UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

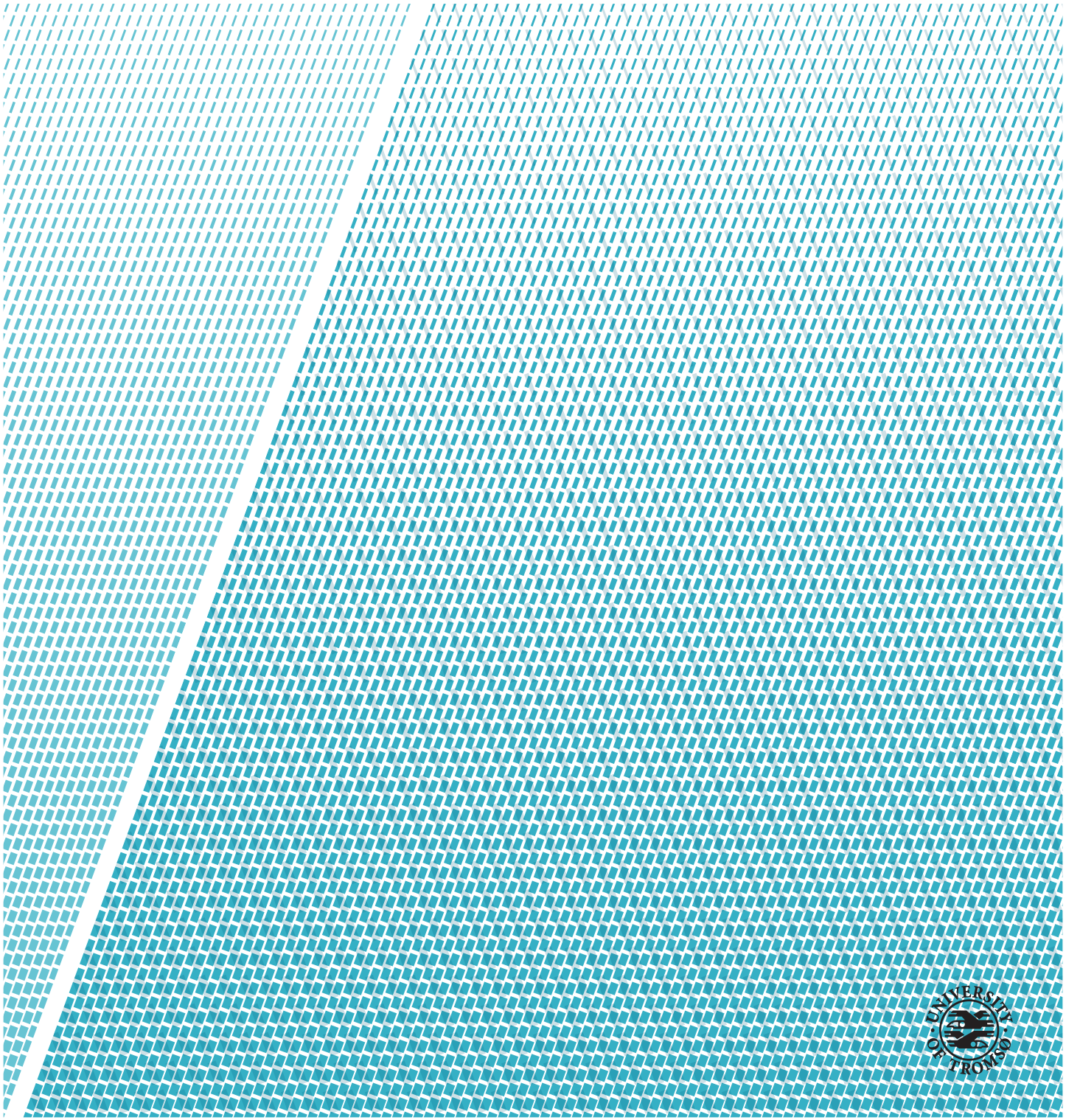
Faculty of Science and Technology
Department of Computer Science

Peer Observations of Observation Units

—

Camilla Stormoen

INF-3981 Master's Thesis in Computer Science ... May 2018



Abstract

What is wrong with the world? Motivation 1-3 sentences, Arch, Des, Imp, Exp 1,2-3 sentences, results and main conclusion.

Acknowledgements

Contents

List of Figures

List of Tables

1 Introduction

FRA CAPSTONE: *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]. The Climate-ecological Observatory for Arctic Tundra (COAT) is a long-term research project developed by five Fram Center[1] institutions. 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[10].*

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 and implementation of a peer observation that can observe and accumulate data from in-situ Observation Unit (OU)s.

