UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

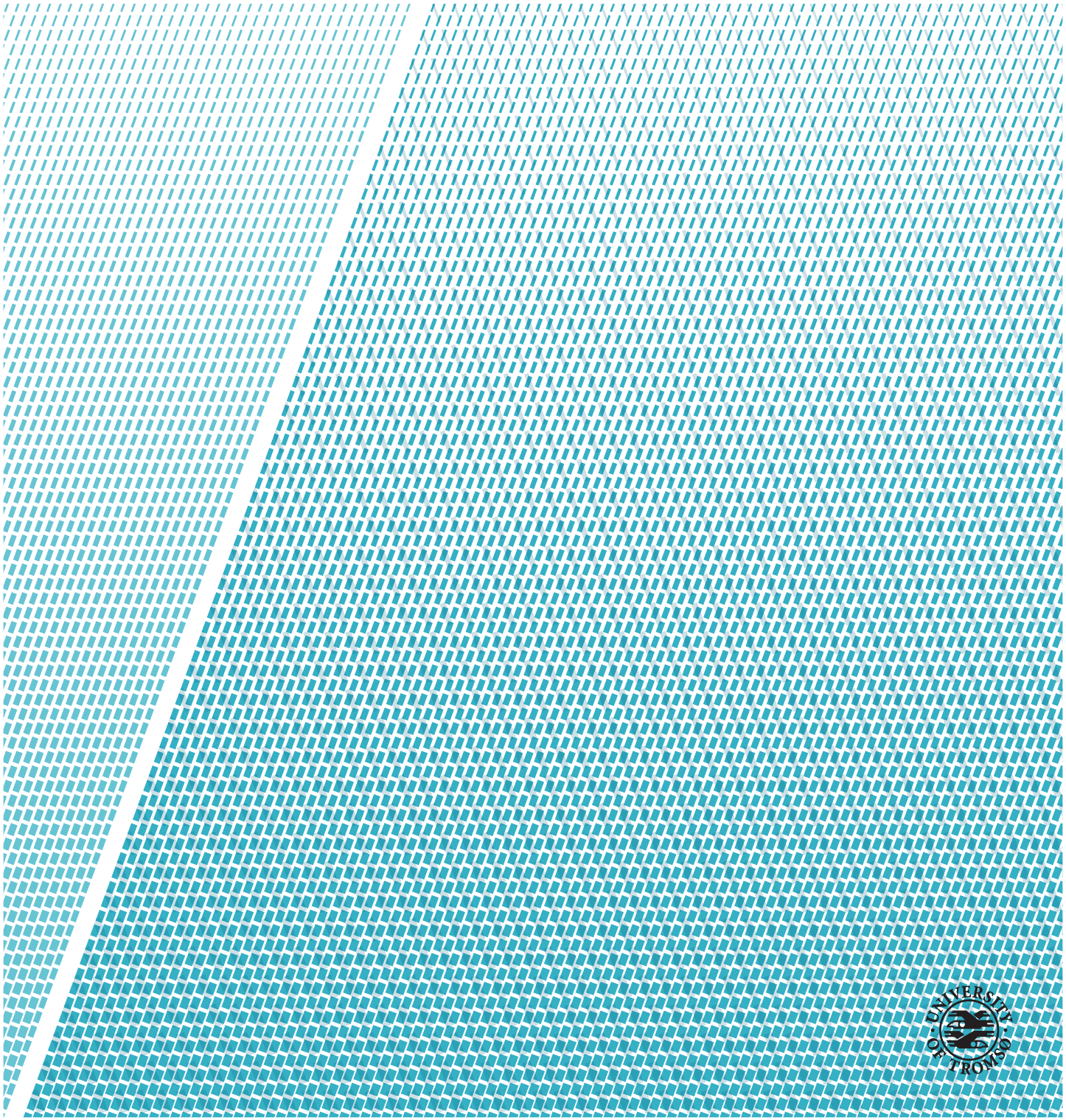
Faculty of Science and Technology
Department of Computer Science

Peer Observations of Observation Units

—

Camilla Stormoen

INF-3981 Master's Thesis in Computer Science
June 1, 2018



Abstract

The Arctic Tundra in the far northern hemisphere is one of ecosystems that are most affected by the climate changes in the world today. Five Fram Center[1] institutions developed a long-term research project called Climate-ecological Observatory for Arctic Tundra (COAT). Their goal is to create robust observation systems which enable documentation and understanding of climate change impacts on the Arctic tundra ecosystems.

This thesis describes a prototype of a Wireless Sensor Network (WSN) system where nodes in the network creates clusters of Observation Units (OUs) to accumulate data. The purpose is to fetch and accumulate data observed by OUs for further use and to provide for a more flexible and powerful sensor in the COAT monitoring of the Arctic Tundra.

We describe a system where nodes discover each other through a range limited broadcast. Together they form clusters. Each cluster elect a Cluster Head (CH) which is responsible for sending out a request for gather and accumulate data from the other nodes in the cluster. The role as CH is rotating among the nodes to conserve battery.

Results show that the system have a steady memory usage between 60% and 76% and CPU usage around 75% during execution. Experiments also show that the CH received fewer packets of data compared to sent packets from OUs in the cluster which indicates that the OUs in the system accumulates data when intended.

The proposed prototype of the system proved capable of electing CHs that gathers and accumulates data efficiently. As a prototype, it still has room for improvements such as the availability of nodes in the system, CH-elections and multiple CHs in each cluster. A future system could further investigate the benefits of having multiple CHs and how to gather and accumulate data more efficiently. There is still a need for conducting further work for a real-life environment in the Arctic Tundra.

1. http://www.framsenteret.no/english

Acknowledgements

First I would like to thank my main advisor Professor Otto Anshus and co-advisor Professor John Markus Bjørndalen for providing guidance, support, ideas and feedback whenever I needed it through this thesis.

I would also like to thank the Department of Computer Science with its technical and administrative staff for support when needed.

I want to express my sincerest gratitude to the *Masterinos*. Thank you for all your help and for five great and fun years, both outside and inside the university. I would not have made it without you guys.

I would also like to thanks my parents for encouraging me to take a higher education and supporting me through every decision. At last I would like to thank my boyfriend.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

BS Base Station

CH Cluster Head

COAT Climate-ecological Observatory for Arctic Tundra

DAO Distributed Arctic Observation

F-LEACH Fuzzy-LEACH

GAF Geographic Adaptive Fidelity

GBCP Gossip-based communication protocol

GEAR Geographic and Energy-Aware Routing

LEACH Low-Energy Adaptive Clustering Hierarchy

OU Observation Unit

P2P Peer-To-Peer

PEGASIS Power-efficient gathering in sensor information systems

TCP Transmission Control Protocol

WSN Wireless Sensor Network

1 Introduction

The Arctic tundra in the far northern hemisphere is challenged by climate changes in the world today and is one of the ecosystems that are most affected by these changes [1]. Five Fram Center[1] institutions developed a long-term research project called Climate-ecological Observatory for Arctic Tundra (COAT). Their goal is to create robust observation systems which enable documentation and understanding of climate change impacts on the Arctic tundra ecosystems. COAT was in autumn 2015 granted substantial funding to establish research infrastructure which allowed them to start up a research infrastructure during 2016-2020 [1].

Wireless Sensor Network (WSN) is a system that consists of hundreds or thousands of low-cost micro-sensor nodes. These nodes monitor and collect physical and environmental conditions. The various activities in the sensor nodes consume lots of energy and the battery of the sensor node is difficult to recharge in wireless scenarios and also because the sensor nodes are located at remote areas in the Arctic tundra.

This thesis presents the architecture, design, implementation and evaluation of a peer observation system that can observe and accumulate data from other peers in multiple clusters in a network.

1. http://www.framsenteret.no/english

1.1 Motivation

The motivation behind this project is that multiple Distributed Arctic Observation (DAO)-Stores [28] may want different data from sensors deployed in the Arctic Tundra. With all the data gathered from each sensor, there will be a need for gathering data that is most important for each DAO-Store. One example can be a DAO-Store COAT that wants data such as images of animals for their research.

This thesis will develop an approach to let Observation Unit (OU) observe each other and gradually accumulate data to OUs being a DAO-Store. There can be multiple DAOs-Stores depending on user needs. The purpose is to fetch and accumulate data observed by OUs for further use and to provide for a more flexible and powerful sensor in the COAT monitoring of the Arctic Tundra.

1.2 Contributions

The thesis makes the following contributions:

- An description of relevant WSNs and routing techniques in WSNs.
- An architecture, design and implementation of a WSN system.
- An evaluation of the system.
- Thoughts on further improvements to the current system and future work.

1.3 Limitations

This thesis focuses on creating a prototype of a cluster network and do not focus on the data passed between the nodes. Data collected from the nodes are not actual data collected from sensors, as this isn't crucial for the system implementation itself.

1.4 Outline

This thesis is structured into 12 chapters including the introduction.

Chapter 2 describes different routing- and communication protocols in WSNs.

Chapter 3 presents related work in the field of WSN, node communication and comparing it to the work done in this thesis.

Chapter 4 describes the technical idea.

Chapter 5 describes the system architecture.

Chapter 6 describes the design of the system and how a cluster network may appear.

Chapter 7 describes the implementation of the system together with choices and issues during implementation.

Chapter 8 describes the experimental setup, metrics used to evaluate the implemented system and the results from the experiments.

Chapter 9 discusses our approach, experiences, how we solved the problem and why we chose the solution we ended up with.

Chapter 10 concludes the thesis.

Chapter 11 suggests future work to improve the system.

Chapter 12 Appendix

2

Routing Techniques in WSNs

Wireless Sensor Network (WSN) is a system that consists of hundreds or thousands of low-cost micro-sensor nodes which monitor and collect physical and environmental conditions. Figure 2.1 shows how a WSN architecture can be, with multi-hop routing.

WSNs main task is to periodically collect information of the interested area and broadcast the information to a Base Station (BS). An easy approach to achieve this task is to make each sensor node transmit their data directly to the BS. But the problem is that the BS can be far away from the sensor node so a direct data transmission would not be possible, or if the routing path from the sensor node to the BS is long, the sensor node may require more energy than available. There are multiple hierarchical protocols that has been proposed as a solution to this problem.

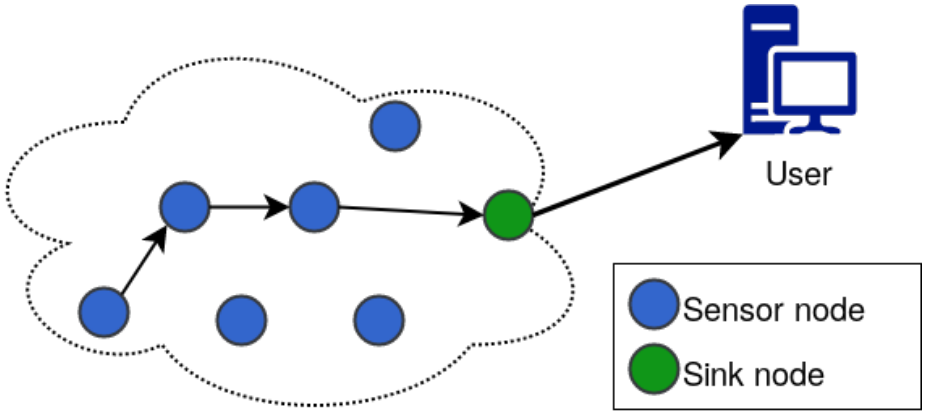


Figure 2.1: Figure shows an example of a multi-hop WSN architecture.

2.1 Routing Protocols in WSNs

2.1.1 Hierarchical Routing

Hierarchical routing, also called cluster-based routing, is a well know technique for network routing with special advantages related to scalability, efficient communication and lower energy consumption in WSNs. Each cluster in the hierarchical routing protocol have a special node which is responsible for coordinating data transmission between all nodes in the cluster [2, 3, 5].

Two approaches based on hierarchical routing are Low-Energy Adaptive Clustering Hierarchy (LEACH)[2] and Power-efficient gathering in sensor information systems (PEGASIS)[6] which are described in Chapter 3.

2.1.2 Location-based Routing

In location-based routing, nodes are addressed based on their location. The distance between a node and its neighbour can be estimated by incoming signal strengths, by GPS or via coordination among nodes [5]. Two approaches are Geographic Adaptive Fidelity (GAF)[18] and Geographic and Energy-Aware Routing (GEAR)[19].

2.2 Flood And Gossiping Protocol

2.2.1 Flooding Protocol

The simplest forwarding rule is to flood the network. In flooding, a node sends a packet received to all its neighbours expept the neighbours which sent the packet until the packet arrives at the destination or maximum number of hops for the packet is reached. Its biggest drawback is that the protocol is not an energy efficient protocol and is not designed for sensor networks [17].

2.2.2 Gossiping Protocol

Gossiping is a modified version of flooding attempting to correct some of its drawbacks. In gossiping, a node with updated data will contact an arbitrary node with updated information. However, it is possible that the contacted node was updated by another node and in that case may not spreading the update any further [20].

