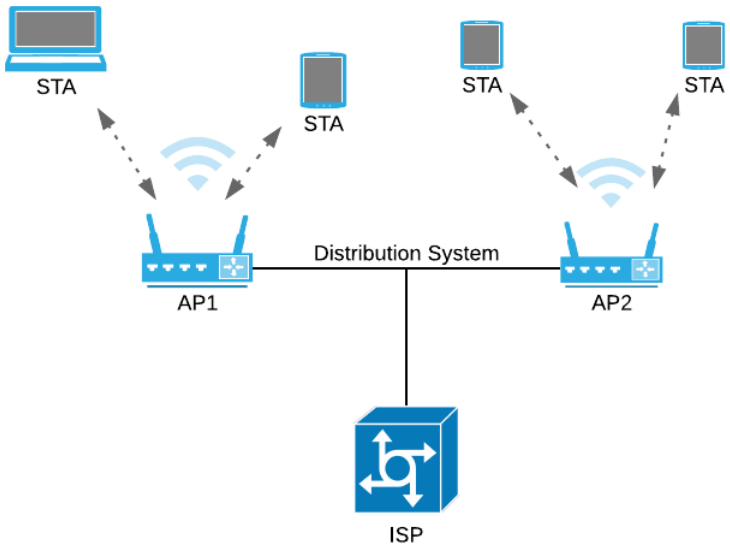


Figure 2.3: Extended Service Set

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 described here.

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 containing 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 (DIFS), 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 have 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 on the same network. Figure 2.4 shows the frame exchange that happens. Following

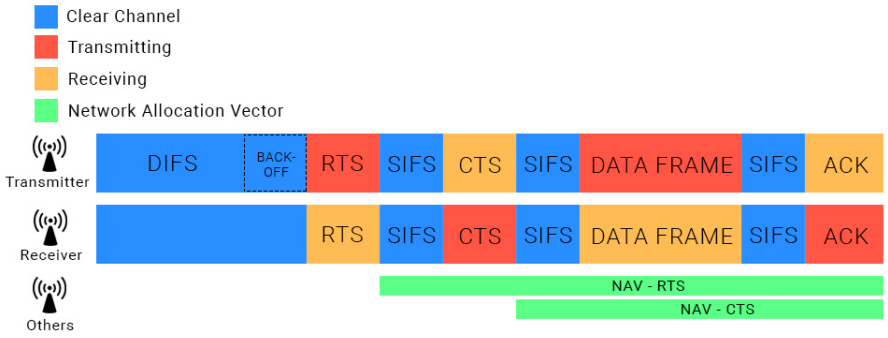


Figure 2.4: The timeline one frame transmission cycle in DCF mode

the figure, the transmitter has to wait DIFS time, before drawing a number from the contention 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 successful 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 it on or off [4].

2.4 PHY Layer

The physical layer in 802.11 is also divided in two sub-layers. The upper sub-layer is the Physical Layer Convergence Procedure (PLCP), responsible for clear channel assessment and acting as a common interface for MAC layer drivers. The lower sub-layer 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). 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. This is the structure of a PPDU header:

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

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

2.4.2 Clear channel assessment

The PLCP layer provides clear channel assessment. The purpose of clear channel assessment is to give information to the MAC layer if the medium 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 nonetheless 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 observed 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 frequency at the same time. A receiver will have problems deciding which parts of the signals belongs to which transmitter, and the signal may be 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 in response. Additionally to waiting the minimum EIFS time, the station also has to wait for an idle channel indication from the PLCP layer again.

This is just the added delay if a collision happens. Of course, the more co-channel interference, the more transmitters are waiting for medium access, meaning the contention window will increase, and the frequency of which a transmitter is granted access to the medium will be reduced. Thus, interference is severely damaging for QoS on wireless networks.

2.5.2 Countermeasures

Several countermeasures to limit the impact of RF-interference have 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 sends 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 aggregation 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 [20].

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 [21]. 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 2.5. 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 2.5: Channel distribution

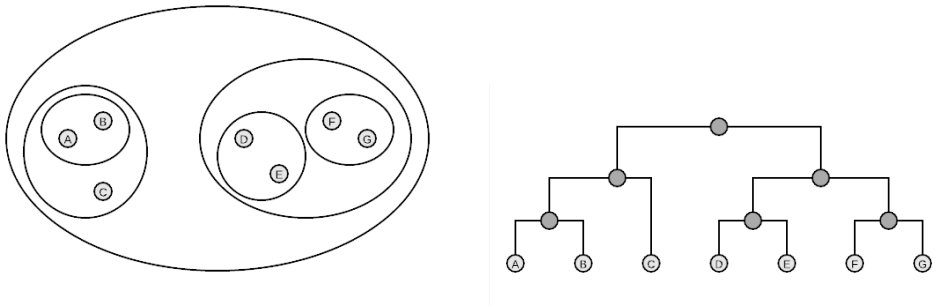
CHNL_ID	Frequency (MHz)
1	2412
2	2417
3	2422
4	2427
5	2432
6	2437
7	2442
8	2447
9	2452
10	2457
11	2462
12	2467
13	2472
14	2484

2.7 Relevant clustering methods

This section introduces some of the clustering methods that will be used in this thesis.

2.7.1 Hierarchical Agglomerative Clustering

Hierarchical agglomerative clustering works in a bottom-up, greedy approach. This means that the algorithm begins at the bottom node level, instead of considering the entire topology at once. 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 [22]. It falls under hierarchical clustering category, because the cluster subsets are iteratively being built for each merge and a dendrogram that encompasses all sub-clusters can iteratively be constructed. An



(a) Cluster formation (b) Dendrogram representation

Figure 2.6: Agglomerative clustering

example of agglomerative clustering used on two dimensional data points can be seen in figure 2.6 with its respective dendrogram.

Distance metric

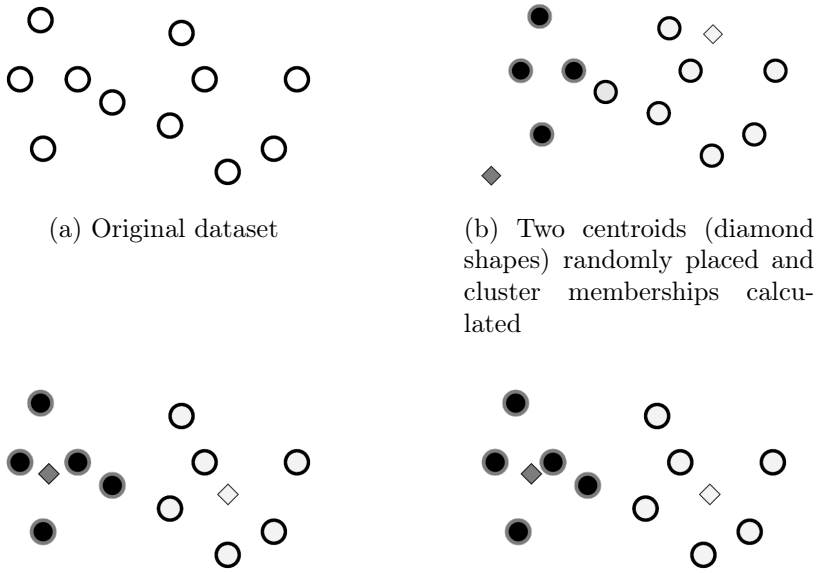
The distance metric lets us specify exactly what the distance between two clusters signifies. As each cluster usually consists of several nodes, there are multiple options for what the distance between two clusters could be defined as. The distance metrics used in hierarchical clustering are typically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

2.7.2 K-means

The standard K-means algorithm, also called Lloyd's algorithm as it was first suggested by Seth Lloyd in [24], is a clustering algorithm often used for cluster analysis, but also classification purposes in machine learning and unsupervised learning [10]. 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).

In the default version of Lloyd's algorithm 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 the same centroid becomes part of the same cluster set. When all data points are associated with a cluster, each cluster computes



(a) Original dataset

(b) Two centroids (diamond shapes) randomly placed and cluster memberships calculated

(c) The positions of centroids and the cluster memberships are updated

(d) The clusters have converged

Figure 2.7: Illustration of the K-means clustering algorithm

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 [25].

A well known problem with K-means is that the result strongly depends on how the algorithm is initiated. It always converges to the local minimum, and has no way of transcending this local minimum This problem can be reduced by running K-means several times and choosing the best result of all the runs.

Chapter 3

Related work

This chapter is dedicated to give a brief account of works that addresses the issue of controlling and managing 802.11 wireless access points. It is constrained to technologies that in some way provide channel allocation or other ways to reduce RF-interference using more information than what is available at each individual access point.

3.1 Centralized Solutions

3.1.1 Cisco RRM

CISCO offers an RRM (Radio Resource Management) solution directed at enterprise networks [8]. Their architecture consists of one or more Wireless Lan Controllers (WLCs), which performs the responsibilities of regular APs, but also performs RRM, through grouping, leader election, dynamic channel assignment and transmit power control. Regular APs that are not WLCs, tunnel all their packets through their associated WLC unit [36]. There are many finesses and details to the Cisco RRM, here we will only present the major concepts.

RF-Grouping

In CISCO RRM, the RF-grouping stage is the identification of controllers and APs that are under the administrative control of the same RRM. To identify the RF-group an RF-group name is used. The name itself is an ascii string, and the APs obtain it from the WLC they are connected to. The WLCs have to be preconfigured with this group name.

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 broad-

cast message to the controller, and then rebroadcasts the packet similar to the flooding routing mechanism. After all APs has done this, the RF-group can identify all APs that can physically hear each other. This is called the neighbour discovery stage, and APs within physical hearing of -80dBi and above makes up what Cisco calls an RF-neighbourhood. If applicable the RF-groups perform leader election to decide which controller becomes the RF Group leader, but a preconfigured leader can also be decided. The RF group leader has the responsibility of running the relevant channel allocation algorithms and adjusting the radio power level of the APs.

Dynamic Channel Assignment

To perform Dynamic Channel Assignment, the RF-leader has the neighbour lists for all access points in its RF-group. All APs track a set of metrics which tells something about its transmission quality. The metrics are:

- Co-channel interference from RF-group
- Co-channel interference from rogue APs (not a part of the group)
- Background noise that does not originate from 802.11
- Channel load measured at the physical layer
- Sensitivity to change (user selected variable to indicate how fast the network should respond to bad quality)

These metrics together end up in a unified metric called the Cost Metric, and the Cost Metric is then reported to the RF-leader. The DCA algorithm looks at the AP with the lowest cost metric first, meaning the AP with the worst QoS. The transmission channel can be changed for this AP, and all 1-hop neighbours of this AP, but no AP further away than 1-hop. This is to prevent impacting APs throughout the RF-group. That process is repeated, computing several channel plans, and at last all the channel plans are evaluated through a cost function to see if the Cost Metric has improved for the AP, without degrading the Cost Metric for neighbouring nodes. If an improvement is seen then the channels are changed, and the algorithm moves on to the next AP on the list of low Cost Metrics.

3.1.2 DenseAP

DenseAP described in [27] 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 will naturally reduce the throughput on the medium because of the CSMA/CA protocol. 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 (Received Signal Strength Indication) measures, co-channel interference, and associated clients. Based on this information the DC decides which DAP each client should associate with, and also which channel each DAP should transmit on.

3.1.3 HiveOS

HiveOs [28], 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 a 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 withholding 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 to determine which channel has the lowest amount of interference. 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 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 from the PPDU header, channel utilization, and the utilization

of overlapping APs. Based on all these metrics, each channel gets a channel cost, and the HiveAP picks the channel with the lowest cost.

3.1.4 Comparison and assessment

An important difference between HiveOS and the other two solutions is that HiveOS connects to Aerohive's cloud service to perform tasks as association and load balancing. This eliminates the need for a central controller, and largely removes the risk of being physical single points of failures, albeit they still rely on the Aerohive service which still is a single point of failure. HiveAP's channel allocation scheme uses mostly local observations to control the selected channel, but if there are other HiveAPs nearby, the channel selection is negotiated between the HiveAPs based on the amount of neighbours each AP has. This is different from the Cisco system, which optimizes channel allocation by adjusting the channel of 1-hop neighbours as well. This assumes that nearby APs are under the same RF-group as each other.

While both HiveOS and Cisco strives to optimize the channel plan and control of the transmission power of the antennas, DenseAP suggests having more APs and control client association. This technology focuses less on optimizing the channel plan, and more on providing many different channels in the same area.

What is common for all solutions is the need for proprietary hardware for it to work. It does not fit well with the nomadic routers today, where house and apartment owners bring their own routers and access points when they move. HiveOS has arguably the model that would work best in residential areas, as the introduction of more HiveAPs would only increase the benefit of their channel allocation scheme.

The use of RF-neighbourhoods in Cisco RRM is a testament to the importance of being able to define collaborative access points that in some way enables reevaluation of the channel distribution throughout the entire neighbourhood. Such RF-neighbourhoods are what we seek to define with a distributed clustering algorithm.

3.2 ResFi

Description

ResFi [37] is a protocol that aims to enable self-organized management in residential deployments of wireless LAN through communication between access points.

ResFi assumes that all access points have two interfaces, one connected to a wired backhaul (like the Internet), and another 802.11 compatible wireless interface. ResFi communicates with the device that interfaces with the antenna, which in most residential homes will be a router. This controls the operating channel and the transmission 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 requiring them to be a part of an extended service set. If two access points are running ResFi and are within physical hearing range of each other, then ResFi establishes a secure communication channel between these APs over the wired backhaul. ResFi enables all of this without doing any modifications to hardware and drivers (like modifying standards or requiring proprietary equipment).

The services provided by ResFi are 1-hop unicast communication and n-hop broadcast messaging between access points. These services can be valuable for an implementation of a distributed clustering algorithm, where access points may need to perform message passing between each other to coordinate and form groups.

ResFi uses a modified version of hostapd (host access point daemon), a user space application, which means any protocols that runs on top of ResFi does not require modification to kernel modules or drivers. This is unquestionably a huge benefit for the development and deployment of new services that benefits from communication between access points, eliminating the need for routers with a specific firmware to be compatible. We will take a closer look at this technology in chapter 6, where we more closely discuss how it can be used to enable the communication channel between access points to run a distributed clustering algorithm.