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

1.1 Motivation

The motivation behind this project is...

The purpose is to fetch and accumulate data observed by OUs for further use.

1.2 Contributions

The dissertation makes the following contributions:

- An introduction to WSNs and routing techniques in WSNs.
- An implementation and description of a system ...
- An evaluation of the system.
- Thoughts on future work and further improvements to the current system.

1.3 Assumptions

Avgrense viktig!

1.4 Limitations

Avgrense viktig!

1.5 Outline

This thesis is structured into X 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 shows ...

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

Chapter 8 Evaluation

Chapter 9 discusses the results etc ..

Chapter 10 concludes the thesis.

Chapter 11 suggests future work to improve the system.

Chapter 12 Appendix

2

Routing Techniques in WSNs

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.

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.

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

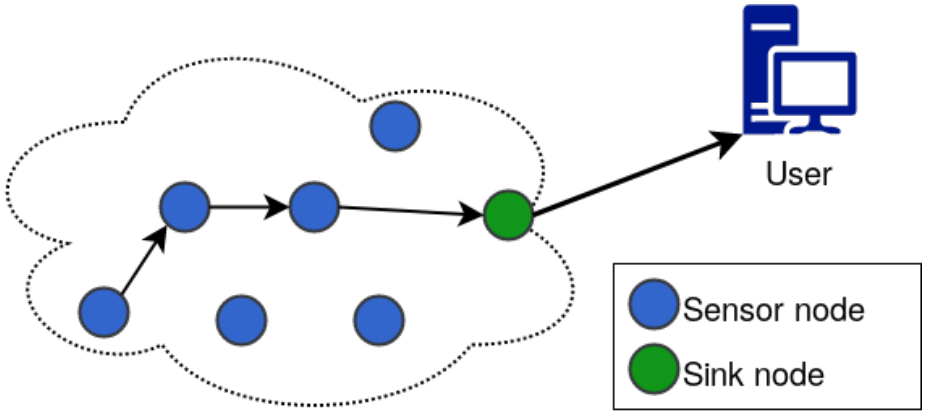


Figure 2.1: Figure shows an example of a multi-hop WSN architecture.

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 thatyhe 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 WSNs, 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 is began.

LEACH do not consider a node's energy level when calculating the CH. It has therefor 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 "variables" such as battery level or number of nodes in the cluster.

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 neighbor. The sensor nodes will also take turns on being the leader for transmitting data

to the BS and therefor 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].

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 leader. 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 OUs, connect to nearby nodes and create clusters. Using an unstructured network would be efficient with regards to when a large number of nodes are frequently joining and leaving the network since its highly robust/**low cost** in the face of high rates of churn.

The node lookup service in Figure 4 should provide a functionality for nodes to discover other nodes. Together the nodes 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 Head (CH)s 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.

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

It is not feasible to deploy nodes in the Arctic Tundra at such an early development, there should be a functionality to make nodes discover nearby nodes they can connect to. The technical idea is shown in Figure 4.1.