3

Related Work

To improve the overall energy dissipaton of Wireless Sensor Networks (WSNs), Low-Energy Adaptive Clustering Hierarchy (LEACH) [2] introduce a hierarchical clustering algorithm for sensor networks. It is self-organized and use randomization to distribute the energy load evenly among the sensors in the network. The sensor nodes organize themselves into local clusters where one node is the local Base Station (BS) or Cluster Head (CH). The CH are not fixed to avoid nodes to drain their battery and to spread the energy usage over multiple nodes. The nodes self-elect a new CH depending on the amount of energy left at the nodes at different time-intervals. LEACH is divided into different rounds where each round include a setup phase and a steady-state phase [4]. In the setup phase will each node decide whether to become a CH or not. When a CH is chosen, each node will select its own CH based on the distance between the node and the CH and join the cluster. In the steady-state phase will the CH fuse the received data from the node members in the cluster and send it to BS. Details of the LEACH algorithm is listed below:

- **Advertisement Phase:** where each node decides whether to become a CH or not. If a node has elected itself as CH, it will broadcast a message to the rest of the nodes. Each node will then decide which cluster it belongs to for this round.

- **Cluster Set-Up Phase:** the nodes will inform the CH that it will be a member of the cluster.

- **Schedule Phase:** based on the number of nodes in the cluster, the CH creates a schedule telling each node when it can transmit data.

- **Data Transmission:** as long as a node has data, it sends data to the CH during their allocated transmission time. After a certain time, the next round begins with each node determine if it should be a CH.

In contrast, will nodes in our approach first connect to a cluster and then start a CH-election rather than elect a CH first and then nodes joining the cluster. A resemblance between the two approaches is that neither of them consider a node's energy level when calculating the CH. Details of our approach is listed below:

- **Cluster Set-up Phase:** nodes will create clusters by connecting to reachable nodes.

- **Advertisement Phase:** when a node joins a cluster, it will start a CH-election to decide whether to become a CH or not. The nodes will gossip the election until the CH is consistent at all nodes.

- **Data Transmission:** the CH broadcasts a message to all nodes in the cluster asking for data. The CH will ask for data a random number of times and when it has received all the data, the next round of a CH-election begins.

LEACH do not consider a node's energy level when calculating the CH. It has therefore been a benchmark for improving algorithms such as the centralized clustering algorithm LEACH-C [10] and distributed clustering algorithm such as LEACH-E [12] and LEACH-B [13]. They concentrate on energy consumption reducing a nodes residual energy and more relevant criterions [9].

Fuzzy-LEACH (F-LEACH) [7, 8] have three different fuzzy descriptors such as energy, concentration and centrality used to complement the cluster head selection process. The BS performs the CH-election in each round by computing the chances of a node becoming a CH by calculating the three fuzzy descriptors. F-LEACH also assumes that the BS elects the appropriate CH because it has a complete information about the whole network.

In contrast to F-LEACH, our approach will elect a CH by the nodes in the cluster and not in a BS. The CH-election will not consider sub-factors such as battery level or number of nodes in the cluster.

Geographic Adaptive Fidelity (GAF) [18] propose a energy-aware location-based routing algorithm designed for mobile ad hoc networks, but are also

applicable for sensor networks. First, the network area is divided into different zones and forms virtual grids. Inside each zone, the nodes communicate with each other to figure out their different roles. E.g., one node will be elected to stay awake for a certain period of time and is responsible for monitoring and reporting data to the BS while the rest of the nodes goes to sleep. Each node is also located by a GPS-position which is associated with a point in the virtual grid. Nodes within the same point are considered equivalent in terms of packet routing. This means that some nodes within the same point can sleep in order to save energy. This apporach is also an example of adaptive fidelity [26] where the quality (fidelity) of the answer of the algorithm can be traded against battery lifetime, network bandwidth, or number of active sensors.

Our approach have no adaptive fidelity algorithm for trading battery lifetime, network bandwidth or number of active sensors. However, our approach use simulated GPS-coordinates (x,y) to place nodes within a grid as similar to GAF [18]. Nodes can only contact other nodes within a simulated broadcast width. This is explained further in Section 7. In contrast to GAF, will CH and regular nodes in our approach accumulate received data with its own data.

Power-efficient gathering in sensor information systems (PEGASIS) is a chain-based protocol with the idea to form a chain among the sensor nodes so each node will receive and transmit data to a close neighbour. The sensor nodes will also take turns on being the CH for transmitting data to the BS and therefore distribute the energy load evenly among the sensor nodes. The chain can be organized by the nodes themselves using a greedy algorithm starting from some node, or the BS can compute the chain and broadcast it to all the nodes in the network [6]. The greedy algorithm assumes that all nodes have a global knowledge of the network for constructing the chain. Each node, except the end nodes, performs data aggregation with data received from its neighbours data and transmit that to its other neighbours.

Our approach do not assume that all nodes have a global knowledge of the network and computing the path is done in the same task as electing a CH. Furthermore, our approach is a cluster based protocol and not a greedy based approach used for chain formation. Our network have one CH per cluster instead of only one node who will aggregate data and pass it to the BS.

To increase the robustness of devices and lower power consumptions, ZebraNet [14] provides a low-power wireless system for position tracking of wildlife by using peer-to-peer network techniques. This reduces the researchers effort to manage the sensors and collecting logged data for their research. The radios on their devices also operates on different frequencies and have different bandwidth, range and other characteristics.

A diversity to our approach is that ZebraNet stores multiple copies of the same data across multiple nodes while our approach forwards the data to the next node in the path to CH. They've also deployed their sensors on zebras in a real-life environment. For us, it's not feasible to deploy sensors out in the real-life Arctic Tundra in such an early development.

Gossip-based protocols, or epidemic protocols, are popular protocols due to their ability to reliably pass information among a large set on interconnected nodes. Jelasity et al. [15] provide a Gossip-based communication protocol (GBCP) where each node have peers to gossip with in a large-scale distributed system [11]. These nodes can quickly join and leave the network at any given point of time. The general principle of their framework is that every node (1) maintains a relatively small local membership table that provides a partial view of all nodes and (2) periodically refreshes the table using a gossiping procedure.

The difference between GBCP and our approach is that our approach does not use a gossip protocol to update its table of nodes, but instead relies on the communication with other nodes to know about the election of a new CH, which of the node is the CH and when a node should accumulate data and send it to the CH.

4

Idea

The technical idea behind the system is that it should be a partially centralized unstructured Peer-To-Peer (P2P) system where nodes, also called Observation Units (OUs), connect to nearby nodes and create clusters. Using an unstructured network would be efficient with regards to a large number of nodes that are frequently joining and leaving the network since it's highly low cost in the face of high rates of churn. The technical idea is shown in Figure 4.1.

The node lookup service in Figure 4.1 should provide a functionality for nodes to discover nearby nodes they can connect to. Together, the nodes creates a network of nodes. However, it may occur that the network will be partitioned into multiple clusters due to nodes not reaching each other.

In a partially centralized network, the role of all peers are the same except of some nodes that assume a more important role. These nodes are called Cluster Heads (CHs) or super nodes. These CHs will be important when accumulating data for further use. A node will assume a more important role in the network decided by multiple sub-factors. The CH nodes are the red nodes in Figure 4.1.

All nodes should be able to observe data observed by other nodes and to gradually accumulate data to OUs being a CH or e.g. a Distributed Arctic Observation (DAO)-Store.

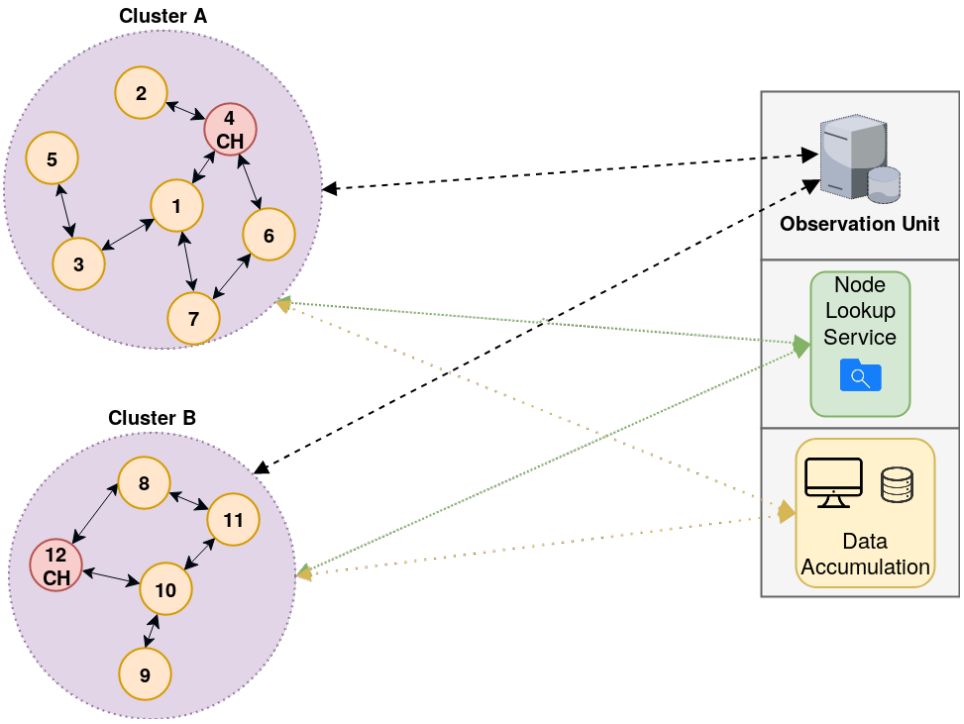


Figure 4.1: Figure shows the technical idea of the system.

5 Architecture

This chapter describes the architecture of the system. The main functionality can be divided into 4 sub-sections: a nodes lookup service, discovery of other nodes, data accumulation and incoming- and outgoing network requests. The architecture of the system is presented in Figure 5.1.

5.1 Node Lookup Service

The lookup service is responsible for storing all previous discovered nodes and their address. This lookup service is responsible for detecting which nodes that are in range for other nodes and return this result to a corresponding node.

5.2 Discovery Of Other Nodes

Each node can only discover other nodes within a simulated radio range. Figure 6.2 shows how a simulated radio range of a node may be discovered. When a new node in the network has been discovered, meta-data about the node such as address, is stored locally on the node in the cluster.

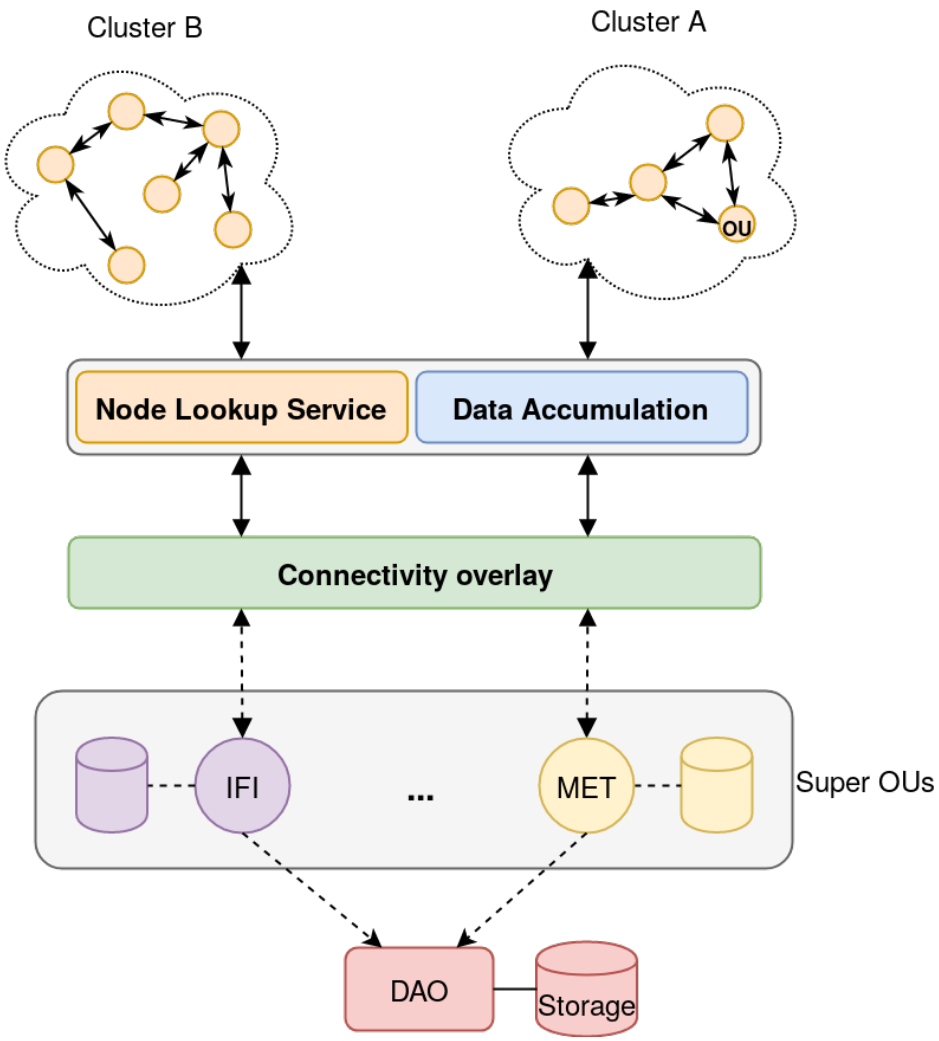


Figure 5.1: Figure shows the architecture of the system.