3.3 Channel allocation using DSATUR

??: This section is dedicated to previous work done using the DSATUR heuristic.

DSATUR (from degree of saturation) is a heuristic created by Daniel Brélaz [5] to find solutions to the NP-hard problem of coloring the vertices of a graph so that no adjacent vertices share the same color. With a network topology mapped out, this heuristic can be used to optimize the distribution of channels. This is done by treating access points as vertices in a graph, and different channels as colors. Vertices connected by an edge indicates that the two access points can physically hear each other, weight of the edge being the strength of which they can observe each other. An important property of DSATUR its deterministic nature. Meaning that if different access points have an equal graph, then all access points will compute the same channel distribution.

Channel allocation schemes relying on the DSATUR algorithm have been proposed before. In 2004 a paper was published, called "Automatic channel allocation for small wireless local area networks using graph colouring algorithm approach" [26].

The paper suggest building the graph by using flooding, and its done in the following way:

- When an access points boots up, it waits for 802.11 beacon messages from other APs to discover its neighbours. A neighbour list is built using their

MAC-addresses.

- After all neighbours have been discovered, it broadcasts its list of neighbouring MAC-addresses to all neighbours.

- When an AP receives a broadcasted neighbour list, it rebroadcasts the list to all its own neighbours. The information from the list is used to build the network graph.

To prevent eternal propagation of the same message, all nodes only rebroadcast a list one time, and to keep the lists up-to-date, each node has to broadcast new lists periodically. When the protocol converges, all access points in the area should be a part of the graph, and the graph should be equal for all access points. The DSATUR algorithm can then be used to compute the channel distribution.

This protocol does in fact identify a cluster of access points. However, the flooding mechanism used to identify clusters would most likely not work well today. In 2004 the low density of access points meant that the protocol would converge quickly, but in some areas today it might never converge because of the vast amount of access points. If it did converge, the amount of nodes in the graph could potentially be way beyond what is plausible to solve when the problem is NP-hard and with the computational power of an access point.

What it lacks is a way to constrain the size of the graph, and a method to distinguish more disturbing nodes from nodes that cause little interference.

3.4 Distributed clustering in ad-hoc networks

This section is dedicated to briefly address distributed clustering algorithms used in ad-hoc networks.

In ad-hoc networks all access points in the network have to implement a routing mechanism with the goal of establishing end-to-end communication between APs without a wired backhaul network. As all APs have to act as routers, they have the roles of sending routing messages, as well as relaying data messages going to and from neighbouring nodes. If the ad-hoc network structure is flat, routing does not scale well. Thus, just as the Internet, ad-hoc networks benefit from creating a hierarchical topology. Algorithms such as DCA, DMAC and GDMAC [3] seek to partition network topologies into clusters to form a hierarchical network structure. Common for these algorithms is that information flow between clusters go through so called Cluster Heads (CH). These clustering algorithms focuses on finding CHs that will hold routing information and topology information. The purpose is that

every neighbouring node can route their inter-cluster traffic through the CH node, to simplify addressing and reduce routing traffic [19].

According to the paper which describes the previously mentioned algorithms [3], the following properties have to be satisfied for any ad-hoc clustering algorithm:

1. Every node in the topology that is not a CH, has a CH as a neighbour.
2. All nodes select the neighbouring CH which has the highest weight
3. CHs can not be neighbours

These clustering algorithms are mainly concerned with identifying 1-hop cluster heads. Hence, even though the naming is similar, the goal of these clustering algorithms is different from the purposes of the distributed clustering algorithm we develop in this thesis.

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 simulate the distributed clustering algorithm. 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.

A testbed would also be beneficial for development, but it would require a large amount of routers and is not within the scope of the thesis.

4.1 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 of wireless access points. 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 state is it in then? Well, it is hard to say without having a bird's eye view of a 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-existent. Nor will a

full-fledged architecture be presented in this thesis (even though suggestions will be made in chap 6), 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. One of the major reasons for performing group creation in the first place is to be able to collaboratively allocate channels within the group, hence 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 neighbouring 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 be a part of this model. In all succeeding simulations nodes are transmitting with equal strength, and the environment is flat and obstacle free.

Requirements

This subsection describes the requirements for a program that will both generate and represent Wi-Fi network topologies. Having a clear specification of the 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 computed 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 decibels relative to isotropic (dBi).

[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 a unique id.

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. The unit of the axes is meters.
- Number of nodes to place on the topology
- Minimum distance between nodes (in meters). This is only to avoid unlikely placement and extreme interference of nodes that are placed on top of each other.
- Minimum loudness measured in dBi for a node to account another node as a neighbour (e.g -100 is too low for anyone to hear).

4.2 Program design

4.2.1 Primary functionality

The resulting program consists of two main functionalities.

The first functionality is creating a topology and generate nodes which are uniformly and randomly positioned on the network topology. The topology size, node 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 signal stength 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 [17], can be used to derive the formula for the Free Space Path Loss (FSPL) in dB.

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

```
#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: Pseudocode for computing the signal strength between nodes

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 signal strength calculation can be seen in figure 4.1.

Program result

The resulting program, written in Python 3 [16], 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 standardize 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[2].

[2] `https://github.com/hansjny/GroupSimulations/blob/master/GenerateTopology.py`

```
{
  "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

4.2.2 Data output and visual representation

The result of the topology generation is stored in a JSON [23] 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.

4.3 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.

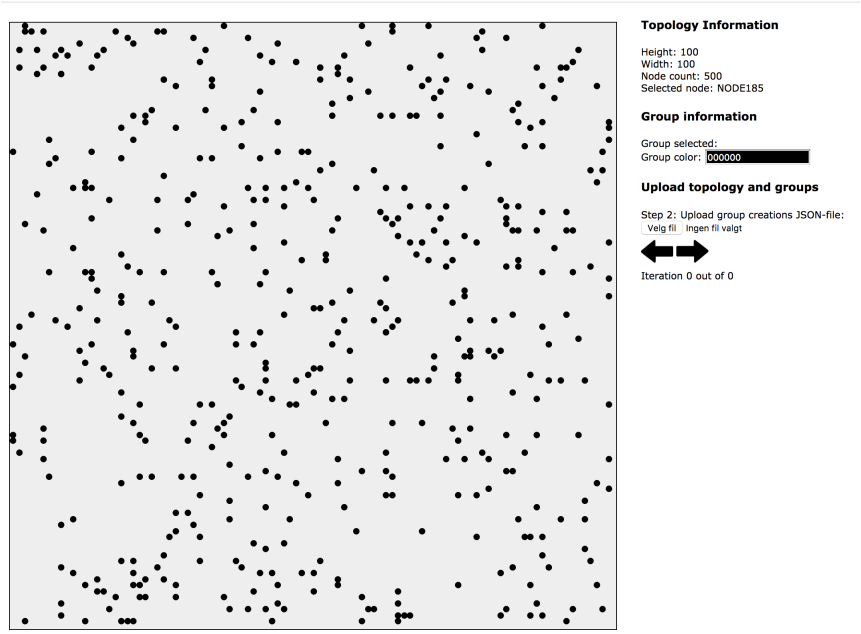


Figure 4.3: Generated topology with random, uniform distribution and the interface for viewing topologies

4.3.1 Introduction to WiGLE

The Wireless Geographical Logging Engine (WiGLE) [35] 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[3], 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 WiGLE openly supports research projects. This is good news for us, and after contacting them they upgraded the account which will be used for data gathering to a higher daily data limit.

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

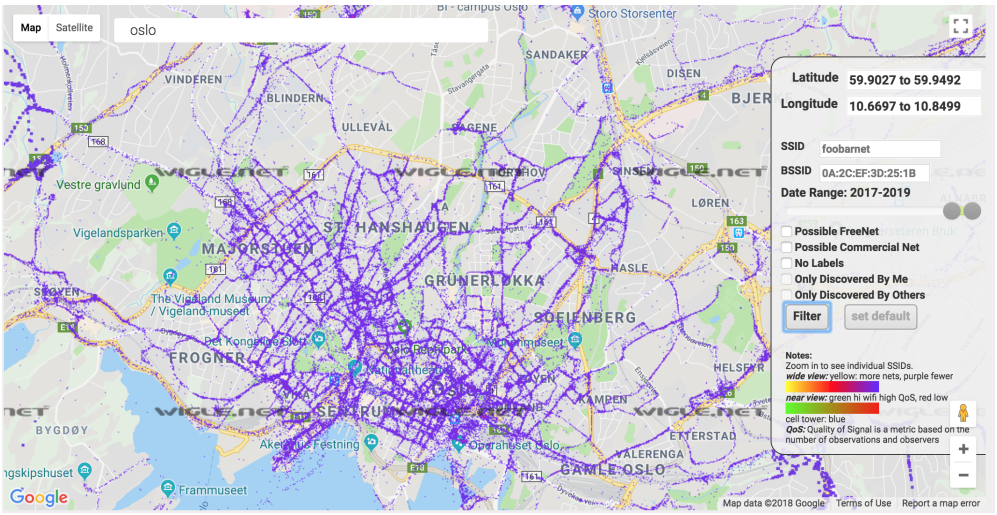


Figure 4.4: WiGLE map of Wi-Fi networks observed in Oslo from 2017-2018

4.3.2 Data quality

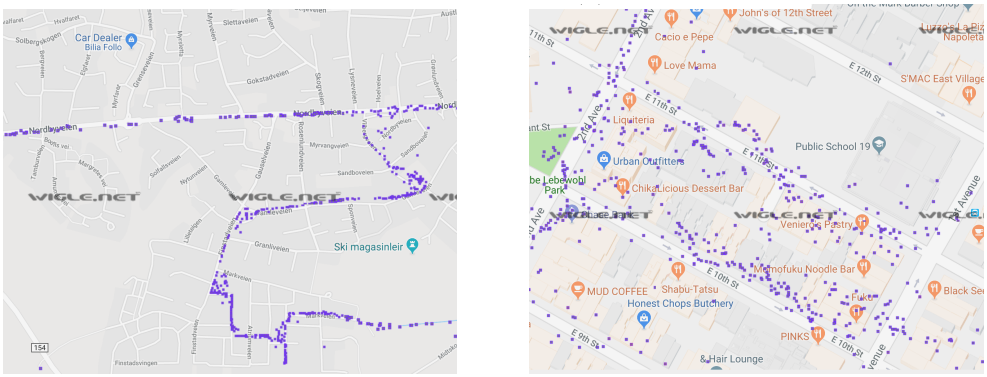
WiGLE determines the geographical coordinates of the position of each access point using weighted-centroid trilateration [31]. This means the AP location is not necessarily accurate: if the measurements of an access point's signal strength is done in multiple places around the access point location the access point location becomes very precise. On the other hand, if the measurements are taken on only one side of the access point, the 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 done from the road when wardriving, this phenomenon 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.

4.3.3 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



(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

available at Wigle's API reference [34].