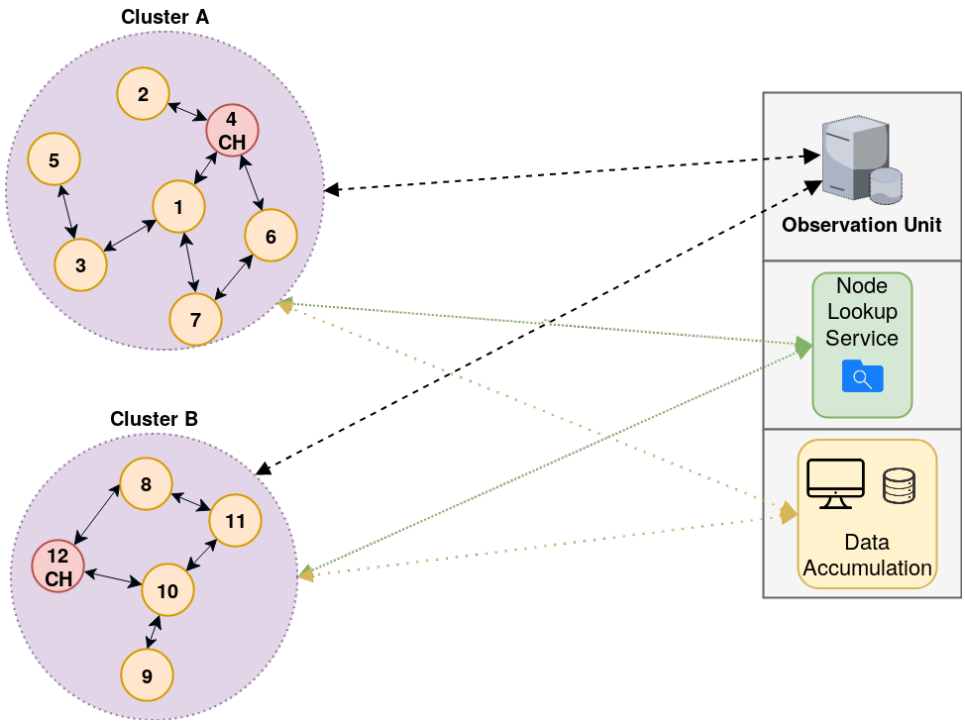


Figure 4.1: Figure shows the architecture of the system.

Node Lookup Serivce

The node lookup service in Figure 4 should provide a functionality for nodes to discover other nodes.

Clusters

The clusters are created when nodes contact the node lookup service and contact reachable nodes. Each cluster will have a leader, also called a Cluster Head (CH).

Observation Unit

Data Accumulation

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 be stored locally on the node in the cluster.

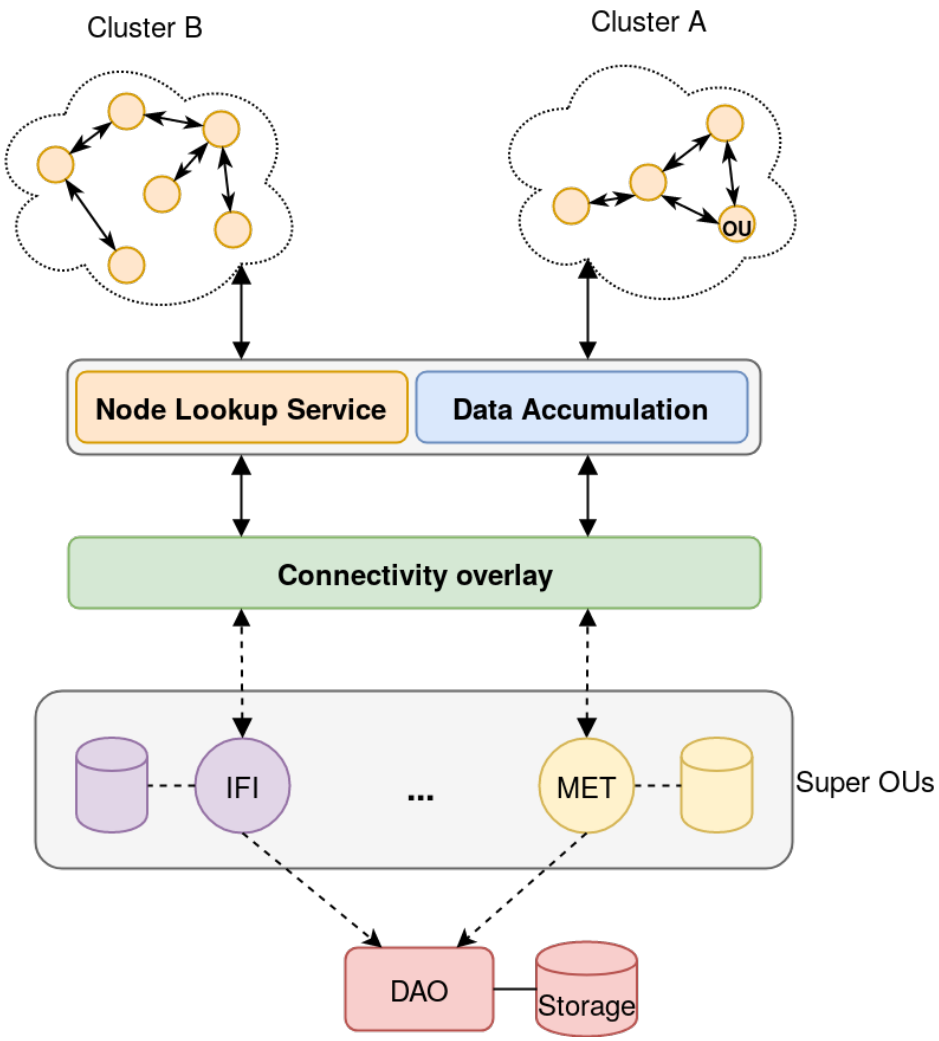


Figure 5.1: Figure shows the architecture of the system.