5.3 Data Accumulation

A node will accumulate data received from another node if its data hasn't been sent to the Cluster Head (CH) earlier. A CH will gather data from nodes in order to provide the accumulated data to a Base Station (BS) for further use.

5.4 Incoming And Outgoing Network Requests

A node may receive incoming requests from other nodes in the network. The request handler will handle the request based on the type of request. The types of requests a node may receive are listed below.

5.4.1 Connect To Neighbours

When a node receive a list of neighbours in range from the lookup service, it will try to connect to the neighbours that are within the nodes range. It will only connect to the neighbour node if it receives a OK-message, i.e. the node is allowed to connect.

When a node receives a OK-message it will connect itself to the neighbour. The neighbour will also then be connected to the new node.

5.4.2 Cluster Head Election Request

When a node receives a CH-election request it will perform a CH-election and forward the result to it's neigbours which will do a CH-election as well. A node may receive a CH-election request when a node has joined the cluster.

Cluster Head Election Calculation Request

If there is a CH in the cluster already and a new election should be proposed, a CH-calculation request is sent from the CH. Nodes receiving this request will calculate their CH-election number.

5.4.3 Data Transmission

Notify Neighbours About Sending Data To Cluster Head

A node may receive a request that it should send its data to the CH. This request is forwarded to the nodes neighbour and so on.

Send Data To Cluster Head

This request forwards a nodes data to the next node in the path to the CH. If the node receiving this requests hasn't sent its data to the CH of the cluster, will the data be accumulated with the received data and then forwarded to the next node.

6

Design

In this chapter we will look at the design of the system and present the design of each component. Figure 6.1 shows how the cluster network may appear in the system. Nodes are connected to other nearby nodes represented by arrows and together they form a cluster network.

6.1 Discovery Of Other Nodes

6.1.1 Broadcasting

When a new node start, it will contact the node lookup service to discover other nodes in the network. The node will then initiate a broadcast. Broadcast is limited due to a radio range limitation where only nodes that are within this range will receive the broadcast, shown in Figure 6.2. *Node 1* will only reach *node 5* and *node 7*.

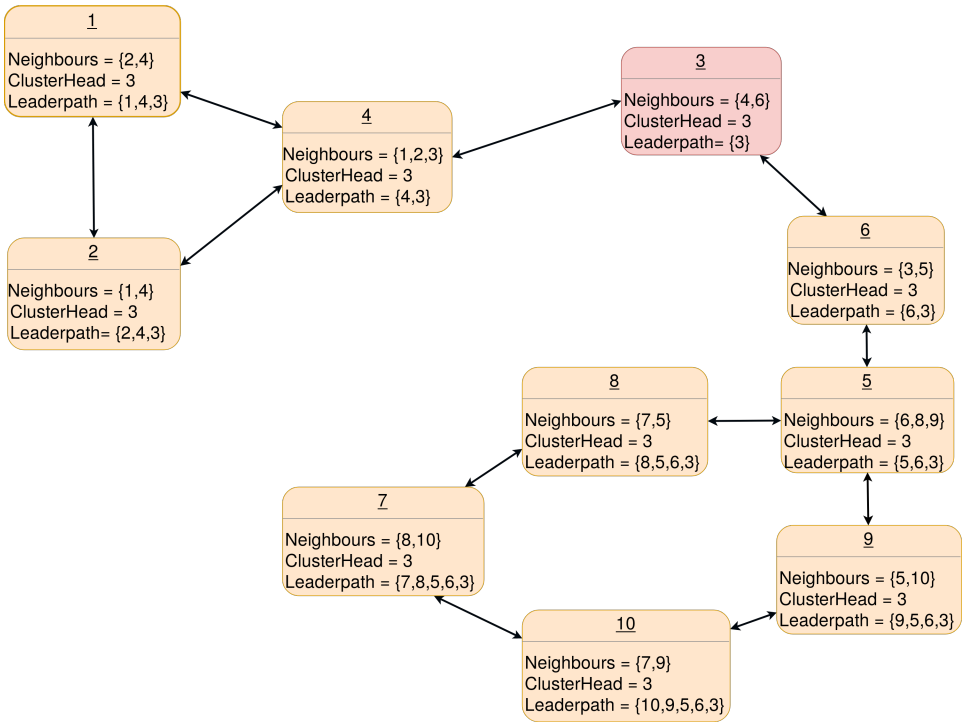


Figure 6.1: Figure show design of the system.

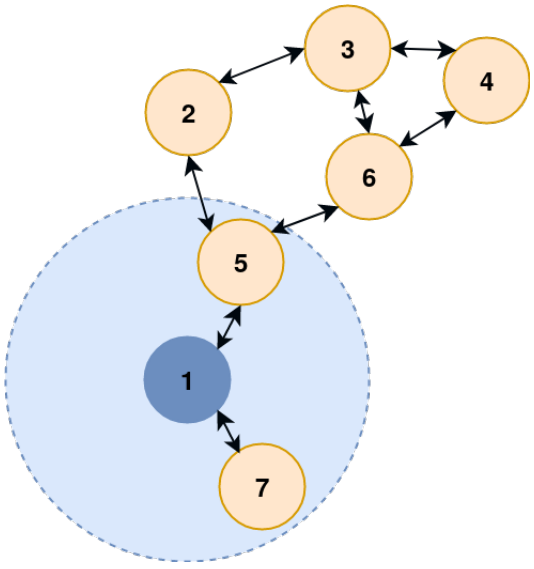


Figure 6.2: Figure show broadcast range of a Observation Unit (OU).

6.2 Cluster Head Election

A Cluster Head (CH)-election may occur in two scenarios listed below.

6.2.1 New Node In Cluster Starts Election

When a new node has joined the cluster, it will start a new CH-election based on the Bully algorithm [23]. It will calculate it's own CH-score and gossip the score to its neighbours. The neighbours will then start a CH-election comparing the received CH-score against their own CH-score. The result of the CH-election will then be gossiped to the node's neighbours with either the received CH-score or the nodes own CH-score. The gossiped message also contains a path to the CH. The node append its own address to the path if the received CH-score won the election, otherwise will the path to CH only contain the node itself. Eventually a new CH is elected by all the nodes and the CH-election result will be consistent in the whole cluster. An example of a CH-election is shown in Figure 6.3.

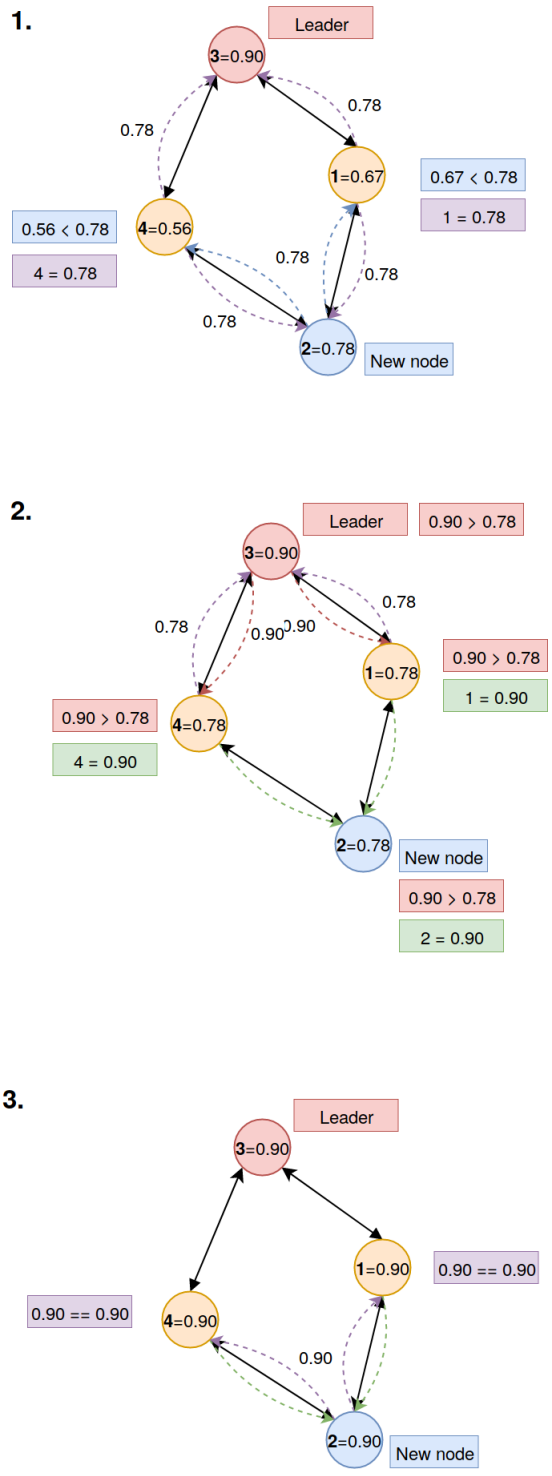


Figure 6.3: Figure show how a new node starts a CH-election.

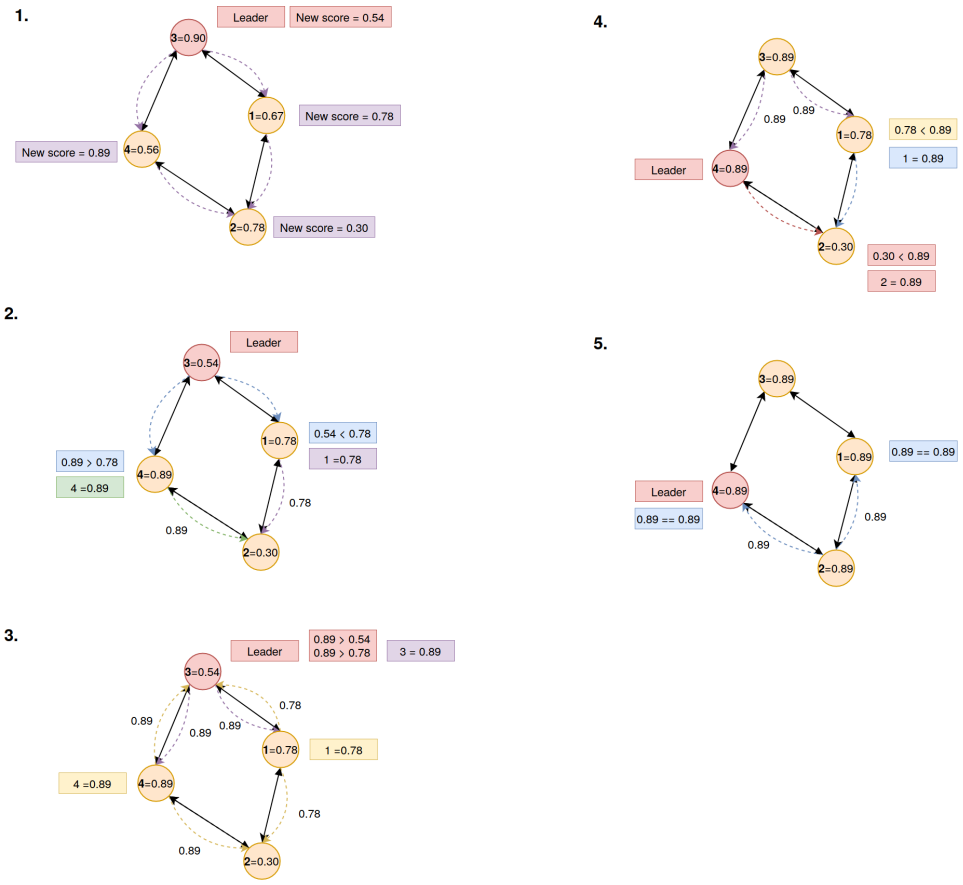


Figure 6.4: Figure show how a CH starts a CH-election.

6.2.2 Cluster Head Starts Election

When a CH has accumulated data 'X' times, it will start a new CH-election. Initially, the CH will gossip a message to the nodes the cluster to calculate a new CH-score. Then, the CH will start a new election and gossip this election to the other nodes. The nodes receiving this gossip message will start their CH-election as described in the section above. The election is illustrated in Figure 6.4.