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 *latrange*1 and *longrange*1 are the coordinates that marks the beginning of the area of interest, while *latrange*2 and *longrange*2 marks the end. To 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 $100-199$

```
{
        "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

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 are not fetched. The most valuable properties are *trilong* and *trilat*. As the names suggest these contain the estimated coordinates of each access point.

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 must be translated into relative distances.

The haversine formula [32] 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} as the origin of the coordinate system, and the second set is the coordinates of the AP.

4.3.4 Data output

The resulting python program[4] takes an output filename and two latitudes and two longitudes as input. It queries WiGLE's API for all the APs between these coordinates. As long as the number of APs in the coordinate range does not exceed the daily data limit, all the data will be stored in a topology datastructure imported from the topology generation program in section 4.2. The APs will be placed correctly relative to each other, with real world distance between them, based on the coordinates of each AP.

[4]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. If an optimal channel distribution is to be computed within the group using e.g. DSATUR [5] which is an NP-hard problem, 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 signals from other adjacent apartment buildings. 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. A simple flooding mechanism as suggested by a paper from 2004[26] and described in section **??**, could allow routers to discover each other. When computing channel distribution there would be no risk of surpassing the viable amount of nodes to compute the distribution for, as the groups would naturally be limited to the apartment building.

Alas, in the real world this would not be not be possible. The density of Wi-Fi deployment today is too high, and it is not impossible that every access point in e.g. Manhattan can observe each other through a transitive relation. This means that if a method that simply floods the network to create a collaborative groups was to be used, the amount of nodes in the group could end up being millions. Flooding mechanisms would saturate the network with update mechanisms, local list describing which other nodes was in the group would be too large. Coordination between millions of peers is not only infeasible, but also redundant. There is no need for a node on the north

side of Manhattan to cooperate with one located in the south.

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 less impactful nodes to create an approximated version of the ideal example.

Having a picture of a 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. Hence, the geographic distribution of access points is far from uniform. The problem addressed in this chapter is finding a method for access points deployed in a chaotic and unplanned landscape to group together in clusters of access points. This can either be solved with a centralized coordinator or with a peer-to-peer distributed protocol with a distributed clustering algorithm. In this thesis we will be focusing on the distributed clustering algorithm.

5.2 Distributed group creation

In the related work section, we presented a couple of centralized solutions. 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, or at least confined to the boundaries of an enterprise. Here we will discuss the benefits and challenges of a distributed model

5.2.1 Benefits

A distributed approach has the advantage of being scalable and requires less administration and maintenance like central controllers would. While the distributed approach also requires other nodes in the network to run the same technology stack to be useful, this could potentially be easier to achieve. If the technology does not require proprietary software and hardware and is open source, it would be easier for vendors or ISPs to include the technology in their products. It would benefit all customers, without imposing any significant scalability or administration costs for the providers. Another option for a distributed solution is going through 802.11 standardization. This would make sure all 802.11 compliant access points would be a part of the clusters.

5.2.2 Challenges

Of course the distributed approach has some major challenges that has to overcome. Here are the most prominent ones:

1. **The group creation algorithm.** 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 is the distributed clustering problem that will be addressed in the rest of this chapter.

2. **The communication channel**. In a finished implementation of the group creation algorithm from step 1, access points would need to be able to communicate with each other. The 802.11 standard does not specify any 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 for reliable delivery of protocol message, and the communication channel would have to convey messages that enables group creation, most likely a custom application layer protocol.

3. **State synchronization**. As all nodes should be able to compute a channel plan for the group they need to have synchronized information about the members of the groups and every members' signal strength measurements. Else the results of the channel distribution would be different for different nodes, and provide little real world value.

4. **Channel distribution calculation**. 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 less computational power than a centralized controller.

5.3 Clustering assumptions and requirements

We will be using the data created in chapter 4 to simulate group creation in network topologies. The data provides a topology of nodes where all nodes have a list of neighbouring nodes. Just as real life access points can scan for neighbouring networks, this neighbour list contains the appropriate computed signal strength measurements for each node.

5.3.1 Clustering requirements

1. Clusters must be formed based only on the locally available information at each node. This means that nodes have no concept of their position, nor their relative position to their neighbours. All information available to each node is the SSIDs of its neighbours as well as the signal strength of these neighbours.

2. The clustering algorithm has to be dynamic. Meaning that when nodes appear or disappear it can adjust for the changes in the network topology.

3. The clustering algorithm needs to be able to adjust its size to a given maximum size. We do not make any suggestions for what this maximum size should be, but we have used 128 as the maximum size for the rest of the thesis.

4. When clusters are being formed, the border between two different groups should be placed in such a way that the signal strength between the two access points that disturbs each other the most, is as minimal as possible. This is to ensure that the access points that would interfere the most heavily with each other ends up in the same group.

5. Has to work without a proper bi-directional signal strength between nodes. Meaning that a node can observe another node stronger or weaker than the strength itself is observed at.

5.3.2 Assumptions

To simplify the computation procedure and to reduce the workload for building the simulation program we are going to make a number of assumptions:

1. Assumption 1: All nodes involved in the simulation also run the group creation algorithm. There exists no "rogue"-nodes.

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 implies that there exists 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 6 this issue will be discussed.

3. Assumption 3: All the nodes in a group are completely synchronous, and always have an equal image of the state of the group at a given time. This means that through the communication protocol, all nodes in a group are aware of all members' neighbours and their respective signal strengths. Sharing of this information with the group is instant. This is also an issue that will be addressed in chapter 6.

5.4 Program design

5.4.1 Design choices

The simulation program is designed to be modified so it can accommodate different algorithm variations without changing the fundamental framework. Everything regarding group computation is implemented in Python 3 [16], and is designed to parse the output from the data generation program from chapter 4, and use it directly. The data generation (or fetching) is a slow process and should only be done once every time a new data set is required. The decision was made to not parallelize the program to keep results consistent and to more easily debug and locate program and algorithm implementation mistakes.

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 are 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.

5.4.3 Output file structure

The results of the group computations are written to file. The results do not only contain the resulting division 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. 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 to a large extent 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 application 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.

5.5 Cluster algorithm development

In this section we will iteratively develop a distributed clustering algorithm by first assessing agglomerative clustering, then make modifications to gradually fulfill all requirements specified in chapter 5.3.1. All of the code written in this chapter can be found on GitHub [1].

5.5.1 Agglomerative clustering as a starting point

Agglomerative clustering seems like a natural starting point. It does not need a prespecified amount of clusters to be able to run, like for instance partitioning clustering needs. It has a bottom up approach, which means it works well for a distributed clustering scenario which also has to begin at node level.

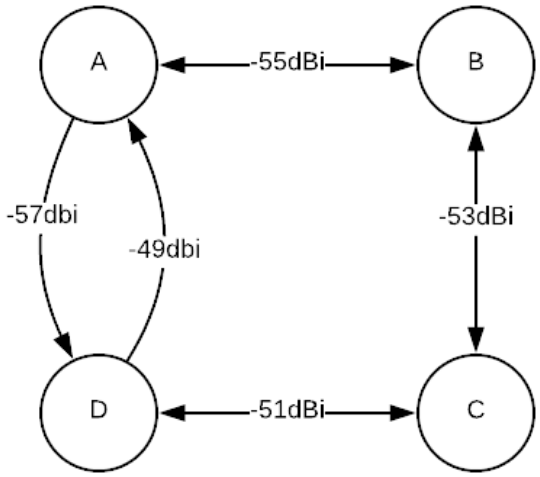
In our test data simulations all pairs of neighbouring nodes observes each other with equal signal strength. That is because we have used the free space path loss formula to derive the signal strength in dBi. The major reason we wont use an unmodified hierarchical agglomerative clustering method, is because in a real-life distributed network topology two nodes might not observe each other with equal signal strength. If that is the case, this could lead to a gridlock situation with agglomerative

[1] https://github.com/hansjny/GroupSimulations/blob/master/GroupCreation.py

```
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.



(a) A wants to merge with B, B with C, C with D and D with A.

Figure 5.2: Gridlock situation with agglomerative clustering

clustering: all nodes are waiting to merge, but nobody can because no pair of nodes observes each other as mutually close. This is illustrated in figure 5.2.

Vanilla agglomerative clustering also does not facilitate an upper cluster size bound, albeit this would be a trivial modification to do.

5.5.2 K-Closest Neighbour Clustering

To resolve the problem discussed in the previous section, we will perform a little modification to agglomerative clustering. To separate it from agglomerative clustering, we will call it K-Closest Neighbour Clustering (KCNC). In this method, similar to agglomerative clustering, in the beginning there are an equal amount of clusters as there are nodes. The difference is that instead of looking for pairs that are mutually close, each cluster will attempt to merge with the cluster that it itself observes as closest. The distance is defined by the same distance metric, observed signal strength. In short: the merge will always happen as long as the resulting cluster does not contain a higher node count than K.

Description

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 its group, and picks the node with the highest observed -dBi value to contact. Of course, 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 5.3.

Merges can not always be accepted, else the group would eventually contain all nodes in 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 are 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 Simulations

Figure 5.4a shows the result of running the K-Closest Neighbour Clustering algorithm on a uniform distribution topology, while figures 5.4b, 5.4c, 5.4d shows the resulting clusters after running it on topologies on cities and towns collected by WiGLE. The different node colors indicate different group memberships. For closer inspection, the large scale images of each topology can be seen in appendix A.1.

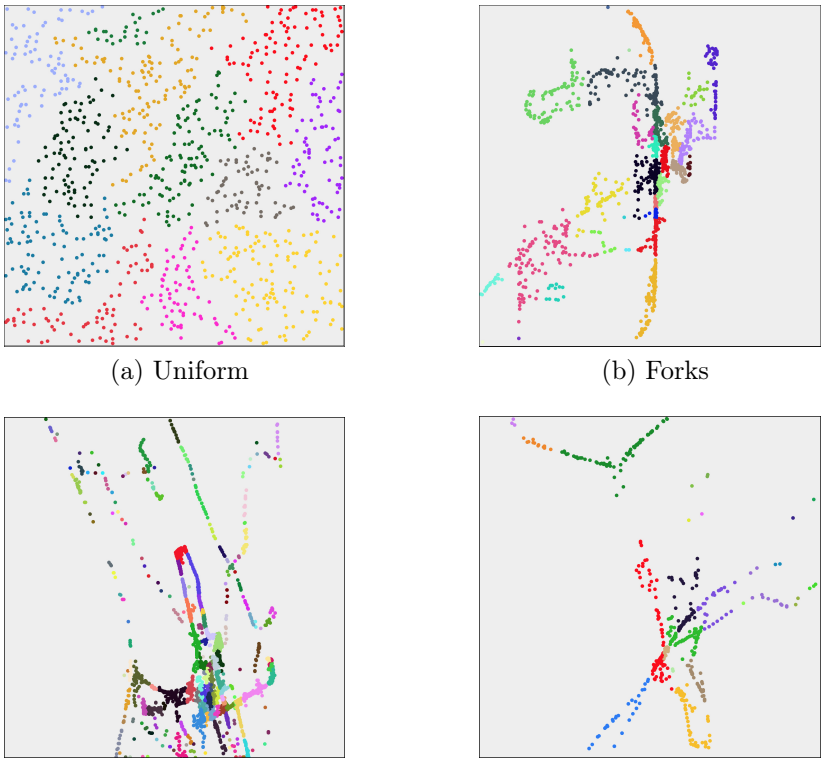
5.5.4 Observations

By looking at the results of the simulations of the K-Closest Neighbour Clustering it is obvious that clusters are created that contains the nearest nodes. Even in the more chaotic topologies as Lillehammer there are distinct clusters that encompasses the most impactful nodes. However, there are two obvious problems with this algorithm which makes it unsuitable:

- In contrast to agglomerative clustering, our K-Closest Neighbour Clustering algorithm does not care if the closest distance between the groups is pairwise

```
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 5.3: Pseudocode sample of how the K-Closest Neighbour Clustering runs in a simulated environment



(a) Uniform

(b) Forks

(c) Lillehammer

(d) Tynset

Figure 5.4: K-Closest Neighbour Clustering on different topologies

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. This violates requirement 4 from 5.3.1.

- Once a cluster reaches the maximum size K, there is no possibility for any other nodes to join the cluster. In the 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 this algorithm alone there is no way to handle this event, and the new node could not become a member of the cluster that surrounds it. This violates requirement 2 from 5.3.1.

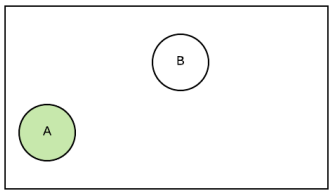
In the next section we will look at how we can use group splitting to offer a solution for these problems.

5.6 K-means splitting

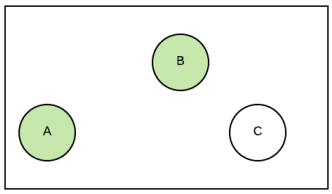
In this section the concept of group splitting will be introduced. Group splitting might improve the K-Closest Neighbour Clustering algorithm suggested in the previous section.

5.6.1 Group splitting hypothesis

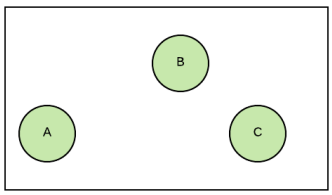
An implication created by letting one cluster merge with another without knowing if the merge is mutually beneficial, is that not all merges will be optimal. A way to increase the likelihood of a group ending up with the neighbouring nodes that impacts each other severely would be to accept merges that 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 5.5, 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.



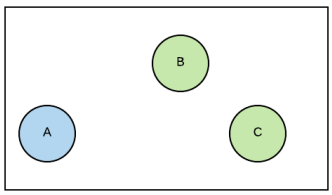
(a) Two neighbouring nodes



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



(c) All nodes joined in transient group



(d) Splitting to reduce group size

Figure 5.5: Simplified illustration of the idea behind splitting. K = 2

5.6.2 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 surrounding, 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.

Nonetheless, K-means could still potentially help create better groups. Let us consider the following. By extending the K-Closest Neighbour Clustering so groups are allowed to merge into a transient group if it exceeds the given maximum size. 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 division 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

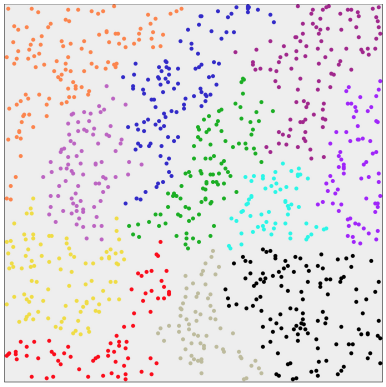
clusters can be pre-calculated before merging, and be used to bootstrap K-means with two non-randomly chosen centroids. If the distance between the most interfering nodes is lower after applying the K-means split, the new groups are applied. On the other hand, if the distance is shorter (meaning a less 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.

5.6.3 Simulations

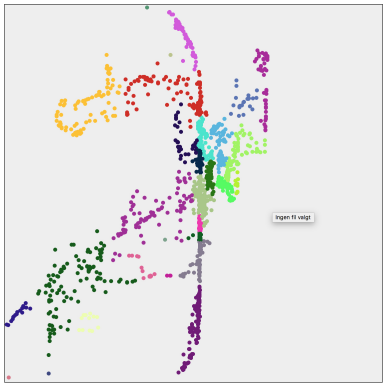
The results of simulating the group creation with K-means splitting implemented can be seen in figure 5.6 (see appendix A.2 for full size images). The groups clearly look very similar to those we computed in 5.4, but in some places there are some major differences. This is where splits have happened. A comparison of the two algorithms executed on the same section of Tynset's town centre can be seen in figure 5.7, and another comparison of a section of the uniform distribution can be seen in figure 5.8.

5.6.4 Observations

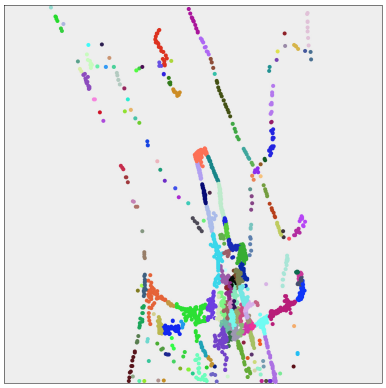
There is little doubt that K-means splitting improves upon the original algorithm, especially when splits are only accepted when the maximum interference between groups is reduced after the split. Even though the clusters in a static environment is slightly improved, the most important use-case for K-means splitting is under the introduction of new nodes to an environment where all groups have converged. Where in the unmodified K-Closest 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 coordinates of the nodes in the group are accessible. This is information unavailable for nodes under real circumstances - the only available information is the approximate 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.



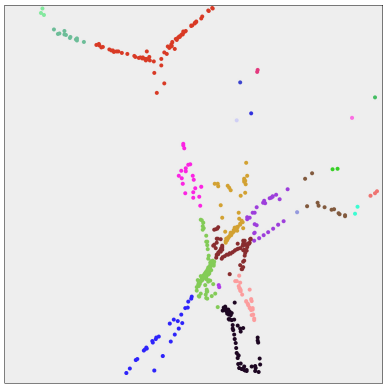
(a) Uniform



(b) Forks

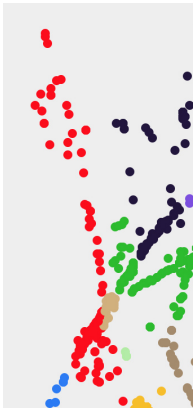


(c) Lillehammer



(d) Tynset

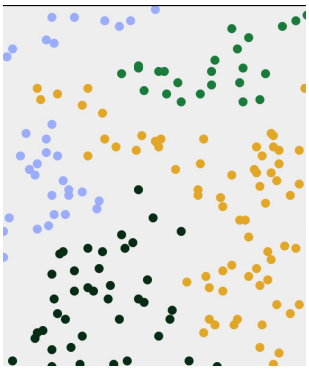
Figure 5.6: K-means splitting on different topologies



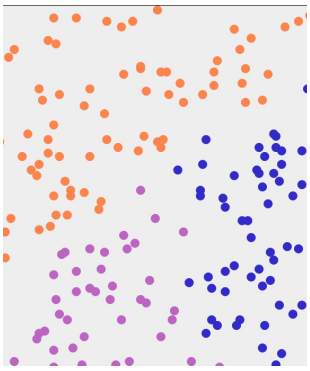
(a) Without K-means splitting

(b) With K-means splitting

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



(a) Without K-means splitting



(b) With K-means splitting

Figure 5.8: Comparison with and without K-means splitting on uniform distribution

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 [15] in a flow network, now called the Ford-Fuelkerson method. The method was extended by the Edmonds-Karp algorithm [14] 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 have 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 of the simulation implementation can be seen in figure 5.9, 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`.

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.

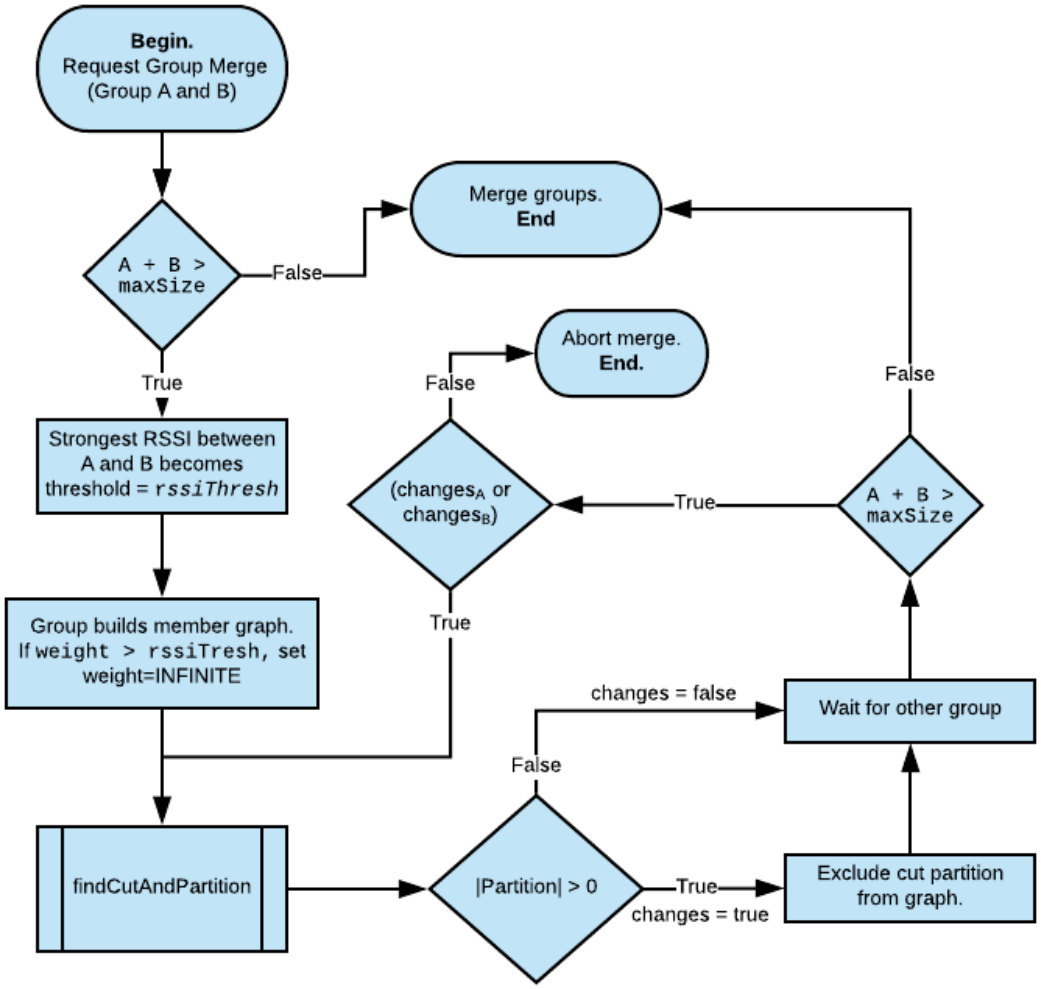


Figure 5.9: Flowchart of the minimum cut implementation for splitting