5.3 Data Accumulation

A node will accumulate data received from another node if its data hasn't been sent to the CH. A CH will gather data from nodes in order to provide the accumulated data to a 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 election calculation request is sent from the leader. 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 nodes data receiving this requests isn't sent to the leader 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*.

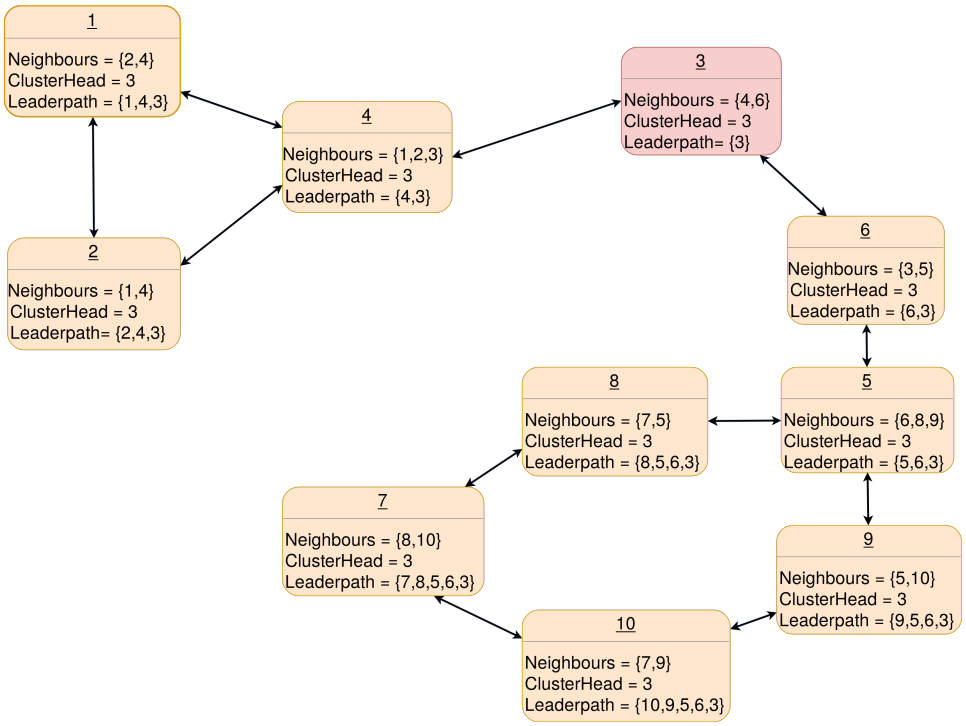


Figure 6.1: Figure show design of the system.

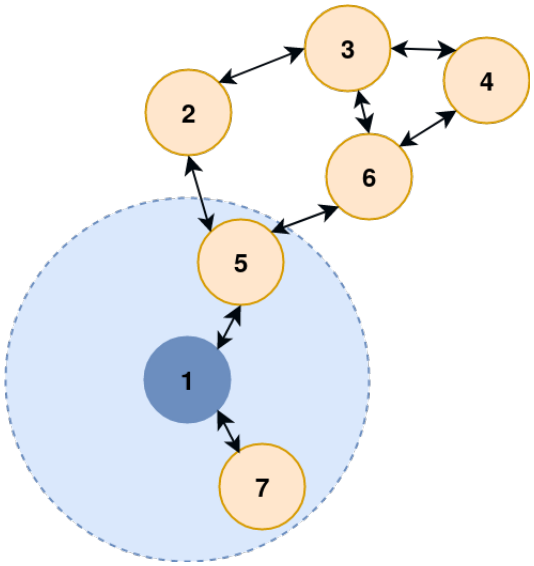


Figure 6.2: Figure show broadcast range of a OU.

6.2 Cluster Head Election

A 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. 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.