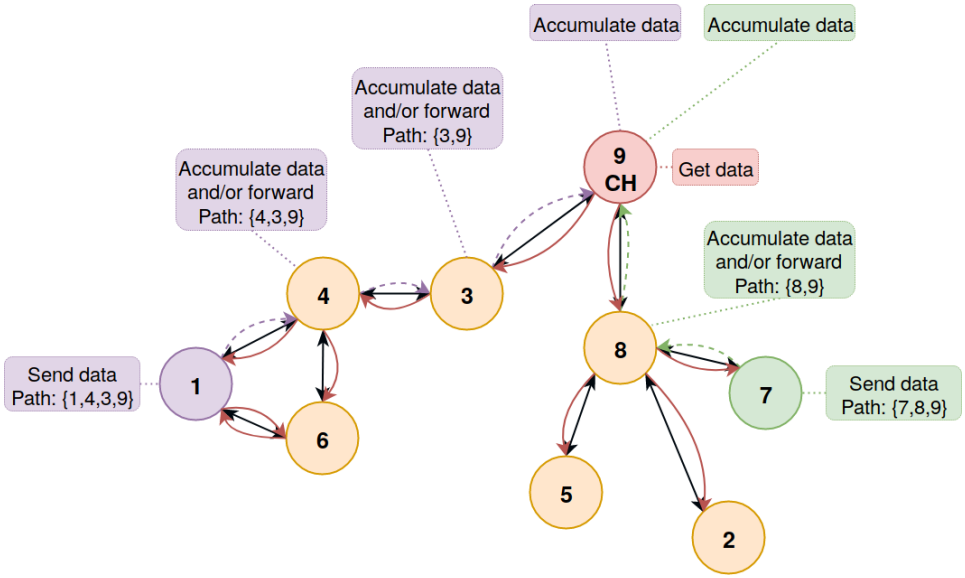


Figure 6.5: Figure show how CH gossips a GET-data request and how nodes sends their data to the CH through the nodes in the path.

6.3 Data Accumulation

A CH will start gathering data by gossiping a message to the nodes in the cluster, illustrated as the red arrows shown in Figure 6.5. When a node receives this message, it will send its data to the next node in the path to the CH, as Figure 6.5 shows. When a node receive data from another node it will check if its own data has been sent to the CH or not by a fingerprint and a status. If it hasn't been sent earlier, the node will accumulate the received data with its own data, and then send the data to the next node in the path to CH. The CH will accumulate all received data and store it.

7

Implementation

This chapter will elaborate implementation of the system, general implementation requirements, issues and choices.

The system is implemented in the open source programming language GO 1.9.3[1]. Each node is launched as an unique process and they communicate with each other by Golangs HTTP Server which listens on the Transmission Control Protocol (TCP) network address. The reason for using Golang is because it's developed for making concurrent mechanisms easy and to utilize multicore and networked machines, and it offers multiple different libraries. When nodes communicate with each other, they send packets structured as JSON [2].

Each node in the network has a limited battery lifetime and during execution the node's battery will be drained causing the node to eventually die. A Cluster Heads (CHs) battery lifetime will be drained faster to simulate that a CH have potentially more tasks than a regular node.

It is not suitable to have nodes deployed in the Arctic Tundra at such an early development. This implementation simulates a real-life environment where nodes can communicate with each other through the TCP network. Each node is assigned *x,y* coordinates within a network grid size and a broadcast width, as shown in 7.1. The purple nodes are new nodes in the network and the purple

1. https://golang.org/
2. https://www.json.org/

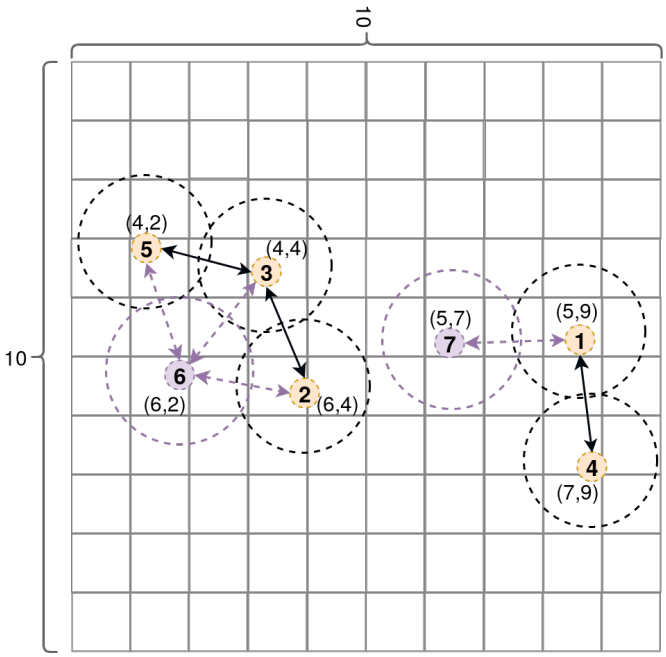


Figure 7.1: Figure show how the network grid size in scale 10 x 10 and nodes broadcast width.

arrows show connection between the new node and reachable nodes.

7.1 Distance To Other Nodes In The Network

A new node will contact the lookup service to discover other nodes in the network by sending a POST-request containing meta-data about itself such as address and position (*x,y* coordinates).

The distance formula 7.1, also called Euclidean distance [24], is used to calculate the range between two points in a two-dimensional coordinate map and is used to see if a node is within a simulated radio range or not, as seen in Figure 6.2. The two points to be calculated is the position of the node itself together with the positions of all the running nodes in the network.

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} \tag{7.1}$$

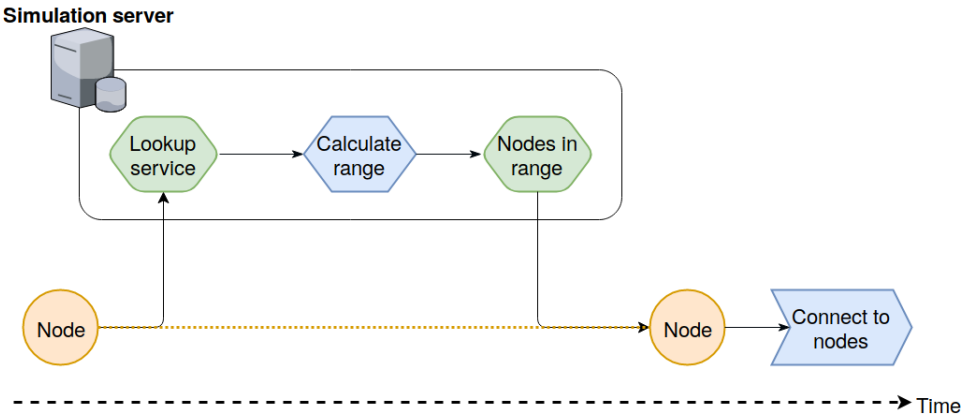


Figure 7.2: Broadcast simulation

Figure 7.2 shows how a node contacts the lookup service. The lookup service computes the node's position. Nodes within the simulated radio range is returned to the node. Finally, the node will try to connect to the these nodes.

7.2 Connect To Neighbours

A node will only be able to connect to another node in the network if the node accept the request to be neighbours. Presently, every node will be able to connect with their neighbours, as long as the node isn't unavailable by means of sleeping or if it's dead. There is no restriction in number of neighbours a node can have or that a node receiving a request from a new neighbour must forward the request to the CH and let the CH decide whether the new node can connect to the neighbour or not. Improvements to this approach is discussed in Section 9.1.1. Figure 7.3 shows the flowchart of when a new node is created and when it will contact reachable neighbours and to start CH-election.

7.3 Cluster Head Election

A CH-election is proposed either when a new node joins the network or by a CH. The CH-election algorithm is based on the Bully algorithm [23] where typically the node with the highest ID number is selected to be the CH. Our approach do not use the node with highest ID number, but elect the node with highest score by choosing a random value between 0 and 1 to be a CH. Each node makes its own decision about whether to become a CH or not

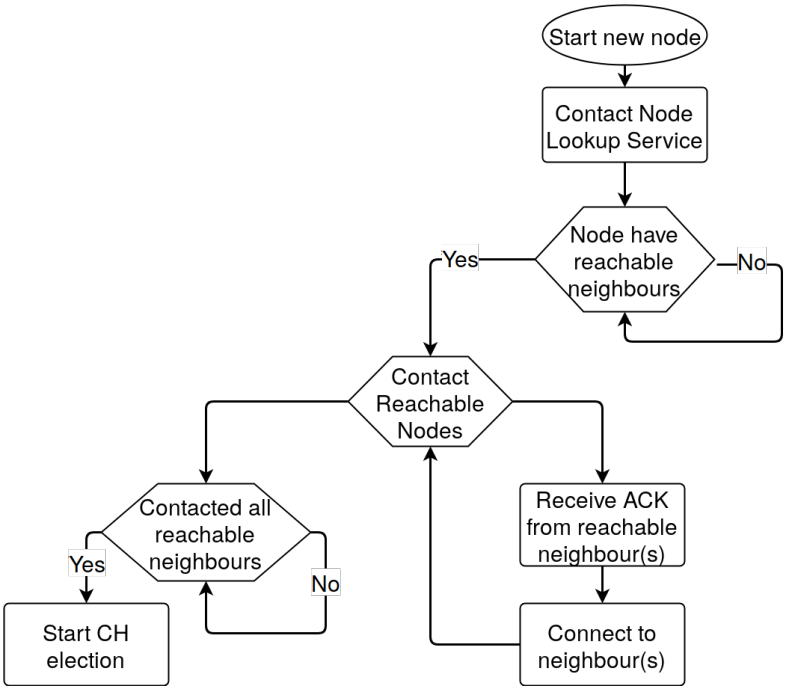


Figure 7.3: Figure show flowchart of when a new node is created and eventually starts a CH-election.

depending on the received score from neighbour nodes. Our approach have several improvements discussed further in Section 9.2.1.

To avoid flooding the network, each CH-election round have an ID. If a node receiving an election have received a request with similar ID and the CH haven't changed, the request will not be forwarded to other nodes considering that the node have forwarded the same request previously.

7.3.1 Cluster Head Calculation

To make the system more realistic in terms of battery lifetime and the capability to become a CH, each node will receive a CH computation request before an election if the CH-election is started by a CH. This approach assumes that all nodes are possible candidates to become CHs. This is to simulate that the nodes characteristics may change during execution and that a node that was highly capable to be a CH before may not be efficient in the next CH-election.

7.4 Minimize Path To Cluster Head

Since CH-elections are gossiped from all nodes, each node can receive multiple gossips per CH-election and some paths will be longer than others. In order to minimize the path to the CH, a node will when receive an election, compare its existing path to the one received [21] and choose the shortest path to be its path to the CH.

7.5 Data Accumulation

7.5.1 Node Data Accumulation

Since all nodes run concurrently and independently, each node can receive multiple requests from different neighbouring nodes. When a CH creates a request for gathering data will the request contain an ID used for identification. Since the request is gossiped to all nodes in the cluster, a node will most likely receive the same message multiple times, but only need to forward its data once. If a node have not received a request with a specific ID, it will gossip the message to its neighbours and then send its data to the next neighbour in the path to CH. Otherwise, the node will just ignore the request.

When a node needs to send data, it will create a request containing the received request-ID from the CH, the data to be sent together with a fingerprint and a path to the CH. The path to the CH is a list containing the addresses for the nodes between the sender and the CH.

The data is a fabricated byte array with a fingerprint that is the hashed value of the data. The fingerprint will make it easier to identify the numerous data on each node. The data will also have a boolean tag which is used to check whether the data has been accumulated or not.

7.5.2 CH Data Accumulation

When a CH starts a gathering of data, the node will chose a value between one and six which indicates how many times it should gather data before it eventual sends out a new CH-election. We chose to implement the CHs to accumulate data between one and six times because finding the exact number of how many times a CH should accumulate data wasn't our primary focus.

As mentioned previously, each node can receive multiple requests from different neighbour nodes. This especially occur in the CH when collecting data from the

nodes in the cluster. The collected data is stored in a map and maps in Go[3] are not safe for concurrent use. If a map is read from and written to from concurrent goroutines, the access must be synchronized. One of the most common ways to protect maps is by using mutexes, illustrated in Listing 7.1.

Listing 7.1: Small Go program showing how mutexes are used when updating a map by CH.

```
1
2  /*SensorData is data from "sensors" on the OU */
3  type SensorData struct {
4      ID          uint32
5      Fingerprint uint32
6      Data        []byte
7  }
8
9  /* DBStation is a strucure that contains
10    a map that store data at CH */
11 var DBStation struct {
12     sync.Mutex
13     BSdatamap map[uint32][]byte
14 }
15
16 func sendDataToLeaderHandler() {
17     var rd SensorData
18
19     DBStation.Lock()
20     defer DBStation.Unlock()
21
22     (...)
23
24     err := json.Unmarshal(body, &rd);
25     if err != nil {
26         panic(err)
27     }
28
29     (...)
30
31     /*Update map in key FingerPrint
32       with received data */
33     DBStation.BSdatamap[rd.Fingerprint] = rd.Data
34 }
```

3. https://golang.org/

7.6 Concurrent CH-Election And Data Accumulation

To avoid having a CH-election occur concurrently with a data accumulation, the following functionalities are scheduled to execute. Firstly, the CH-election will execute until the election result is consistent between all nodes.Second, the CH broadcast a request for gathering data and nodes will accumulate and send data to the CH. Third, after gathering data according to the accumulation interval, the CH starts a new CH-election.