```
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;
        #Identify which partition is to be separated form the grapzh
        if sink in partition[0]:
            minCutPartition = partition[1]
        elif sink in partition[1]:
            minCutPartition = partition[0]

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

Figure 5.10: Pseudocode of the minimum cut stage of the splitting

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 5.9, can be seen in 5.10.

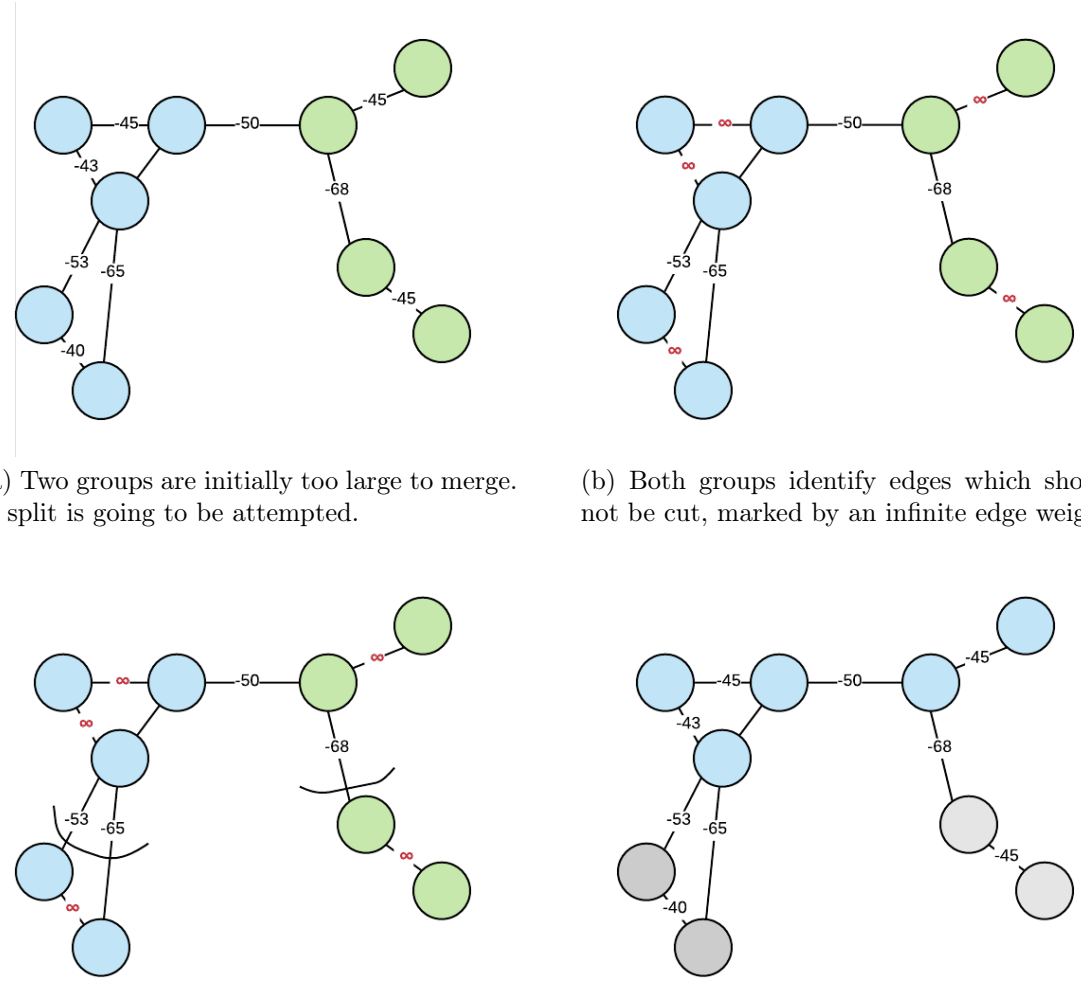
For a detailed step by step illustration describing how a split happens in a topology where a split would be beneficial, see figure 5.11.

5.7.3 Simulation

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 5.12 (full scale images can been seen in A.3). Different groups are distinguished by color, and the group max size is set to 128.

Observations

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 5.13 a close up of the same area has been made between



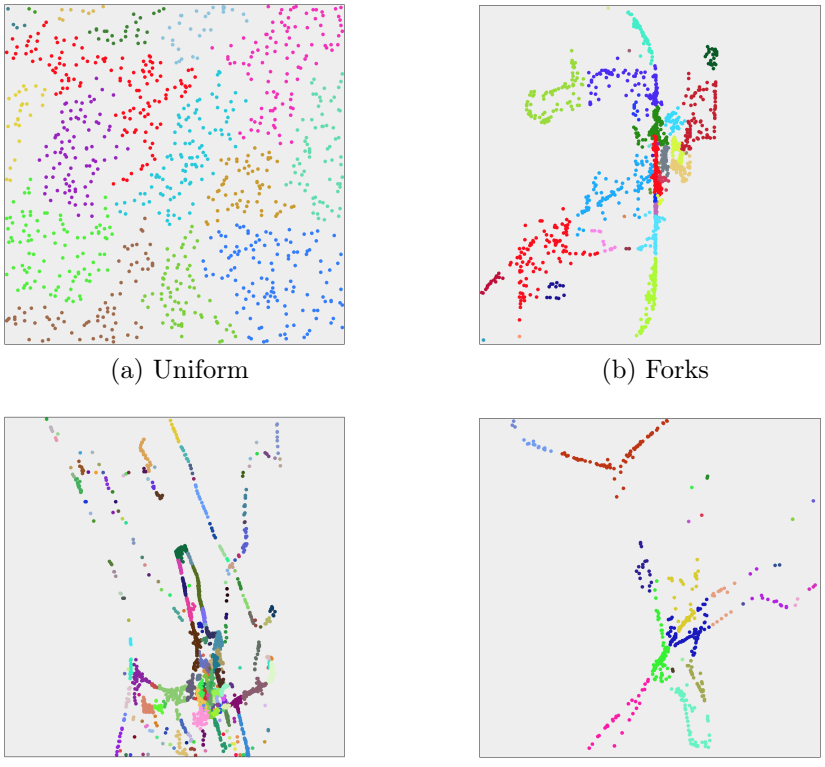
(a) Two groups are initially too large to merge. A split is going to be attempted.

(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 5.11: Minimum cut illustration with group maximum size set to 5



(a) Uniform (b) Forks

(c) Lillehammer (d) Tynset

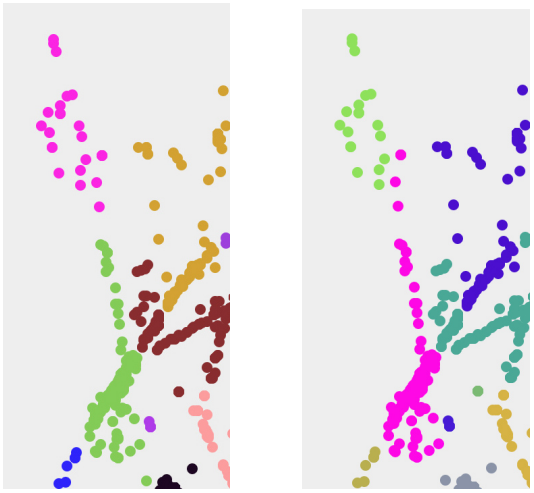
Figure 5.12: Minimum cut algorithm splitting on different topologies

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 5.13a). 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.

By looking at the minimum cut, figure 5.13b, 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 maximum size of 128, so upon a merge it would be reasonable to believe that a split should happen which would redefine the groups in a better way. This is not happening, and we have to inspect the algorithm we suggested to understand why.

Minimum cut version 2

The application we created earlier to step through each iteration of the algorithm can help illuminate the problem. The entire solution converges in 11 iterations, but



(a) With K-means splitting

(b) With minimum cut splitting

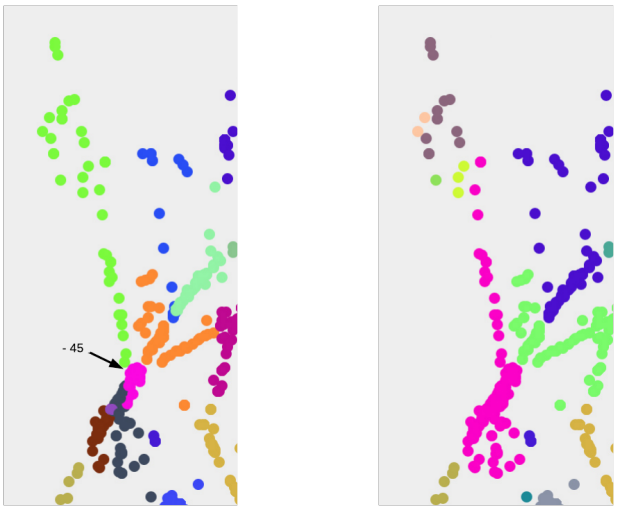
Figure 5.13: Comparison between K-means splitting and minimum cut splitting on Tynset

iteration 8 and 9 seems to be the iterations which specifically impacts the result of the cluster in question. These are illustrated in figure 5.14. In 5.14a, 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 5.14b. 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 could 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. This change would prevent nodes from being left in a group when their closest neighbours have been kicked out, and they would be better off in another group.

Simulations after re-evaluation

The final comparison between K-means and the new minimum cut can be seen in figure 5.14. For the full size version of all four topologies on this simulation as well, see appendix A.4.



(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 5.14: Illustrating two algorithm iterations to find the problem source of the minimum cut

5.7.4 Observations

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 a real-world implementation scenario, minimum cut would only need link weights between nodes to work. Thus while K-means would be hard to realize, the minimum cut method should in theory be little different from simulation to real world implementation, which can be considered a major benefit.