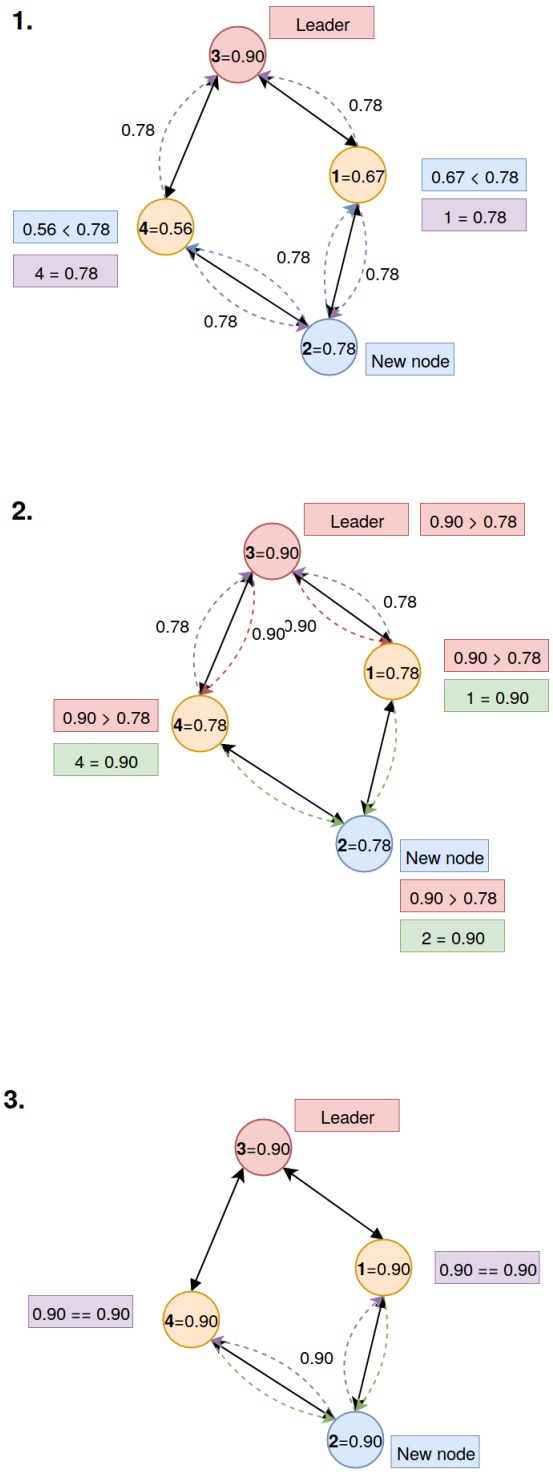


Figure 6.3: Figure show how a new node starts a CH election.

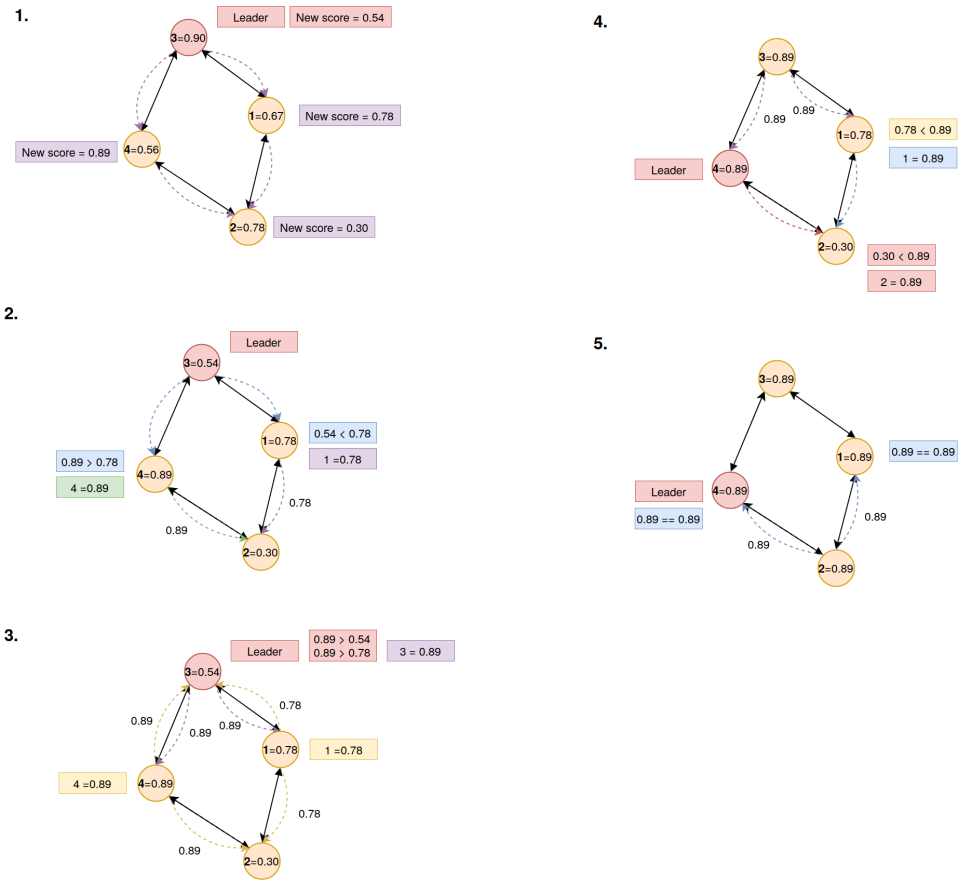


Figure 6.4: Figure show how a CH starts a CH election.