If a CH-election and a data gathering happens concurrently, there may be issues when the node is supposed to send its data because the CH may have changed and the path to the CH may be incorrect. The node assumes there is a path to the CH, but if there isn't any because of a ongoing CH-election, the node will receive an index out of range error. To avoid this issue, the node will only send it's data if there are nodes in the path to the CH. Otherwise, it will wait for a new accumulation request from the CH.

Our approach to a solution to this interference is to have the two functionalies divided into different phases similar to Low-Energy Adaptive Clustering Hierarchy (LEACH) [2], described in Chapter 3. Timers are implemented to stall the next phase by making the system sleep until the timer is ended. This issue and improvements are discussed further in Section 9.8.

8 Evaluation

This chapter describes the experimental setup and metrics used to evaluate the implemented system.

8.1 Experimental Setup

All experiments were done on a Lenovo ThinkCenter with the following specifications:

- Intel® CoreTM i5-6400T CPU @ 2.20GHz × 4
- Intel® HD Graphics 530 (Skylake GT2)
- 15,6 GiB memory and 503 GB disk
- Ubuntu 17.04 64-bit with gcc V6.3.0 compiler and GO 1.9.3

We use gopsutil, a psutil for Golang [22], to measure CPU, memory, network and process connections. Each experiment is described further in Section 8.2.

Parameter	Value
Number of nodes	100
Network grid size	500 x 500
Node broadcast width	50

Table 8.1: Parameters of simulation

8.2 Experimental Design

All experiments were conducted on a computer, where a process is simulating an Observation Unit (OU). In order to determine memory, CPU and network usage, several experiment scenarios were designed and performed.

The experiments ran with a 500 millisecond's measurement interval and the system execute minimum 10 minutes. Each OU starts it's own experiment, and each 500 millisecond during execution are the measurement values written to a log-file for further evaluation. We chose to measure with a 500 millisecond time interval because the system is implemented to run concurrently and the experimental measurements change rapidly in terms of Cluster Head (CH)-elections and data accumulations.

The log-file contains data for each time interval (500 millisecond) where one row is one node and one column is one time interval during execution. We calculate each column in the log-file to obtain an average value because of the huge amount of data. The graphs displayed contains the average values for each time interval (500 millisecond) with a standard deviation of all values.

The experiments are done executing 100 nodes with a broadcast range at 50, as shown in Table 8.1. Each node execute their own experiment when launched. During execution, the nodes in the system will communicate between each other, gossip CH-election and store and accumulate data both between themselves and at the CHs.

A CH will chose a value between one and six which indicates how many times it should gather data before it sends out a new CH-election. The experiments are ran on 4 different accumulation intervals as listed below:

- CH accumulate data two times
- CH accumulate data four times
- CH accumulate data six times

- CH accumulate data a random time between one and six times

8.2.1 Memory Measurements

We measure the memory usage to examine how much memory is consumed when executing our system. The memory usage measured, is the total percentage of RAM used by the program. It is important to keep the memory usage at a minimum due to execution on devices in real-life environment to avoid exceeding the limited memory.

We measure the memory usage independently for each node every 500 milliseconds during execution when the nodes communicate with each other, electing new CHs and accumulating data to the CHs.

- **Psutil - VirtualMemoryStat - UsedPercent:** percentage of RAM used by programs. Listing 8.1 show how the the psutil is used to measure memory usage.

Listing 8.1: Go program showing how psutil is implemented for experiments

```
1
2  func (ou *ObservationUnit) Experiments(pid int) {
3      tickChan := time.NewTicker(time.Millisecond * 500).C
4
5      infoSlice := []string{}
6
7      (...)
8
9      go func() {
10         time.Sleep(time.Second * time.Duration(batteryStart))
11         doneChan <- true
12     }()
13
14     for {
15         select {
16         case <-tickChan:
17
18         //Measure memory usage
19         mem, _ := mem.VirtualMemory()
20
21         //Round used percent memory to 3 decimals
22         memUsedPercentage := toFixed(mem.UsedPercent, 3)
23         infoSlice = append(infoSlice, memUsedPercentage)
24
25         //Measure CPU usage
26         oneCPUPercentage, _ := cpu.Percent(0, false)
27
28         //Round used CPU percent to 3 decimals
29         oneCPUPercentage[0] = toFixed(oneCPUPercentage[0], 3)
30         infoSlice = append(infoSlice, oneCPUPercentage[0])
31
32         //NET - number of connections for process by pid
33         conn, err := net.ConnectionsPid("all", int32(pid))
34         ErrorMsg("conn: ", err)
35
36         //Number of connections
37         numConn := len(conn)
38         infoSlice = append(infoSlice, numConn)
39
40         //Append to a file
41         AppendFile(path, writer, infoSlice)
42     }
43 }
```

8.2.2 CPU Measurements

We measure the total CPU usage for the computers four cores combined to document CPU utilization. This as well, is important to keep at a minimum due to execution on devices in real-life environment to avoid draining the battery.

We measure the CPU usage independently for each node every 500 milliseconds during execution when the nodes communicate with each other, electing new CHs and accumulating data to the CHs.

- **Psutil - Percent:** calculates the percentage of CPU used either per CPU or combined. Listing 8.1 show how the the psutil is used to measure CPU usage.

8.2.3 Network Usage

We measure the number of process connection and network usage when executing our system. The number of communication channels are represented by open sockets that a node have open at any given point. These open sockets may be because of discovery of other nodes, CH-election, forwarding- and data accumulation. It's important to measure the number of process connection and network usage because the system heavily relies on it and it's important to keep it to a minimum.

We measure the number of process connections and network usage independently for each node every 500 milliseconds during execution when the nodes communicate with each other, electing new CHs and accumulating data to the CHs.

- **Psutil - ConnectionsPid:** Return a list of network connections opened by a process. Listing 8.1 show how the the psutil is used to measure number of process connections and network usage.

8.2.4 Number Of Sends To Neighbours

We measure the number of sends to neighbours in the system during execution. This experiment is in connection with the network usage and communication, described in Section 8.2.3.

We measure the number of sends independently for each node every 500 milliseconds during execution every time the node sends a request to a neigh-

bour, like discovering other nodes, connecting to other nodes, CH-election or accumulating and sending data to CH.

8.2.5 Number Of Sends To Cluster Heads

We measure the number of sends to a CH during execution. The number of sends are only counted when there is a accumulation request to send data to a CH. As described in Section 8.2.3, are communication between nodes and network usage important to keep to a minimum.

We measure the number of sends to the CH independently for each node every 500 milliseconds during execution every time there is a accumulation request to send data to a CH.

8.2.6 Cluster Head Count

We measure how many times nodes have been CHs at each time interval to examine if how many CH there is at each time interval. The CH is only counted when it's done gathering all the data.

We measure the number of CHs independently for each node every 500 milliseconds during execution.

8.2.7 Cluster Head Receives Packages

We measure how many times a CH receives data from other nodes in the network. This is measured only at each CH. We want to measure this to examine the difference between the number of sends to a CH and how many times a CH have received a packet and to see the effect of the accumulation.

We measure the number of receives independently for each node every 500 milliseconds during execution every time the CH receives a request containing accumulated data.

8.3 Results

In this section we will present and discuss the results of the experiments described in the sections above.

8.3.1 Memory Usage

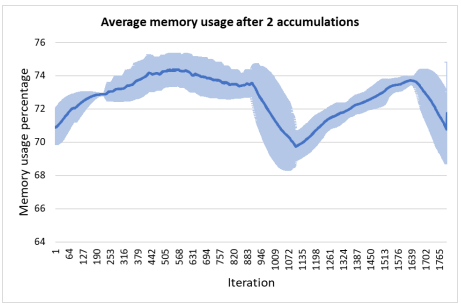
The purpose of this experiment was to examine memory footprint when executing the system.

The result is presented in Figure 8.1. We can see that the system has between 60% and 76% memory usage during execution and the graphs shows that each experiment use approximately the same amount of memory.

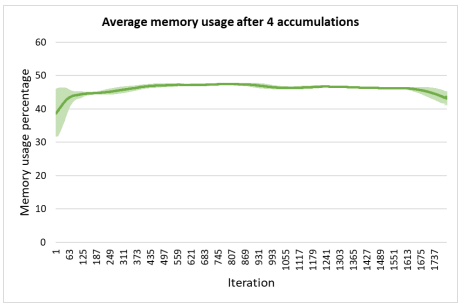
There is one graph that stands out by using under 50% of the memory during execution, which is the graph showing the memory usage after 4 accumulations, as seen in Figure 8.1b.

As for the experiment for the implementation with a random number of accumulations, as seen in Figure 8.1d, we can see that this implementation use between 58% and 68% memory while executing. There is reason to think that this implementation version uses less memory than the others because the accumulation intervals are more distributed during execution in contrast to the other accumulation intervals where all CHs accumulat data appropriately at the same time interval. This also means that the CH-elections also will occur at different times compared to the other implementation where this will happen at the same iterations per implementation.

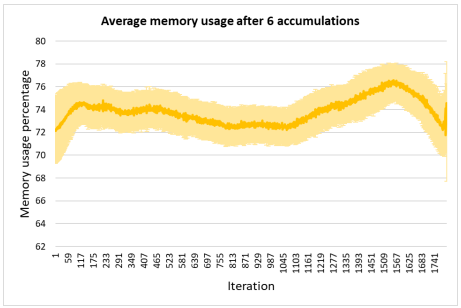
As for the experiment for the implementation with six accumulations, seen in Figure 8.1c, the memory percentage is the overall highest of all experiments. We can assume that this is because of the most amount of accumulation causing the CH to use more memory.



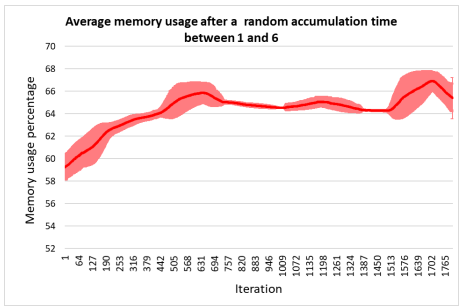
(a) Network 1



(b) Network 2



(c) Network 3



(d) Network 4

Figure 8.1: Figures show average memory percentage (dark colors) for 100 nodes per network with different accumulation intervals with standard deviation (lighter colors).

8.3.2 CPU Usage

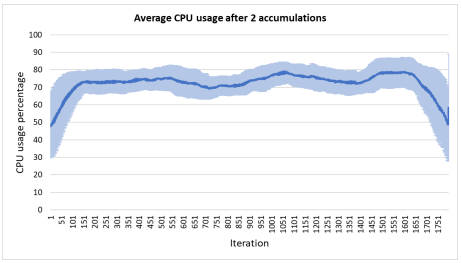
As we can see from Figure 8.2, the CPU percentage is between 50% and 95% during execution and each experiments uses approximately the same amount of CPU. The CPU percentage is stable on around 75% during the execution for each experiment. We believe that the reason there is no peaks during execution, is because of Golangs[1] built-in language features and ability to utilize multicore and networked machines.

Every node in the system spawn new goroutines when they start a new CH-election and sends- or receives requests from other nodes in the network. Golangs goroutines are very cheap to create and because they are so light, it is possible to have hundreds or thousands of them running at the same time, and in the same address space. Threads are the basic units for processor scheduling and distribution. Threads need to go to the kernel when acquiring a lock for synchronization in contrast to goroutines that are implemented on-top of OS threads and can stay in user space and acquire locks.

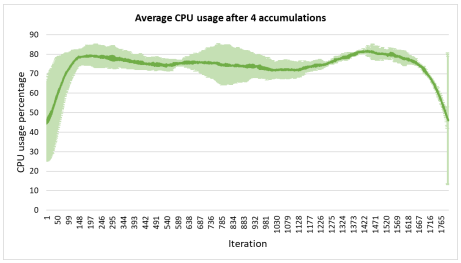
Another reason for the graphs being so stable can be because the nodes are waiting for a request to either accumulate data or elect a new CH. Most of the execution time, the nodes are waiting for requests.

From Figure 8.2, we can see that each implementation uses less CPU usage in the start and end of the execution. This is because in the beginning, nodes takes time to start and to find neighbours to connect to. Then, the CPU usage will increase because nodes that have found nearby neighbours to connect to will start CH-elections. In the end of the execution, nodes will die because of simulated energy consumption during the execution.

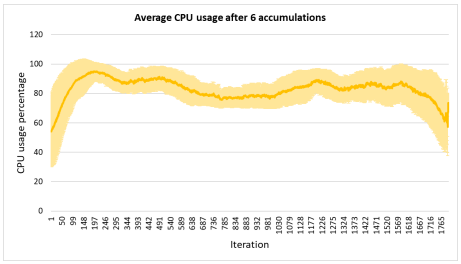
1. https://golang.org/



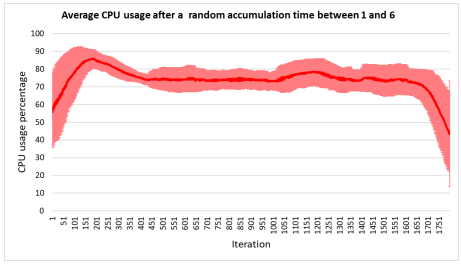
(a) Network 1



(b) Network 2



(c) Network 3



(d) Network 4

Figure 8.2: Figures show average CPU percentage (dark colors) for 100 nodes per network with different accumulation intervals with standard deviation (lighter colors).