5.8 Assessment

In this chapter three clustering methods has been simulated: K-Closest Neighbour Clustering (KCN), KCN Clustering with K-means splitting and KCN Clustering with minimum cut splitting. This section is dedicated to the evaluation of the results done throughout this chapter, with regards to cluster quality and simulation weaknesses.

5.8.1 Result analysis

Davies-Bouldin Index

The Davies-Bouldin Index [13] is an internal cluster analysis formula. The less scattered a cluster is the lower value is produces, hence a lower Davies-Bouldin Index is desirable.

The formula for the topology wide cluster similarity is as follows:

$$DB_{\text{topo}} = \frac{1}{N} \sum_{i=1}^{N} R_i$$

where N = the number of clusters, and R_i is defined as:

$$R_i = \max R_{\text{ij}}$$

where $i \neq j$ and R_{ij} is defined as:

$$R_{\text{ij}} = \frac{S_i + S_j}{M_{\text{ij}}}$$

S is the average distance between all points in the cluster, and M is the distance between the centroids of the two clusters i and j.

To analyse the clusters we implemented the Davies-Bouldin index. The code for it can be found on GitHub [2]. The result of the Bouldin-Index analysis can be seen in figure 5.15.

5.8.2 Simulation weaknesses and data bias

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 take 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 using computers as access points. Additionally

[2] `https://github.com/hansjny/GroupSimulations/blob/master/Analysis.py`

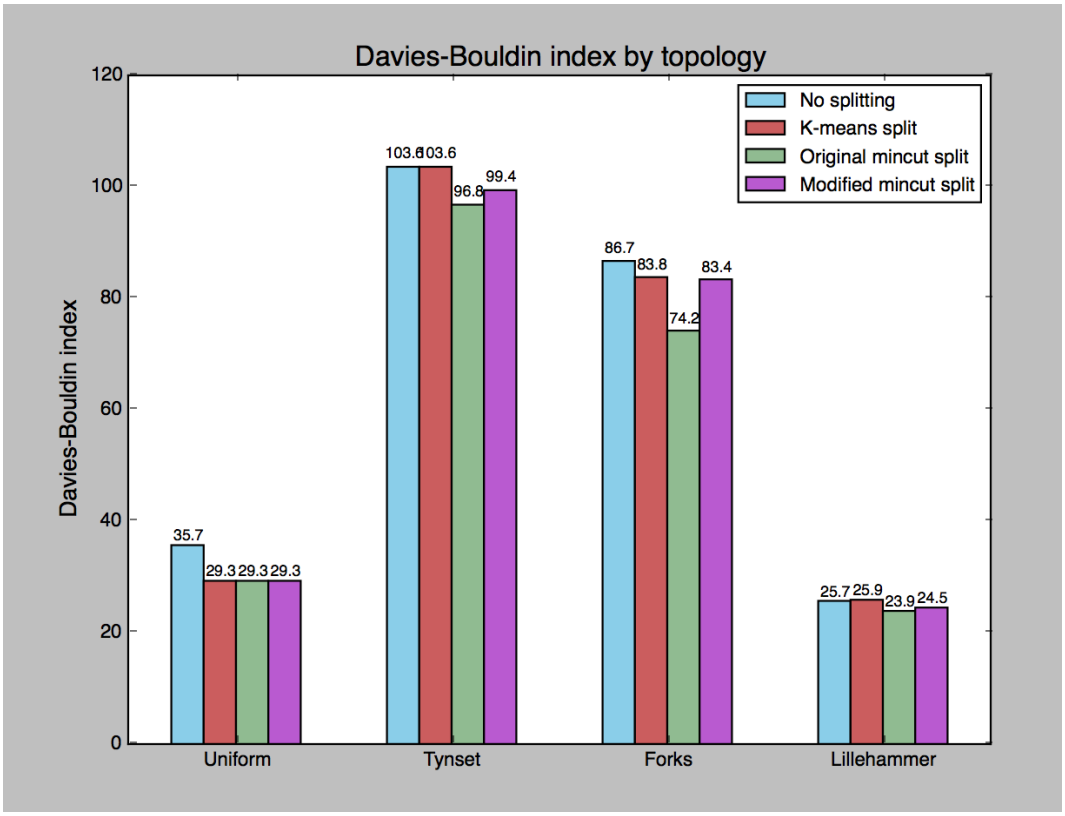


Figure 5.15: Bouldin-Index of all simulations

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 continuous operation. Any reader of this thesis will know that ways the algorithm can handle volatility 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 have been based on the signal strength of all nearby access points, and the dBi values that denotes all signal strengths have 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 agglomerative 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-Closest Neighbour Clustering, even though the topologies used in this thesis actually could work well with agglomerative clustering. We 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 trilateration. 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 in two dimensions should also work well in three dimensions, but there have 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.

5.8.3 Summary and discussion

Before beginning the work on this chapter 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 have also been answered.

The small modification to the agglomerative clustering algorithm which we called KCN Clustering yielded promising cluster topologies already. This algorithm could provide a natural starting point for a real-world implementation in a static network topology. However, this algorithm alone does not handle changes in the network environment very well, as once a group's maximum size is reached, it can not be modified. Hence, we not recommend this to be used for any final implementations, but still it could possibly provide useful for testing scenarios, possibly in an early implementation when a distributed channel allocation algorithm is to be tested across the group.

It deserves mentioning that agglomerative clustering could have been used with the assumption that all signal strength observations was a symmetric relation. This was not done to reduce the amount of assumptions. Not making the assumption does arguably represents a more realistic scenario.

To extend this type of clustering to work in a dynamic environment, 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. i 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. Most importantly, the minimum cut implementation conforms to all requirements postulated in 5.3.1, except requirement 4. By studying the final topologies it is clear that there is still room for further optimization.

The interesting results from the Davies-Bouldin from 5.8.1 analysis is that it shows that the modified version of the minimum cut algorithm performs less well than the original one. But as we saw earlier, the original version of the minimum cut algorithm produces results that are undesirable. Most likely, this means that the results of the modified version of minimum cut produces clusters that are more scattered. So maybe the modification to it was unnecessary, it all depends on how one wants to define the quality of a cluster. Even so, this confirms that the evaluation of clusters can be just as hard as the clustering task itself as indicated by Darius Pfitzner et al. in [30].

Chapter 6

Node communication and group state synchronization

In the previous chapters the focus has been on the algorithm for how clusters can be formed, constrained and updated. In this chapter we suggest some technologies that could enable communication between access points and ultimately help with cluster formation. As we saw in the related work chapter, 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 technology stack that stitches all required components together will be suggested.

6.1 Introduction

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

6.2.1 Distributed consensus with Raft

Raft [29] 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 links to 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.

Why distributed consensus?

We suggest using distributed consensus for the distributed group creation protocol to provide data replication. This makes sure all nodes have a consistent picture of the group, and in-case a group leader goes down any node is equally qualified to take on the role as leader. The need for log replication is derived from the simple fact that there is no centralized controller to take decisions, so equally replicated data across all member nodes is required for all access points to reach the same decisions about which neighbouring group to merge with, and also to compute channel distribution.

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 all the servers in the Raft session. The clients are the actor that requests changes on the data.

A traditional scenario is a 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 a 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.

6.2.2 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.

As mentioned in the related work chapter at the beginning of the thesis, ResFi is a protocol framework that supports creation of radio resource management in legacy residential networks. ResFi is intended to be used in a chaotically deployed landscape of access 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 adjacent and can hear each other over the physical radio. ResFi also supports secure broadcast via n-hop communication. A brief account of the sequential steps of the ResFi standard mode of operation follows below. A more thorough explanation can be found in the ResFi paper [37] 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 exchange 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 [37] 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 uncast 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 returns 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

Implementation

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 in Linux user space by using their modified version of hostapd [33]. 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 router, and a north-bound API to enable applications to use ResFi.

6.3 Architectural 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 represent which of the components that have to communicate with each other. In the next few subsections we address the two blue components individually.

6.3.1 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:

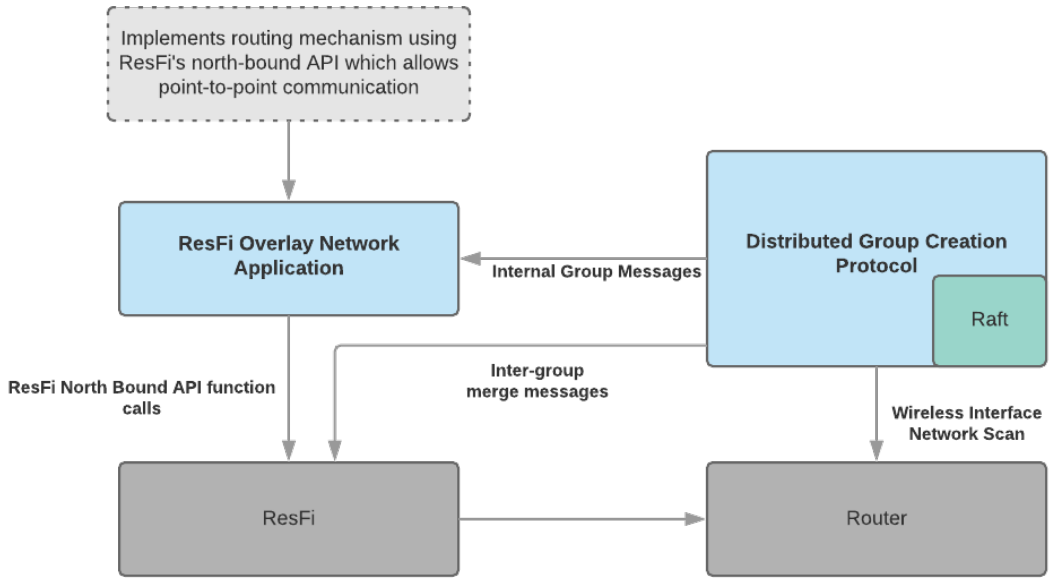


Figure 6.1: Architectural overview of protocol components

- 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.

- Needs a way to uniquely identify nodes to perform routing.

6.3.2 Component overview

This subsection is dedicated to a suggested component overview of a protocol. In order to be able to reference it, we will call it the Distributed Group Creation Protocol (DGCP). A protocol would rely 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 messages through the ResFi Overlay Network Applications to the rest of the group. It also requires 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.

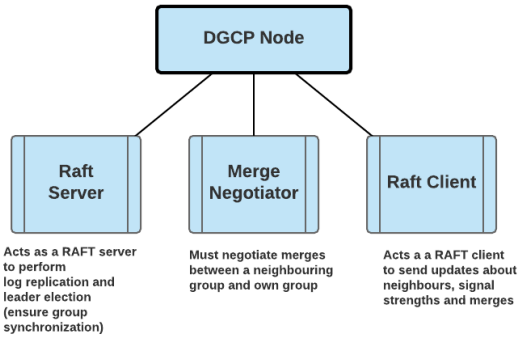


Figure 6.2: The roles of a DGCP Node

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

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 as choosing the neighbours with the highest alphabetical 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 has to individually figure out which neighbour disturbs the most. The node that has the most significant disturber on their direct neighbour list, has to negotiate a merge with that node. We will not address exactly how that message passing would look, but when a merge has been negotiated, and is either accepted, declined or a split is issued, any changes of group membership shouldd be reported to the Raft leader, so the updated group can be replicated to all members.

Internal group messages

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
- Results of merges

6.4 Assessment

This chapter has introduced two pre-existing technologies that can help the process of building and implementing a group creation protocol. ResFi focused on the communication aspects between neighbouring access points, while Raft has been suggested as a way to provide distributed consensus and data replication within the group. We have looked at the architecture that would need to be in place to facilitate this protocol, and suggested some required functionality.

We have not directly addressed what types of messages would have to be passed in the protocol, nor the data structure of the messages. The main reason for not addressing these aspects is that the complexity could quickly become large, and it would exceed the scope of the thesis. It would be more fitting to address these topics when there is an intention of creation a protocol implementation to be able to test a proposed architecture in its entirety.

Hopefully this can be helpful as a starting point for a complete protocol architecture in the future.

Chapter 7

Conclusion

This chapter includes a summary of the work that has been done, a conclusion and a discussion about future work.

7.1 Summary

In this thesis we have suggested a distributed clustering algorithm through iterative development and observation of simulation results. It is intended work with 802.11 wireless access points with the goal of enabling group creation meant for cooperation of channel allocation and possibly other resource management.

At the beginning of thesis we created the data structure to be used for storing network topologies and group simulations. Instead of only generating artificial data, we consulted WiGLE, an online web resource that gathers location information about access points from the entire world, and used their API to download data by specifying a coordinate range. To visualize the data, as well as provide a way to step through each iteration of the clustering stage, we implemented a small web tool in HTML and JavaScript that parses the data. It offers a visual representation of the topology, and an interface to step through it. This web tool has been used throughout the thesis to visualize the groups, separated by color.

We continued by setting some requirements for a distributed clustering algorithm that would need to be fulfilled for the clustering to work in current wireless network topologies. One of the main concerns was that all nodes can only know the signal strength to their own neighbours to begin with, meaning that clustering decisions had to be made from each nodes point of view. Hence the bottom-up agglomerative hierarchical clustering was assessed. Before running any simulations, we modified the agglomerative clustering algorithm to a slightly different algorithm we decided to call K-Closest Neighbour Clustering. This algorithm lacked flexibility in case of

topology changes or introduction of new nodes, and did not have a way to break up existing clusters. To address these issues we proposed an additional functionality to the K-Closest Neighbour Clustering algorithm we called splitting.

Two flavours of splitting has been tested. One using K-means bootstrapped with the centroids from the merging groups to identify a separative line between a large transient cluster. This would result in two disjoint groups from every merge that exceeded the maximum group size. The other using minimum cut theorem from graph-theory to identify partitions of the groups that would be less beneficial to keep than the proposed merge, which could result in several partitions being severed from the group.

Based on the evaluation results, and the nature of the algorithm, we concluded that the K-Closest Neighbour Clustering algorithm with minimum-cut provided the most desirable results, while also being the easiest to implement in real-life.

Finally, we presented some relevant technologies that could facilitate a distributed group creation protocol, and presented an abstracted architecture of the components required for such a protocol.

7.2 Discussion and contribution

In the related works chapter we discussed some centralized solutions that are meant to optimize the performance of 802.11 Wi-Fi networks. But centralized solutions are single points of failures, and certainly very hard to deploy in residential networks where there is little homogeneity with regards to router brands or service providers.

The fundamental issue in residential networks remains largely unsolved: regaining control of the wireless spectrum. To take one step in the direction of addressing this issue, we formulated three problems in the problem statement. First we wanted to define requirements for a distributed clustering algorithm that could run in residential and chaotic networks. This was done in 5.3.1.

The next problem was about developing the clustering algorithm itself. By iterative development we have suggested a relatively simple way to create clusters of access points: using the suggested K-Closest Neighbour Clustering algorithm described in 5.5.2, and respond to changes in the network topology by group splitting using minimum cut described in 5.7. By simulating using the real-world data we gathered in 4.3, and abiding by the requirements for the algorithm we specified, this method should not only work in a simulated environment, but also in a real-world implementation (provided that there exist a protocol to address assumption 2 and 3 from 5.3.2). This leads us to the third problem, which was about identifying technologies that could lead to a real-world implementation of a protocol that implements clustering method. This aspect was covered in chapter 6.

When there are self managing, distributed clusters of access points, algorithms that uses NP-hard heuristics to compute channel distribution can once again be viable to use. This could potentially help resolving many of the congestion problems related to co-channel interference in 802.11.

7.3 Future work

There is a long road ahead to be able to perform the group creation in a real-life wireless topology. This section is dedicated to presenting some possible topics of research that could further the vision of a decentralized, self-managing wireless AP topology.

Protocol development

We briefly introduced an abstract protocol architecture with relevant supporting technologies and data flow in chapter 6. Further development in this area would include a full protocol design, with specified message types and data structures to be involved. A few of the questions that needs to be addressed are:

- How are merges communicated with the rest of the group?
- What happens when a node is powered off?
- How to prevent merge attempts with a group with which a merge has already been attempted?

An implementation of the protocol would be a natural next step, ideally with a testbed to confirm results.

Security assessment

Are the ideas presented here possible to unify in a secure manner? This is a question untouched by this thesis, and could provide another topic for future work. This should arguably be done before or together with the protocol development.

Clustering assessment

The clustering algorithms we developed and tried in this thesis may not be the ultimate answer. We have provided a proof-of-concept for how distributed clustering can be performed using an algorithm that satisfies all the requirements we provided. We have not done a comparative study between other possible methods, and no detailed

evaluation of the quality of the clusters has been performed. There might be different approaches to the distributed clustering problem posed in this thesis. A detailed assessment of the ones suggested here and other alternatives might be the basis for another thesis or further research.

7.4 Final remarks

When I started working with the thesis, I thought there would be time to both create a protocol and possibly begin a large scale network simulation of the protocol. Hence the data structure used for the clustering simulations has been a bit overcomplicated, and in retrospect I could have gotten more done if my simulation framework was simpler. A lot of time has been spent on discovering and resolving bugs in the simulation software, and also verifying that the algorithms have been implemented according to specification. Due to some "off-by-one"-errors a lot of time was spent getting minimum cut splitting to converge.

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] S. Basagni. 'Distributed clustering for ad hoc networks'. In: *Parallel Architectures, Algorithms, and Networks, 1999. (I-SPAN '99) Proceedings. Fourth InternationalSymposium on*. 1999, pp. 310–315. DOI: 10.1109/ISPAN.1999.778957.

[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] Adam Coates and Andrew Y. Ng. 'Learning Feature Representations with K-Means'. In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 561–580. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_30. URL: https://doi.org/10.1007/978-3-642-35289-8_30.

[11] 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).

[12] D. E. Comer et al. 'Computing As a Discipline'. In: *Commun. ACM* 32.1 (Jan. 1989). Ed. by Peter J. Denning, pp. 9–23. ISSN: 0001-0782. DOI: 10.1145/63238.63239. URL: http://doi.acm.org/10.1145/63238.63239.

[13] David L. Davies and Donald W. Bouldin. 'A Cluster Separation Measure'. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 1.2 (Feb. 1979), pp. 224–227. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1979.4766909. URL: http://dx.doi.org/10.1109/TPAMI.1979.4766909.

[14] 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.

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

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

[17] 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.

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

[19] D. Gavalas et al. 'Clustering of Mobile Ad Hoc Networks: An Adaptive Broadcast Period Approach'. In: *2006 IEEE International Conference on Communications*. Vol. 9. June 2006, pp. 4034–4039. DOI: 10.1109/ICC.2006.255712.

[20] 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.

[21] '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.

[22] A. K. Jain, M. N. Murty and P. J. Flynn. 'Data Clustering: A Review'. In: *ACM Comput. Surv.* 31.3 (Sept. 1999), pp. 264–323. ISSN: 0360-0300. DOI: 10.1145/331499.331504. URL: http://doi.acm.org/10.1145/331499.331504.

[23] *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.

[24] S. Lloyd. 'Least squares quantization in PCM'. In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137. ISSN: 0018-9448. DOI: 10.1109/TIT.1982.1056489.

[25] 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.

[26] 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.

[27] 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.

[28] Aerohive Networks. 'Radio Resource Management in HiveOS'. In: (2011). Technical report. URL: https://media.aerohive.com/documents/1753540826_Aerohive-Solution_Brief-Radio_Resource_Management_in_HiveOS.pdf.

[29] 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.

[30] Darius Pfitzner, Richard Leibbrandt and David Powers. 'Characterization and Evaluation of Similarity Measures for Pairs of Clusterings'. In: *Knowl. Inf. Syst.* 19.3 (May 2009), pp. 361–394. ISSN: 0219-1377. DOI: 10.1007/s10115-008-0150-6. URL: http://dx.doi.org/10.1007/s10115-008-0150-6.

[31] 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.

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

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

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

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

[36] *Wireless LAN Controller (WLC) FAQ*. Accessed 21. January, 2018. Oct. 2009. URL: https://www.cisco.com/c/en/us/support/docs/wireless/4400-series-wireless-lan-controllers/69561-wlc-faq.html.