8.3.3 Network Usage Amounted By Number Of Connections

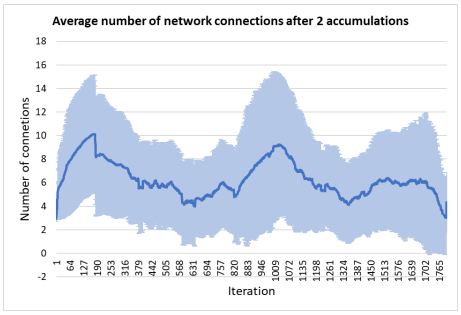
The purpose of this experiment was to measure the network usage and communication between nodes since it's important to keep this to a minimum since it's aimed to execute on small, low-power OUs. The result is presented in Figure 8.3 and show a wave like pattern. The number of connections in the system range from 1 connection to 18 connections on average per node for 100 node in the system. There is reason to believe that these high wave like patterns showing a high amount on connections is because of three reasons:

- Firstly, nodes are created and communicate with reachable nodes to connect to and eventually starts a CH-election resulting in a high amount of connections.

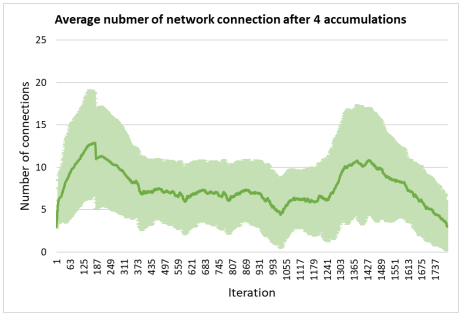
- Second, when a CH broadcasts a message that a data accumulation should be started, the message is broadcast to every node in the cluster resulting in many open connections.

- At last, after accumulating data from other nodes in the cluster, the CH gossip a new CH-election calculation and then gossip a new CH-election.

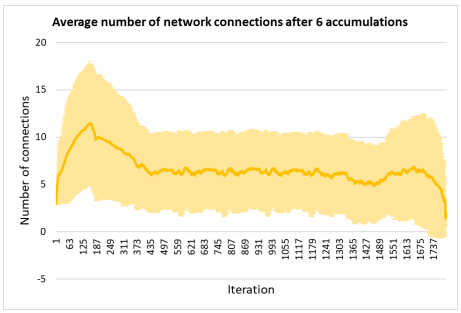
There is also reason to think that these wave like patterns occurs at different time intervals is because of the different implementations accumulate data and elect CH at different time intervals. Each cluster of nodes may also get out of sync with the other clusters resulting in that accumulation of data and CH-elections occurs at different iterations. This may smooth out the graphs eventually, as seen in Figure 8.3c



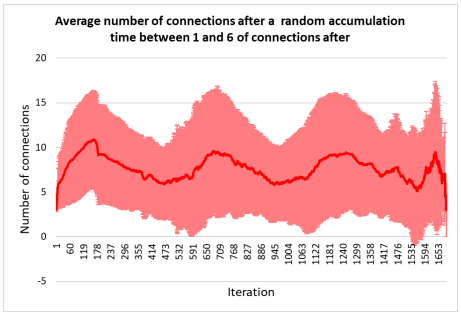
(a) Network 1



(b) Network 2

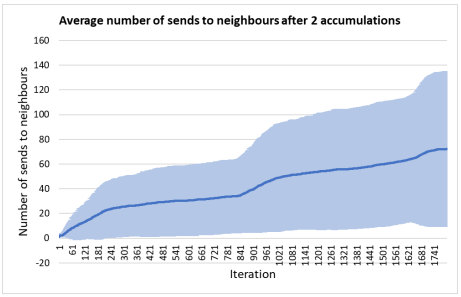


(c) Network 3

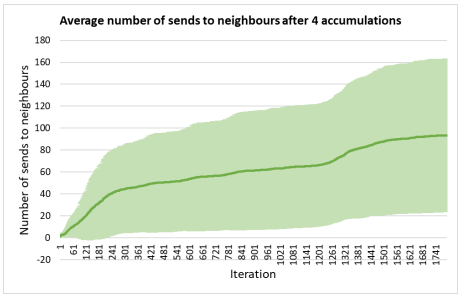


(d) Network 4

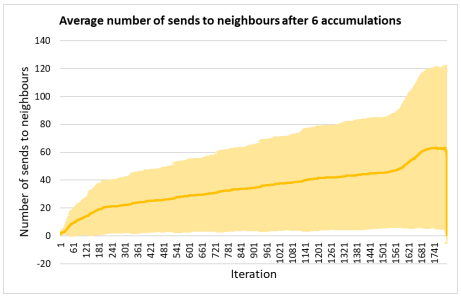
Figure 8.3: Figures show average network connections (dark colors) for 100 nodes per network with different accumulation intervals with standard deviation (lighter colors).



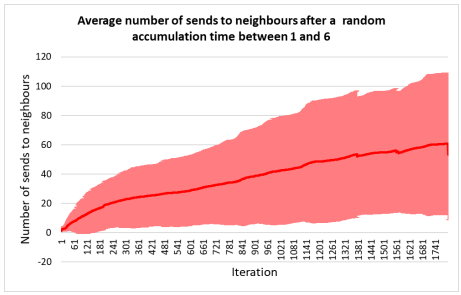
(a) Network 1



(b) Network 2



(c) Network 3



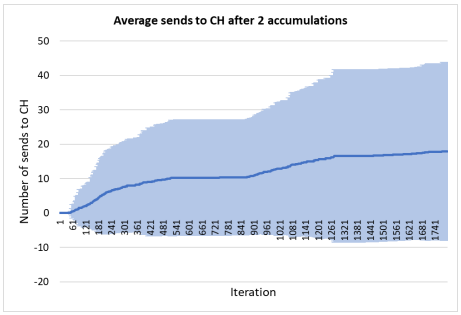
(d) Network 4

Figure 8.4: Figures show average number of sends from nodes (dark colors) to neighbour nodes for 100 nodes per network with different accumulation intervals with standard deviation (lighter colors).

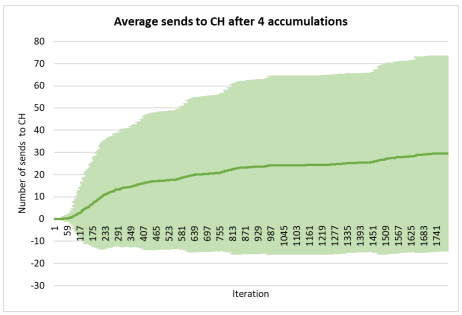
8.3.4 Number Of Sends To Neighbours

Since it's important to keep communication and network usage to a minimum, we measure the number of sends from nodes to neighbours. As we can see from Figure 8.4, the number of sends from one node to another increases during execution. The average number of sends from nodes range between 1 send up til around 95 sends. This is an expected result since each time a node communicate with another node, it's counted and therefore will increase during execution.

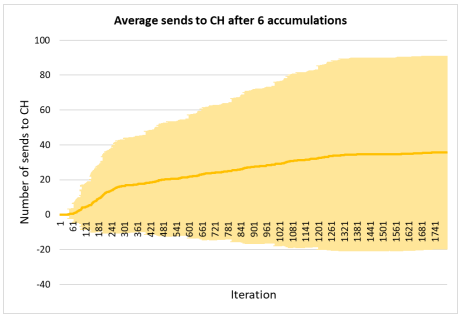
As Figure 8.4 also show, is how some nodes send few messages while other nodes send a lot of requests. As we can see from Figure 8.4b, some nodes have sent requests over 160 times and other nodes only around 20 requests. This will vary in terms of where the nodes are placed in the network. Those nodes that have a low rate of sends are most likely placed at the outer edge of the cluster while the nodes that have a high rate are most likely placed in the path to the CH for many nodes resulting in a higher send rate.



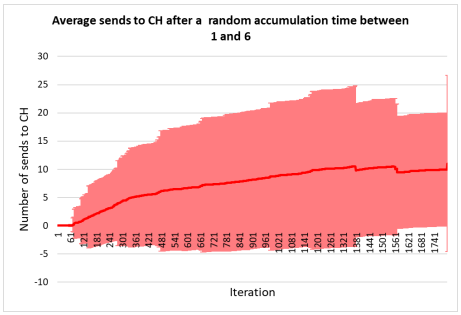
(a) Network 1



(b) Network 2



(c) Network 3



(d) Network 4

Figure 8.5: Figures show average number of sends to CHs (dark colors) for 100 nodes per network with different accumulation intervals with standard deviation (lighter colors).

8.3.5 Number Of Sends To Cluster Heads

Number of sends to CHs is also connected to the experiment with communication and network usage. The result is presented in Figure 8.5 where we can see that the number of sends to CHs increase during execution. This is also an expected result in thought of the CH are gathering more and more data. The average number of sends to CH range between 0 sends up til around 35 sends.

Figure 8.5d show how nodes quickly accumulates data to the CH by having the CH chose a random value of how many times it should accumulate data between one and six. Even though the CH starts gathering in a more varied way, it has less sends to the CH than the other implementations. We can also see that this graph decrease right after 1400 iterations. The reason for this is assumable because nodes starts dying of energy consumption because of a more spread CH-election and data accumulation during execution causing the nodes do drain more of their limited battery.

8.3.6 Cluster Head Count

The result is presented in Figure 8.6. The graphs are as expected flat during parts of the execution because the CH-count is only counted when the CH is done gathering data. There is reason to think that the all graphs, except the graph of CH-count after a random number of accumulations, have flat graphs during execution because all the CH-counts happens approximately at the same time interval and don't change until the next CH-election.

We can see that the CH-count after 2 accumulations increase to almost double because of CH-elections in the clusters. Since there is only 2 gathering of data per CH, the CH will get started earlier with a CH-election. Our implementation also support that a CH can be elected as CH again, which can explain why the graph increase significantly. Since it's the average value from the 100 nodes running during the experiment, some CHs may have been elected CH again causing the graph to increase to double.

CH-election after 6 accumulations have also a little increase by around 2. This may also be caused by some a CH-election where some previous CHs have been elected again.

Counting the number of CH after a random number of accumulations, show a more increasing graph during execution because of the accumulations happens at different time interval causing the CH-election to be more spread during execution. We can also see that this graph stop right after 1600 iterations. The reason for this is assumable because nodes starts dying of energy consumption because of a more spread CH-election and data accumulation during execution causing the nodes do drain more of their limited battery.

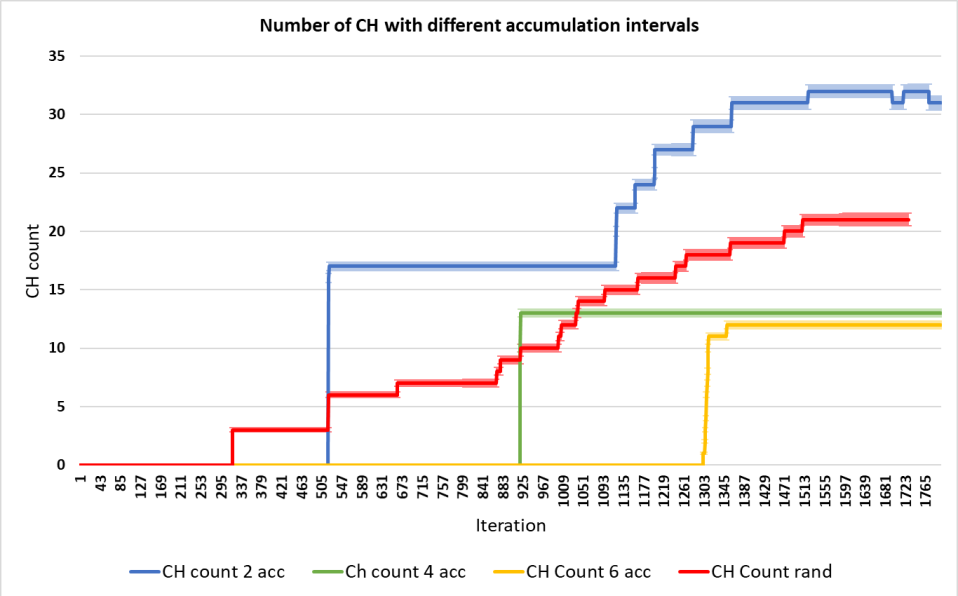
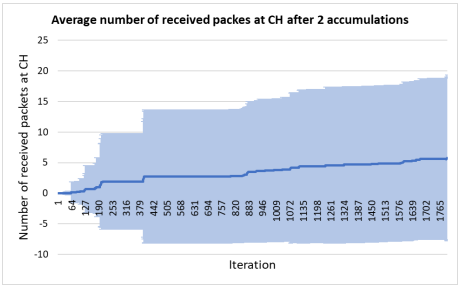


Figure 8.6: Figure shows the number of CH for 100 nodes per network with different accumulation intervals.