[37] 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

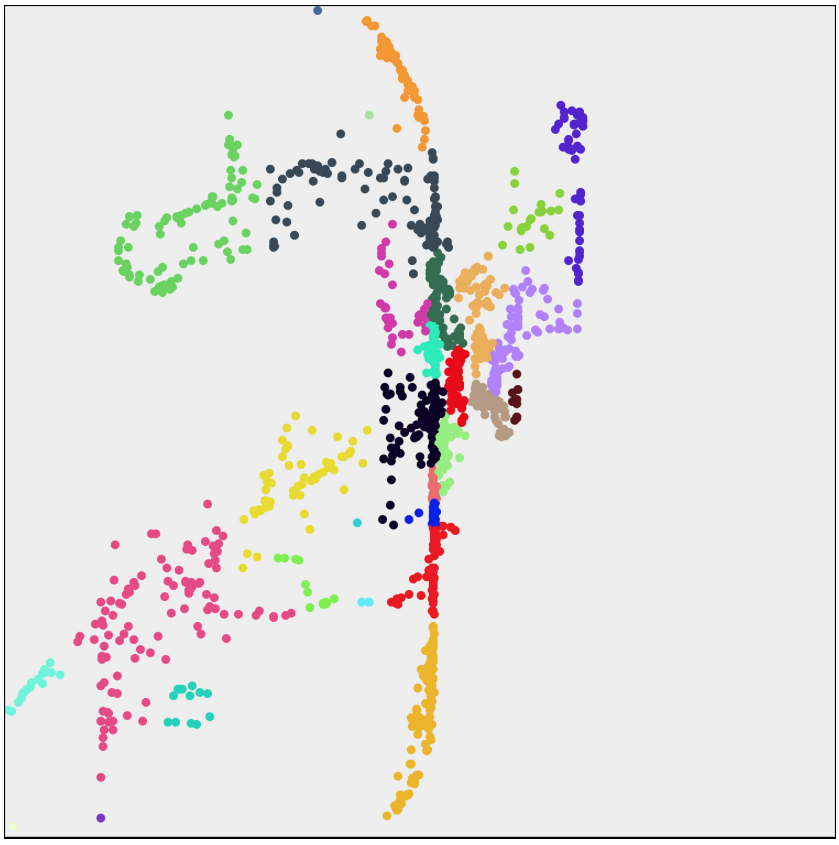


Figure A.2: Forks, Washington, USA

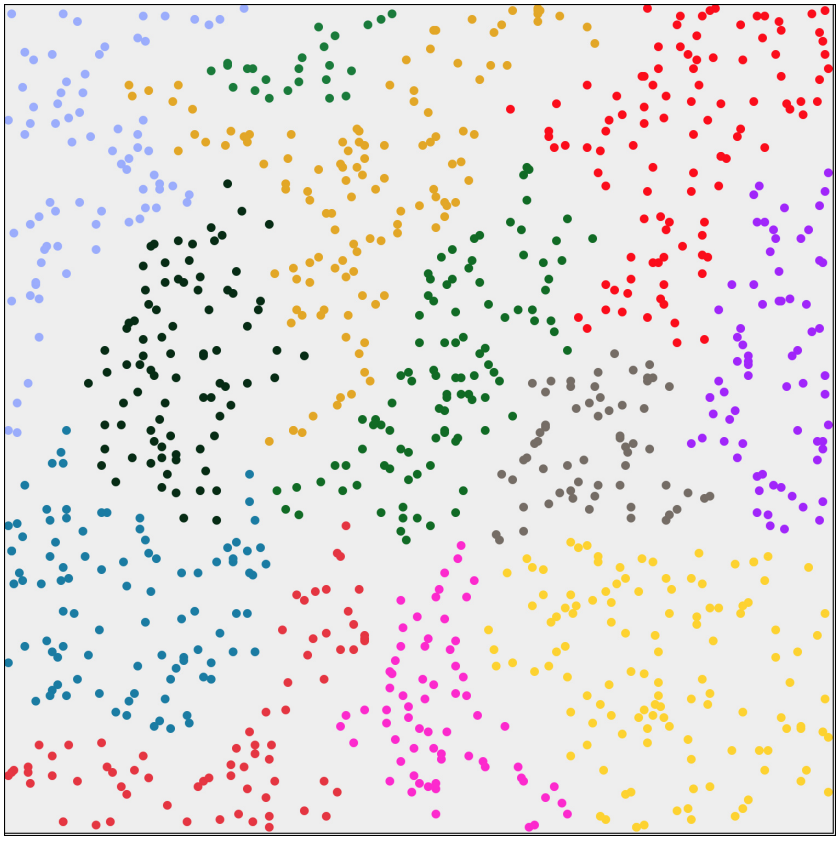


Figure A.1: Uniform distribution, 500x500, 1000 nodes

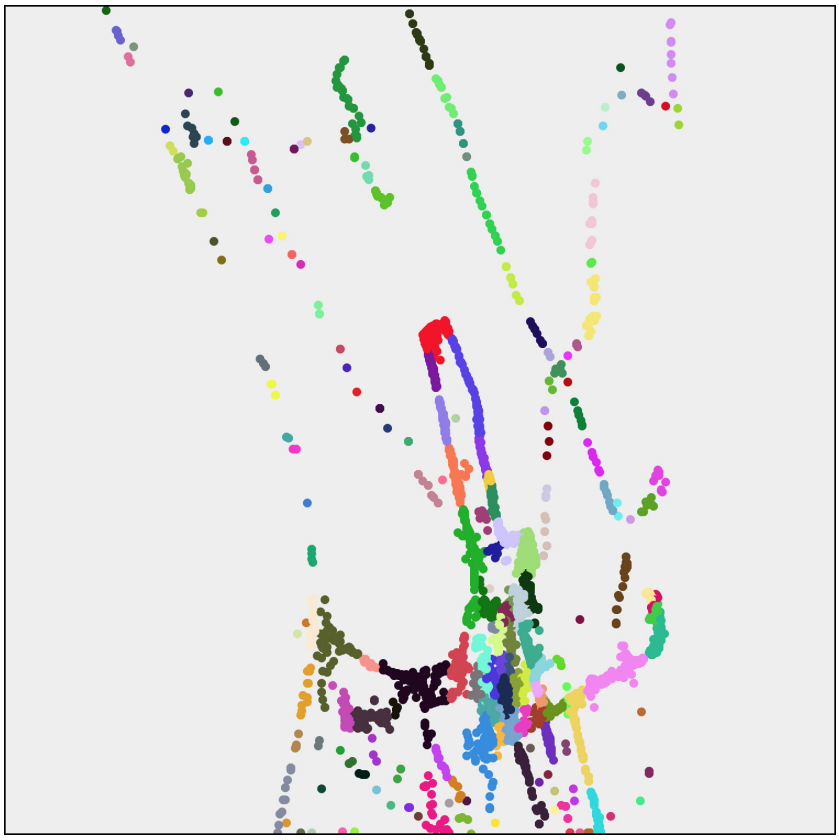


Figure A.3: Lillehammer, Norway

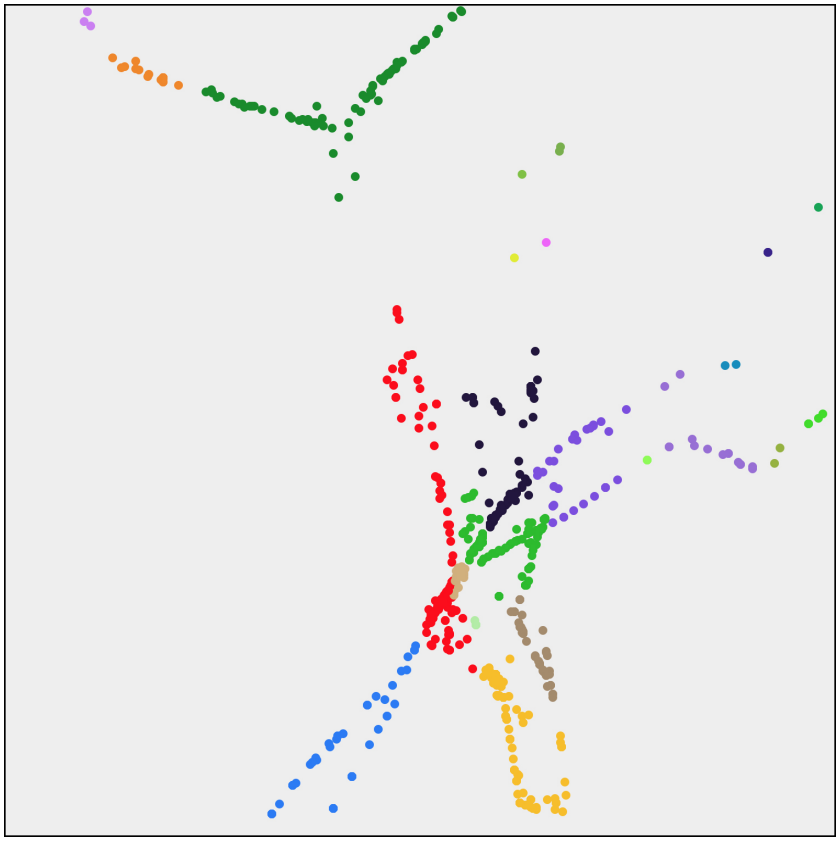


Figure A.4: Tynset, Norway

A.2 K-means Split

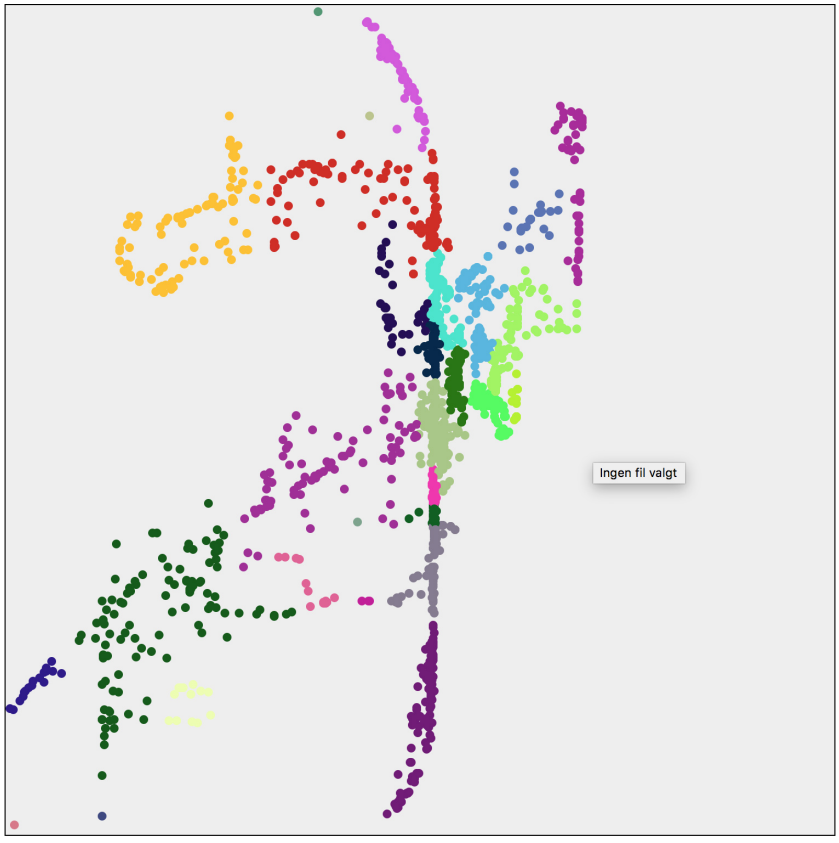


Figure A.6: Forks, Washington, USA

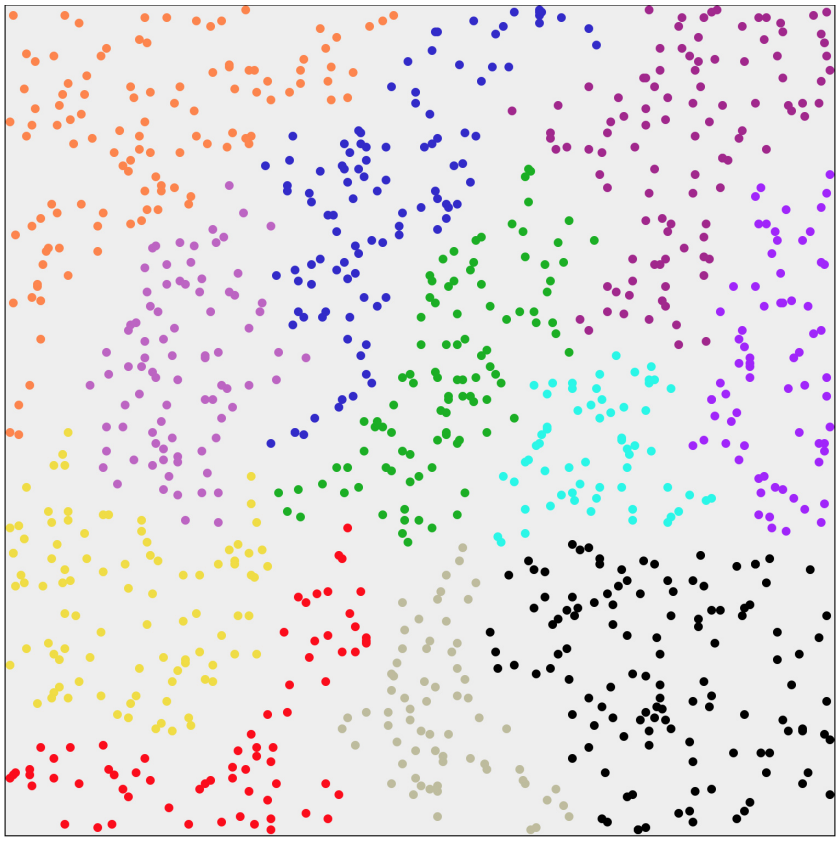


Figure A.5: Uniform distribution, 500x500, 1000 nodes

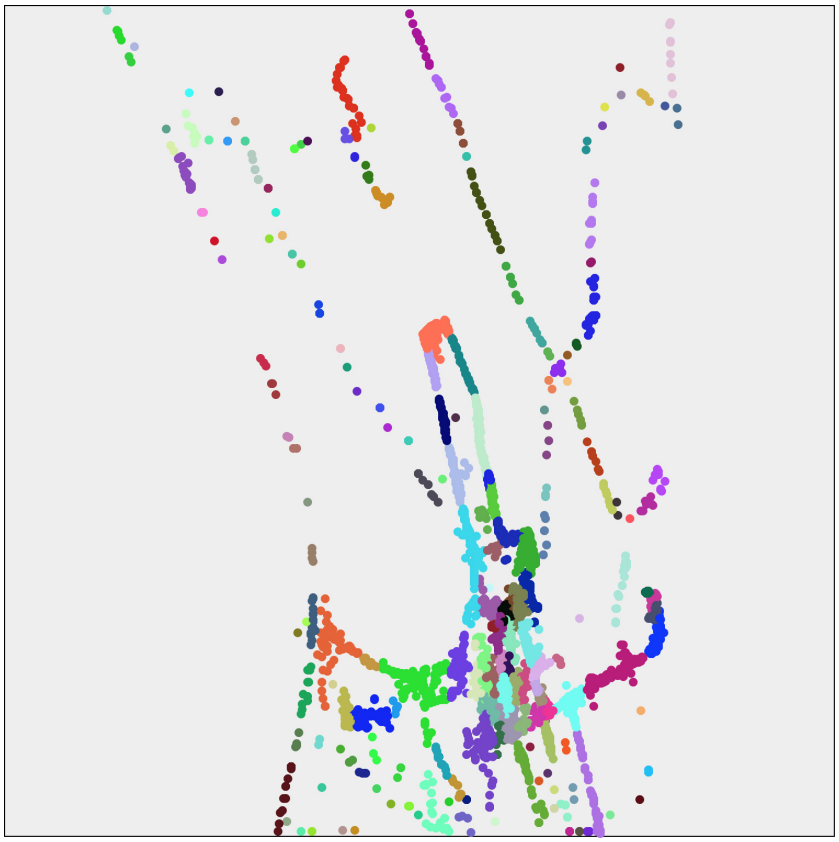


Figure A.7: Lillehammer, Norway

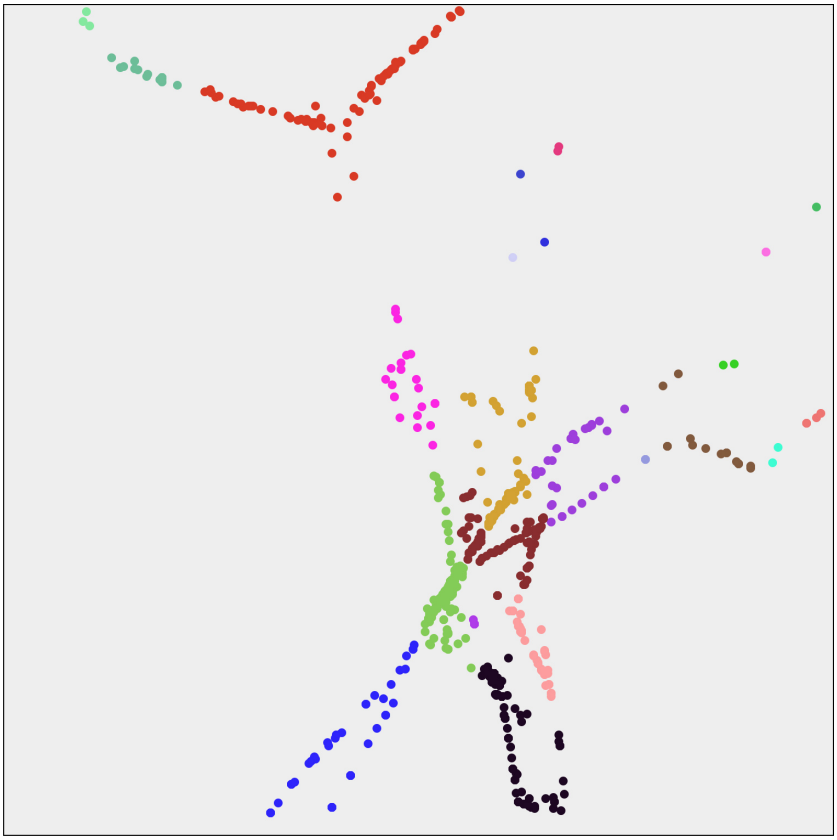


Figure A.8: Tynset, Norway