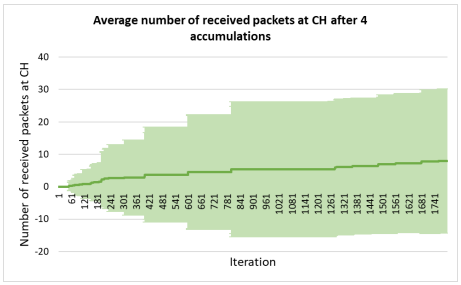
8.3.7 Cluster Head Receives Packages

As we can see from Figure 8.7, are the number of packets CHs receives relatively similar during each experiment. This result as well, is expected because the CHs will receive more data each time they ask for it.

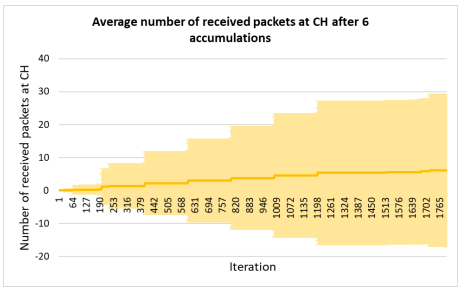
Figure 8.8 show the number of sends from nodes to neighbours, sends from nodes to CHs and number of times CHs have received data. As we can see, are the number of received packets less than sends to CHs. It is therefore reason to think that the nodes accumulate data to reduce the number of packets a CH receives. The reason for the drop down at the end of the iteration is caused by nodes shutting down due to their limited energy and they these nodes are not taking in account when calculating an average.



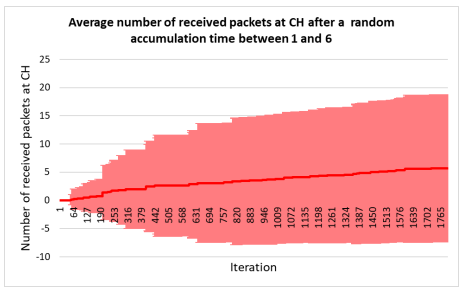
(a) Network 1



(b) Network 2



(c) Network 3



(d) Network 4

Figure 8.7: Figures show average number of CH (dark colors) for 100 nodes per network with different accumulation intervals with standard deviation (lighter colors).

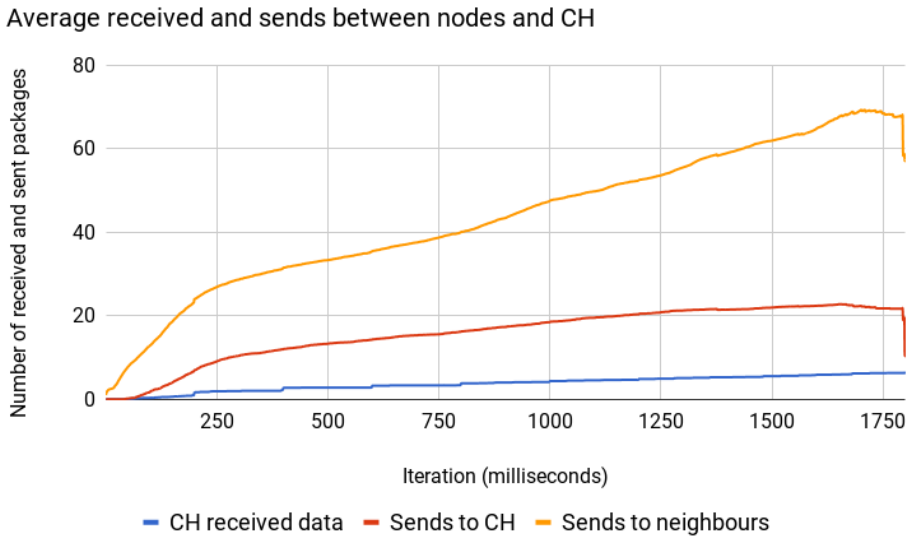


Figure 8.8: Figure shows average number of CH with different accumulation intervals.

9 Discussion

This chapter discusses our approach, experiences, how we solved the problem and why we chose the solution we ended up with.

9.1 Availability Of Nodes In The System

9.1.1 Connect To Neighbours

In the current approach a node can connect to nearby nodes and be a part of a cluster without any restrictions. In our first approach when a node received a request from a new node, it would forward the request to the Cluster Head (CH). The CH then determine if the new node is allowed connect to the cluster or not. However, making a CH take this decision could be problematic because the CH needs to know information about the whole network or at least its own cluster, and also need some requirements for when a node can join and not. Some requirements that a CH can consider are:

- Number of nodes per cluster
- Number of neighbours a node can have
- Distance from new node to CH

- Node characteristics (bandwidth, energy level, memory etc.)

Another interference with having a CH take the decision is that the CH may be unavailable because it's sleeping or if it's unreachable because nodes in the path to the CH are sleeping or unavailable due to e.g. saving battery.

9.1.2 Ping Neighbours

An improvement to check if neighbours are alive or not, is to ping neighbouring nodes frequently to check if they are alive or not. However, pinging other nodes frequently will eventually flood the network. An improved solution is to only ping neighbours if there is a need for communication between nodes. For example to ping neighbours before forwarding a message to the CH. This ping-request will most likely contain a wake-up call to the next node in the path to the CH so nodes are awake when receiving a forwarding request to the CH. Another approach is to have time slots for when nodes are awake. During these time slots, nodes could ping other nodes to check if they're alive or not. CH-elections and accumulating and forwarding data would also happen during these time slots.

9.1.3 Node Waking Up After Sleeping/Being Unavailable

At the current approach, there is no timestamp or schedule for when the CH was elected. If a node was unavailable during the CH-election, it will most likely forward its data to the old CH. However, this isn't necessary a drawback since the node receiving this data will forward the accumulated data in the next gathering phase. To improve this approach and avoid inconsistency of the CH, we could implement a solution to ask the nodes neighbours who the CH is. Each CH-election should have a timestamp to compare in case the node receives multiple different answers. Then the node could compare the different timestamps and choose the timestamp that is the biggest or the closets to its own clock, depending on the implementation.

9.2 Cluster Head Election

Being a CH is more energy intensive than being a regular cluster node because it may transfers data over longer distances and performs more tasks. If the CH is chosen a prior, meaning the CH is chosen before the nodes have knowledge of each other and created clusters, e.g. by a Base Station (BS) without any knowledge about the network, then the node would quickly use up its limited energy. Once a CH runs out of battery, it is no longer operational and all nodes belonging to the cluster will lose their communication ability. To avoid this problem, our approach will not chose CHs a priori, but instead have a CH-election algorithm to possibly rotate the CH.

9.2.1 Cluster Head Calculation

Presently, the CH-election is based on which node has the highest score between 0 and 1. This intentionally simplified implementation does not take in account many other realistic aspects which would improve the CH-election. To improve this approach and make it more realistic in terms of saving battery lifetime and a real-life scenario for deploying sensors in the Arctic tundra, we could have used several sub-factors listed below:

- Number of nodes between a node and CH
- Number of neighbours for the node
- Access to BS
- Power left on node
- Bandwidth - WiFi, LoRaWan[1], Ethernet
- Prior history
- Network traffic on node

9.2.2 Gossip Information Between Nodes

The main advantages of gossiping is its ability to scale. Gossiping have no centralized component where information is coordinated and is therefore an excellent way to rapidly spread information among a large number of nodes

1. https://lora-alliance.org/about-lorawan

using only local information. However, gossiping can not guarantee that all nodes will receive the information [11].

To avoid flooding the network, each message sent between nodes have an ID so the node can check if it has received the same message before. If the node have received the message earlier, the node doesn't need to forward the message because it has forwarded the message in an earlier gossip, as described in Section 7.3.

Another approach to avoid flooding the network and reduce nodes energy level is to limit the number of hops when gossiping to other nodes. To avoid this flooding can each packet have a hop count and every time a packet hops, its hop count increment. When a packet's hop counts equals a hop limit, the packet will be discarded. Flooding the network with updates as the current approach does, ensures eventual consistency.

9.2.3 When To Elect A New Cluster Head

The present approach starts a new CH-election in two scenarios: either when a new node joins the cluster or when the CH have accumulated data 'X' times. Other approaches such as Low-Energy Adaptive Clustering Hierarchy (LEACH) [2], mentioned in Chapter 3, have multiple phases during execution where one phase is to elect a new CH which occurs periodic.

The current approach is similar to LEACH using different phases for electing a new CH and accumulate data. We chose this solution because of concurrency issue when these two events happen at the same time, as mentioned in Section 7.6. This issue is also discussed further in Section 9.8.

There is also a need for electing a new CH if the CH itself crashes, runs out of battery or something happens like a flood or if the node gets destroyed by an animal. This can be done by either ping the CH to check if it's alive during time slots, as discussed in Section 9.1. If there is no response from the CH, the node responsible for the ping starts a new CH-election.

9.3 Remember Previous Cluster Head

At the present approach, nodes do not know that they have been CHs in an earlier election. The nodes should know if they have been elected to CHs earlier because they have accumulated data from other nodes in the cluster. This data should be sent to a BS. The CHs can have access to a BS during their whole

lifetime or at some points during their lifetime. Access to BSs are discussed further in Section 9.9

To improve our current approach, each CH should remember that they have been elected as CH in an earlier election. Eventually, when the CH have been in contact with a BS that collects all the CHs data, they can forget that they have been CHs and also remove their data to increase memory capacity.

9.4 Multiple Cluster Heads

The current approach have one CH per cluster. If there are many nodes in one cluster, there can be a lot of work load for the CH, and the path to the CH can be long for some nodes. An improvement to this is to introduce multiple CH to load balance work and may also provide shorter path for some nodes. The question is then which CH should a node choose to be its CH if there are multiple. CH-elections will be based on several sub-factors described in Section 9.2.1. If we assume that all nodes that are qualified to become CHs are equivalent in terms of battery lifetime, network bandwidth and number of neighbours, the node should choose the CH with the shortest path to avoid flooding the network with requests and to avoid draining the limited energy on more nodes than necessary. Having multiple CHs that accumulates data would also support scalability and performance issues in terms of balance the work load between CHs.

9.5 Path To Cluster Head

One advantage with the present approach is that the CH-election chose the shortest path from a node to the CH to avoid flooding the network with requests. The idea to use the shortest path [21], is presumably not beneficial to change in any approach. However, it may be that some nodes in a path to the CH have a limited battery lifetime and therefore should not be in a path to the CH. This is a corner case which will be difficult to give a right answer to since the paths and nodes can occur in many different scenarios.

9.5.1 Multi-hop or single-hop routing

Wireless Sensor Network (WSN) consists of hundreds or thousands of low-cost micro-sensor nodes and their main task is to collect information of the interested area and broadcast the information to a BS. An easy approach to

achieve this task is to make each sensor node transmit their data directly to the BS, as Figure 9.1 shows. However, the problem is that the BS may be placed far away from the sensor node so a direct transmission would not be possible. Another problem can be that the routing path from a sensor node to the BS is long and the node will require more power consumption than available.

In some cases can multi-hop routing be more energy efficient than single-hop routing[27]. Our approach is currently using multi-hop routing to deliver accumulated data to the CH. To use single-hop routing in our approach would not be efficient in terms of energy consumption or sensor nodes radio range. In a real-life scenario where nodes are deployed in the Arctic Tundra, all nodes would not be able to reach each other by radio range or a BS.

However, multi-hop routing has it's drawbacks. There is more likely a packet will be lost during multi-hop routing than single-hop because the packet will have multiple nodes in it's path that can be unavailable during transmission. Multi-hop routing can also introduce problems of routing in terms of different paths to a BS or a CH, as seen in Figure 9.1. As mentioned in in Section 9.5, will nodes in our approach choose the shortest routing path to a CH.

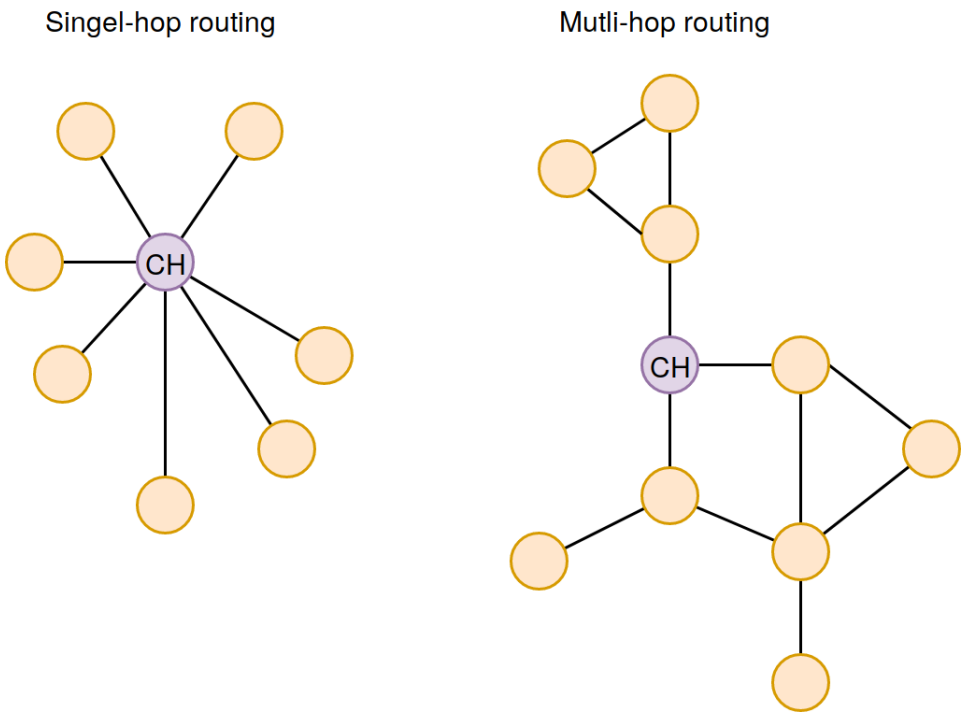


Figure 9.1: Figure show single-hop routing and multi-hop routing.

9.6 Data Accumulation

In the current approach, the CHs are always the ones that initiate a request to the other nodes in the cluster for collecting data. This means that regular nodes can not decide that they want to send data to their CH. Another approach would be that the node's sends their data when they have new data or when they have almost used up their memory capacity. However, if nodes have new data regularly, this approach would not be efficient in terms of energy consumption because other nodes in the routing path must be awake to transmit the data further to the CH. There is reason to think that there is a possibility that the majority of the nodes in the clusters are awake most of their time instead of sleeping and saving battery.

9.7 Replication Of Data

In the current approach will nodes only accumulate data that haven't been accumulated earlier. This approach will therefore have some degree of replication because a nodes data will be accumulated at the next node in the routing path.

It may take long time before a CH gets access to a BS or is contacted by a data mule that collects the data. Meanwhile, the CH may fail or die due to battery limitations and all its accumulated data will be lost.

An approach to this problem is to only let nodes delete its data if the CH have delivered all the data to a BS or a data mule. Otherwise, the data is stored at each node. Replication of node data is important if a node fails and we need to access the nodes data. However, replication of data also require more memory but sensor nodes have limited memory. In this case, there is a need for finding a degree of replica that can both be memory efficient and handling system failures.

Another approach is to have multiple CHs that collects the same data, as discussed in Section 9.4. In this approach, nodes that are not CHs can delete their data when they have sent their data to the CH to minimize memory usage.

9.8 Concurrent CH-Election And Data Accumulation

The current approach provides an eventual consistency when electing a new CH as mentioned before. This may raise some issues when a data gathering occurs when an election is ongoing due to rapidly change of CHs and paths to CH before all nodes have an consistent CH-election, as mentioned in Section 7.6.

LEACH have divided their system into different rounds, as described in Chapter 3. To improve our approach, we could have implemented our system to be divided into rounds such as in LEACH. Instead of having timers to stall the next phase, an improvement to our approach could be to have a status at each node saying if the node is going through a CH-election, if it accumulates data to the CH or if it's just waiting for requests. In this way, when one status is true, e.g. a CH-election has status true, can't a data accumulation happen because its status will be false.

9.9 Base Station Access

At the current approach, all nodes have the same broadcast range to reach nearby neighbours. This means that there is no node that have access to a BS where it can deliver it's data.

Another question is how nodes or CHs have contact with BSs. As discussed in Section 9.2, can a node be elected as CH if it has access to a BS, if there are any placed in the same area as the nodes. Another approach is to use a data mules to collect data from the CHs. One problem is how the CH should be contacted since the BS or data mule don't know who has the data in the cluster. One approach is that they know who the CHs are, another approach is to let the BS or data mule contact an arbitrary node and let the node alert the CH that it should send it's data. The node can also tell the mule where the CH is located by giving the mule the CHs x- and y-coordinates.

10 Conclusion

In this thesis, we have implemented a prototype of a Wireless Sensor Network (WSN) system where nodes observe and accumulate data from other nodes for further use. We describe a system where nodes discover each other through a broadcast range and together they form clusters. Each cluster elect a Cluster Head (CH) which is responsible for sending out a request for gather and accumulate data from the other nodes in the cluster. The role as CH is rotating among the nodes to conserve battery. Even though our approach works as intended, there is a need for improvements in how a CH is elected and when a CH should accumulate data.

Our experiments showed that the system have a steady CPU and memory usage. We can also see that the number of receiving packets of accumulated data is lower than sent packets with accumulated data which indicates that the nodes in the system accumulates data when intended to reduce traffic on the paths to the CHs. There is also a big variety of how many requests nodes in the system sends, depending on their location in the network.

A future system could further investigate the benefits of having multiple CHs and how to gather and accumulate data more efficiently. There is still a need for conducting further work for a real-life environment in the Arctic Tundra.

11

Future Work

We will outline some of the areas that can be elaborated in future work. These include:

- **Availability of node**: In the current implementation, nodes can join the cluster without any requirements, as discussed in Section 9.1. We discuss how how we could improve the implementation by having the Cluster Head (CH) take the decision if a node can join the cluster or not. We also discussed how to improve a nodes knowledge about the network around them and how to get the newest information.

- **CH-election:** The CH-election, together with the CH calculation, in the current solution is not the best, but it is an introduction to a solution. As mentioned in Section 9.2, would another approach be to include sub-factors such as power left on node, network traffic, number of neighbours etc, to calculate which nodes that are most qualified to be a CH.

- **Multiple CHs:** This implementation of the system does not support multiple CHs in one cluster. In Section Section 9.4, we discuss how we could have improve our approach by to load balance work, possibly provide shorter paths for some nodes to CHs and minimize scalability and performance issues.

- **Access to Base Station (BS):** The current approach have no BS or data mule to collect the data. In Section 9.9, we discuss how the cluster should

get access to a CH by implementing either a BS or a data mule and how these should contact the nodes in the network.

Bibliography

[1] Åshild Ø. Pedersen, A. Stien, E. Soininen, and R. A. Ims, *Climate-ecological observatory for arctic tundra-status 2016*, Mars 2016, in *Fram Forum 2016, pages 36-43.*

[2] W. R. Heinzelman and A. Chandrakasan and H. Balakrishnan, *Energy-efficient communication protocol for wireless microsensor networks*, 2000, in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 10 pp. vol.2-.*

[3] K. Latif and M. Jaffar and N. Javaid and M. N. Saqib and U. Qasim and Z. A. Khan, *Performance Analysis of Hierarchical Routing Protocols in Wireless Sensor Networks*, 2012, in *2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications, pp. 620-625.*

[4] Z. Han and J. Wu and J. Zhang and L. Liu and K. Tian, *A General Self-Organized Tree-Based Energy-Balance Routing Protocol for Wireless Sensor Network*, 2014, in *IEEE Transactions on Nuclear Science Vol.61, Nr.2, pp. 732-740.*

[5] J. N. Al-Karaki and A. E. Kamal, *Routing techniques in wireless sensor networks: a survey*, 2004, in *IEEE Wireless Communications Vol.11, Nr.6, pp. 6-28.*

[6] S. Lindsey and C. S. Raghavendra, *PEGASIS: Power-efficient gathering in sensor information systems*, 2002, in *Proceedings, IEEE Aerospace Conference Vol.3, pp. 3-1125-3-1130.*

[7] A. K. Mishra and R. Kumar and J. Singh, *A review on fuzzy logic based clustering algorithms for wireless sensor networks*, 2015, in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), pp. 489-494.*

[8] Indranil Gupta and D. Riordan and Srinivas Sampalli, *Cluster-head election using fuzzy logic for wireless sensor networks*, 2005, in *3rd Annual Communication Networks and Services Research Conference (CNSR'05), pp. 255-260*.

[9] Maryam Sabet and Hamid Reza Naji, *A decentralized energy efficient hierarchical cluster-based routing algorithm for wireless sensor networks*, 2015, in *AEU - International Journal of Electronics and Communications Vol.69, Nr.5, pp. 790 - 799*.

[10] W. B. Heinzelman and A. P. Chandrakasan and H. Balakrishnan, *An application-specific protocol architecture for wireless microsensor networks*, in *IEEE Transactions on Wireless Communications Vol.1, No.4, October 2002, pp. 660-670*.

[11] Demers, Alan and Greene, Dan and Hauser, Carl and Irish, Wes and Larson, John and Shenker, Scott and Sturgis, Howard and Swinehart, Dan and Terry, Doug, *Epidemic algorithms for replicated database maintenance*, in *ACM: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, 1987, pp. 1–12*.

[12] Chen, Bai and Zhang, Yaxiao and Li, Yuxian and Hao, Xiao-Chen and Fang, Yan, *A Clustering Algorithm of Cluster-head Optimization for Wireless Sensor Networks Based on Energy*, in *Journal of Information and Computational Science, Vol.8, 2011*.

[13] M. Tong and M. Tang, *LEACH-B: An Improved LEACH Protocol for Wireless Sensor Network*, in *J2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), 2010, pp. 1-4*.

[14] Juang, Philo and Oki, Hidekazu and Wang, Yong and Martonosi, Margaret and Peh, Li Shiuan and Rubenstein, Daniel, *Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet*, in *ACM: SIGARCH Comput. Archit. News, Vol.30, No.5, December 2002, pp. 96–107*.

[15] Jelasity, Márk and Voulgaris, Spyros and Guerraoui, Rachid and Kermarrec, Anne-Marie and van Steen, Maarten, *Gossip-based Peer Sampling*, in *ACM Trans. Comput. Syst., Vol.25, No.3, August 2007*.

[16] Draves, Richard and Padhye, Jitendra and Zill, Brian, *Routing in Multi-radio, Multi-hop Wireless Mesh Networks*, in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking, 2004, pp.114–*

128.

[17] Holger Karl, Andreas Willig, *Protocols and Architectures for Wireless Sensor Networks*, John Wiley & Sons, Ltd, 2006.

[18] Xu, Ya and Heidemann, John and Estrin, Deborah, *Geography-informed Energy Conservation for Ad Hoc Routing*, in *MobiCom '01: Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, 2001, pp.70–84*.

[19] Yu, Yan and Govindan, Ramesh and Estrin, Deborah, *Geographical and Energy Aware Routing: a recursive data dissemination protocol for wireless sensor networks*, in *UCLA Computer Science Department Technical Report, Vol.463, 2001*.

[20] Tanenbaum, Andrew S. and Steen, Maarten van, *Distributed Systems: Principles and Paradigms (2Nd Edition)*, Prentice-Hall, Inc., 2014.

[21] Dijkstra, E. W., *A note on two problems in connexion with graphs*, in *Numerische Mathematik, Vol.1, No.1, December 1959, pp.269–271*.

[22] W. Shirou, *gopsutil: psutil for golang*, in *GitHub: GitHub repository*, `https://github.com/shirou/gopsutil`, last commit= *57f370e*.

[23] H. Garcia-Molina, *Elections in a Distributed Computing System*, in *IEEE Transactions on Computers, 1982, Vol.C-31, No.1, pp.48-59*.

[24] H. Garcia-Molina, *Euclidean space, ed. (2001)[1994]*, in *Encyclopedia of Mathematics, Springer Science+Business Media B.V. / Kluwer Academic Publishers*. URL: `http://www.encyclopediaofmath.org/index.php?title=Euclidean_space&oldid=38673`.

[25] J. N. Al-Karaki and R. Ul-Mustafa and A. E. Kamal, *Data aggregation in wireless sensor networks - exact and approximate algorithms*, in *2004 Workshop on High Performance Switching and Routing, 2004. HPSR.*, pp.241-245.

[26] Estrin, Deborah and Govindan, Ramesh and Heidemann, John and Kumar, Satish, *Next Century Challenges: Scalable Coordination in Sensor Networks*, in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1999, pp.263–270.

[27] S. Fedor and M. Collier, *On the Problem of Energy Efficiency of Multi-Hop vs One-Hop Routing in Wireless Sensor Networks*, in *Advanced Information*

Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on, 2007, Vol.2, pp.380-385.

[28] O. Anshus, *Distributed Arctic Observatory (DAO): A Cyber-Physical System for Ubiquitous Data and Services Covering the Arctic Tundra*, 2018 in https://www.forskningsradet.no/prognett-iktpluss/Nyheter/NOK_200_million_for_13_research_projects_on_Ubiquitous_data_and_services/1254032932215?lang=no, "Norwegian Research Council (NRC) - Project no: 270672"

Appendices

A

Running The System

Running The System

- Get and install Golang: `https://golang.org/doc/install`
- Run *go get github.com/shirou/gopsutil/...*
- Locate folder *master_thesis_code*
- Run *go install ./...*

Start simulator

- Locate folder *master_thesis_code/cmd/simulator*
- *go run simulation.go -numCh=5*

Start One Observation Unit

- Locate folder *master_thesis_code/cmd/server*
- *go run server.go run -Simport=8080 -host=localhost -port=:8082*

  - **-Simport=:8080**: port of simulator
  - **-host:localhost**: host of Observation Unit (OU)
  - **-port=**: port of OU

Start Multiple Observation Units

- *go run run_all.go 3*
  - **3**: number of OUs