A.3 Original Minimum Cut Split

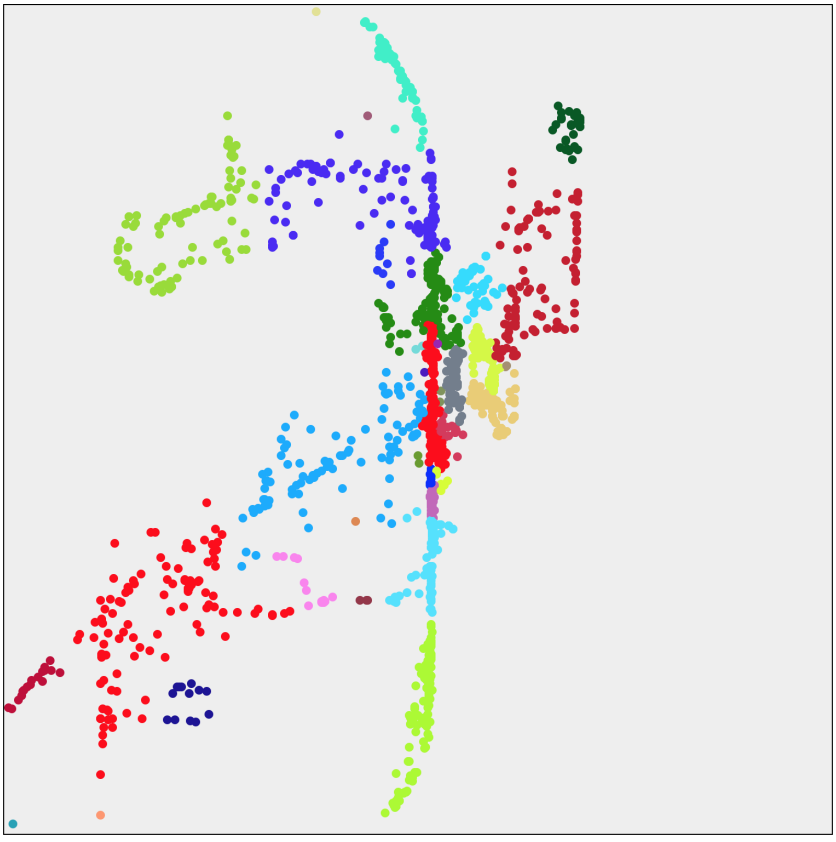


Figure A.10: Forks, Washington, USA

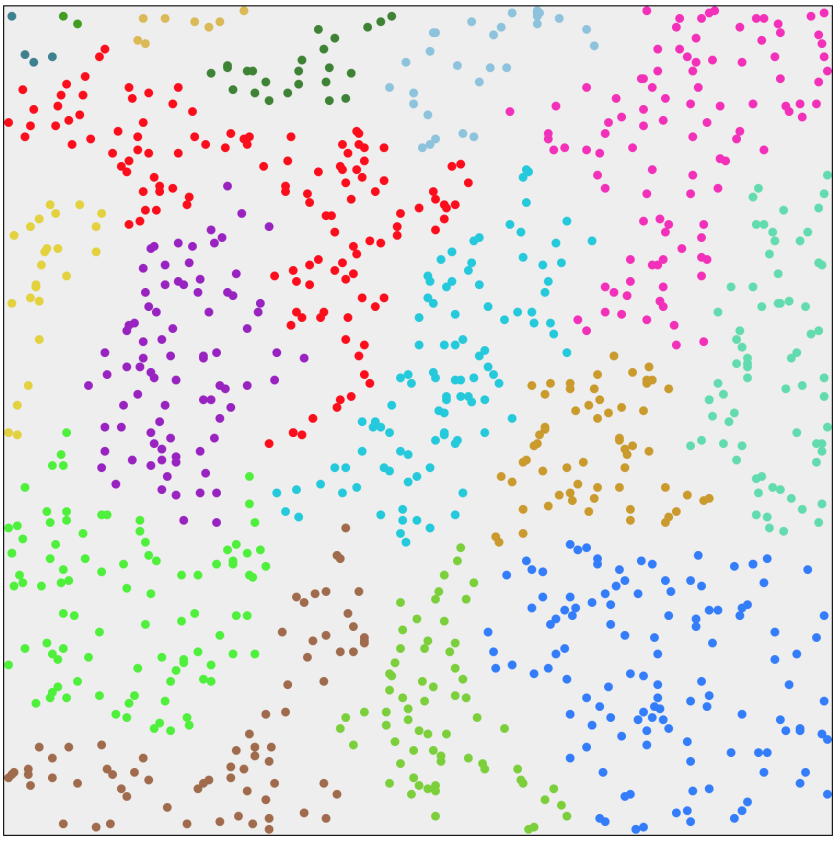


Figure A.9: Uniform distribution, 500x500, 1000 nodes

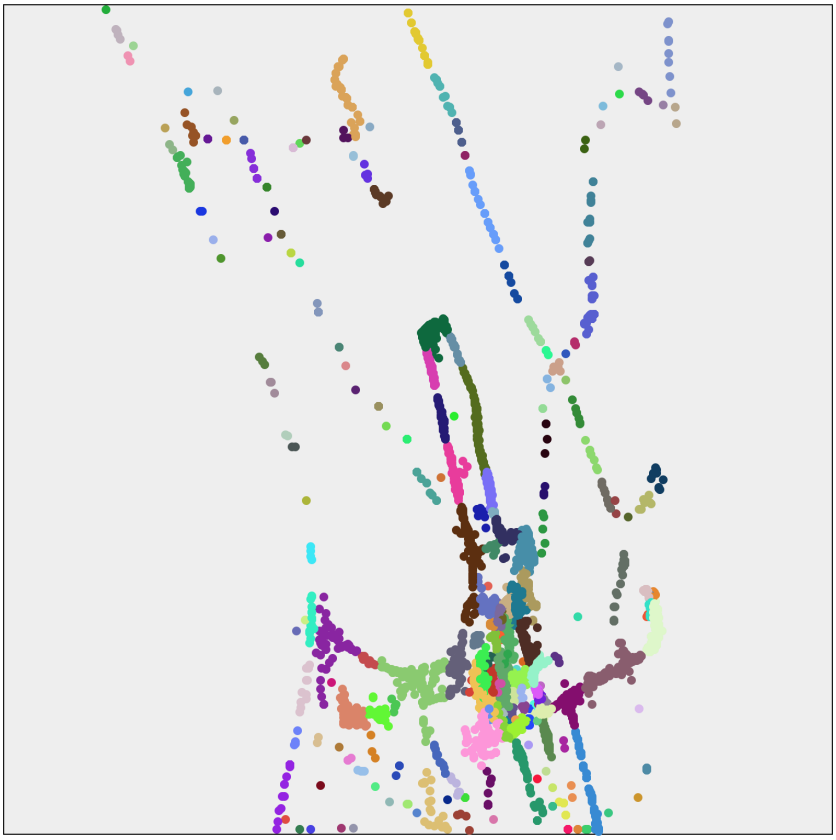


Figure A.11: Lillehammer, Norway

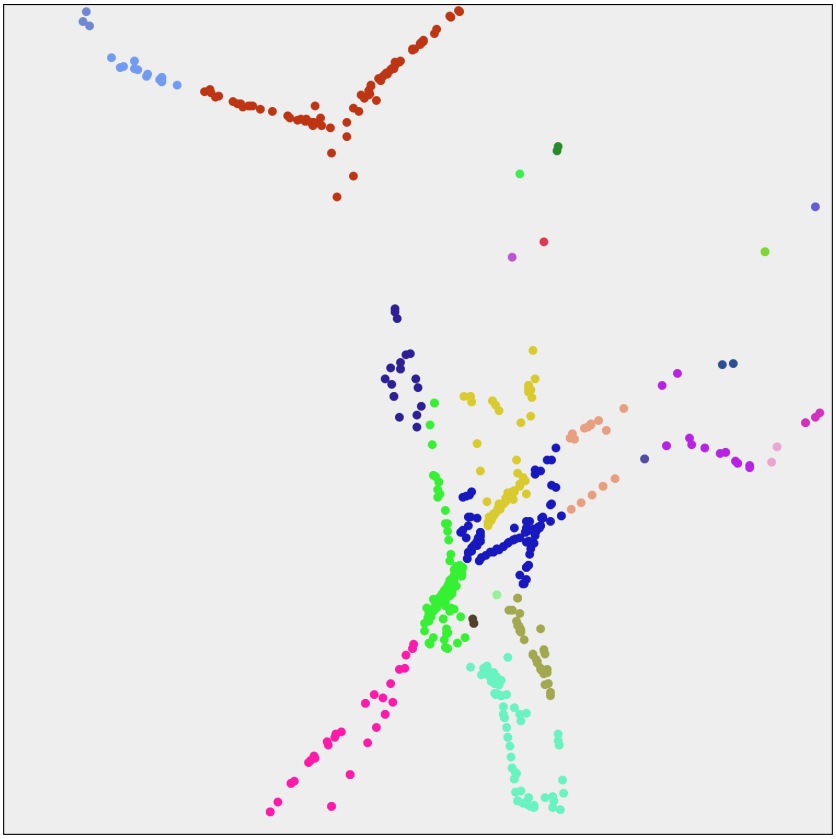


Figure A.12: Tynset, Norway

A.4 Re-evaluated Minimum Cut Split

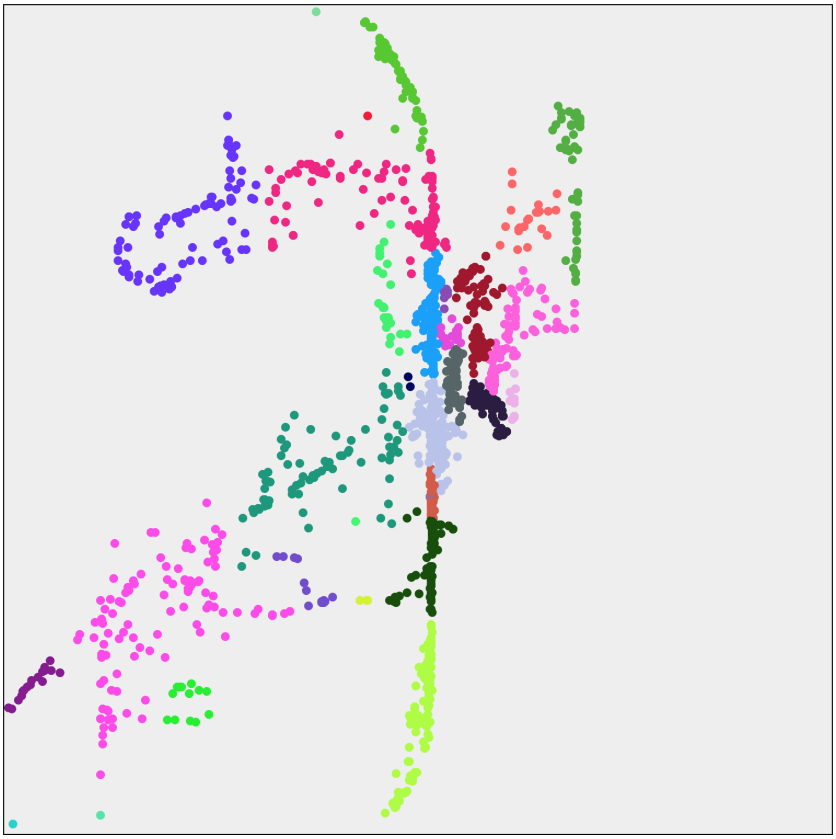


Figure A.14: Forks, Washington, USA

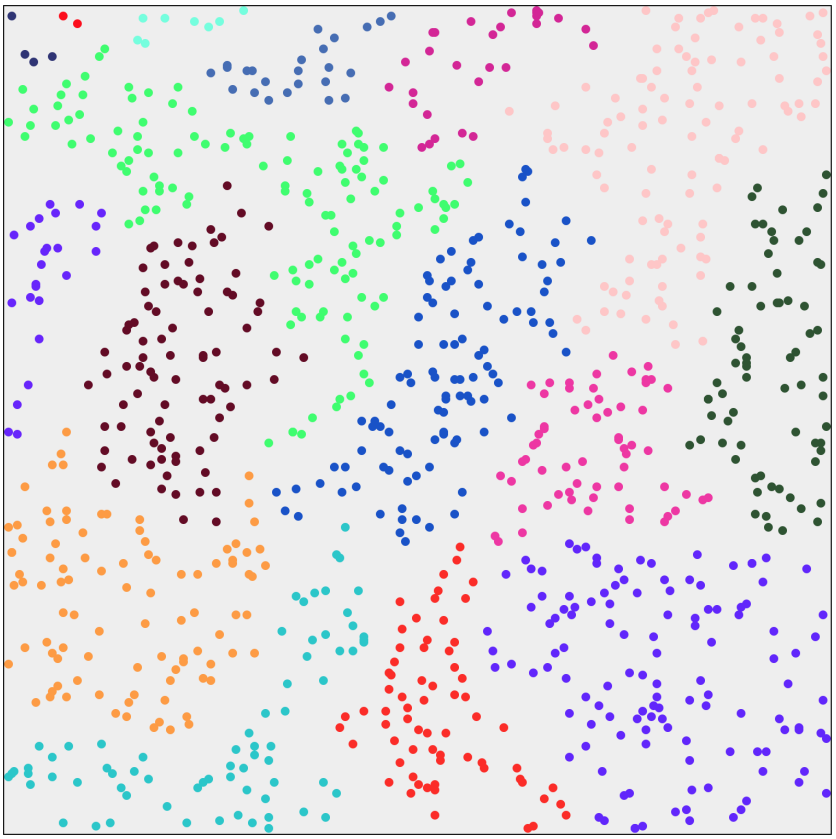


Figure A.13: Uniform distribution, 500x500, 1000 nodes

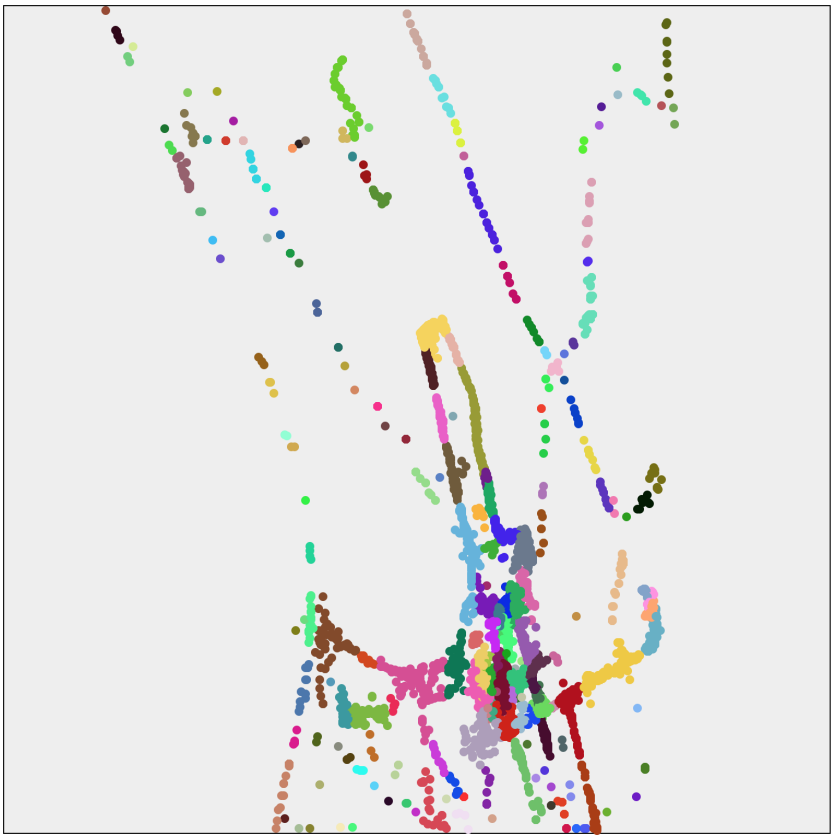


Figure A.15: Lillehammer, Norway

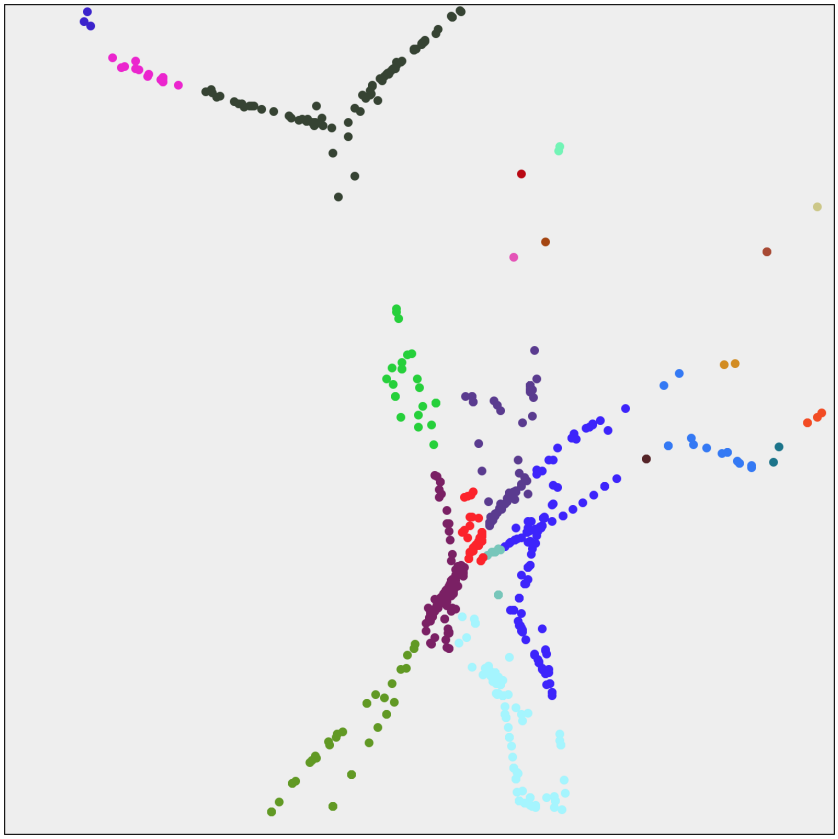


Figure A.16: Tynset, Norway