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 nodes the cluster to calculate a new CH-score. Then will the CH start a new election and gossip this election to the other nodes. The nodes receiving this gossip message will start their leader-election as described in the section above. The election is illustrated in Figure 6.4.

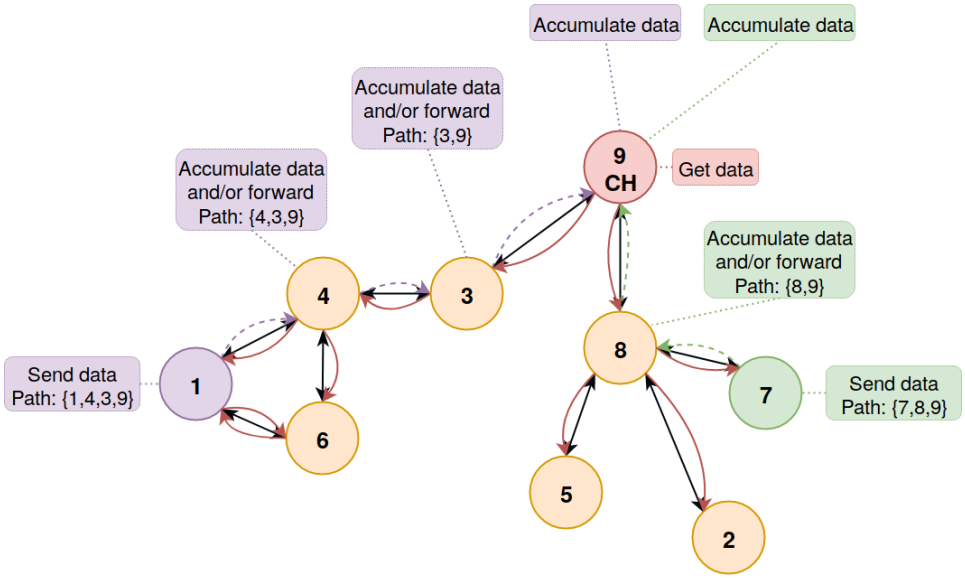


Figure 6.5: Figure show how CH gossips a GET-data request and how nodes sends their data to the leader through the nodes in the path.

6.3 Data Accumulation

A CH will start gathering data by gossiping a message to 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 it's 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. 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.

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 CH 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 arrows show connection between the new node and reachable nodes.

1. https://golang.org/

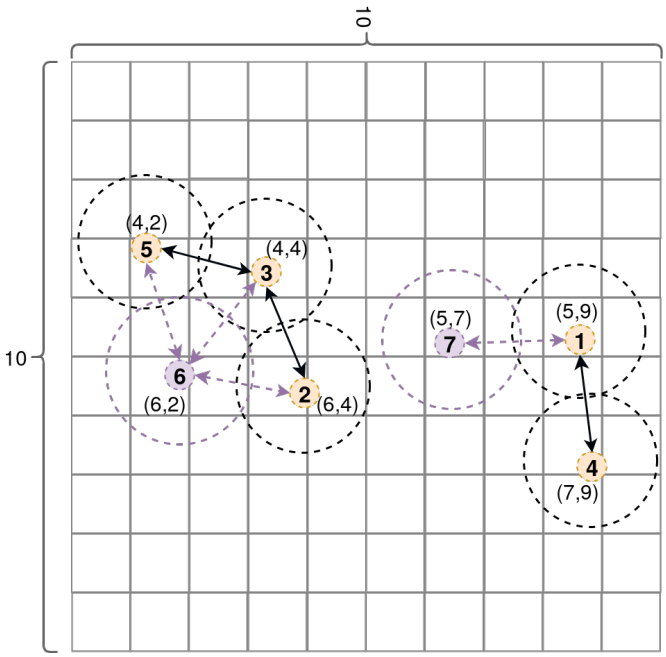


Figure 7.1: Figure show how the network grid size in scale 10 x 10 and nodes broadcast width.

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 shows how a node contacts. 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.

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 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 previous.

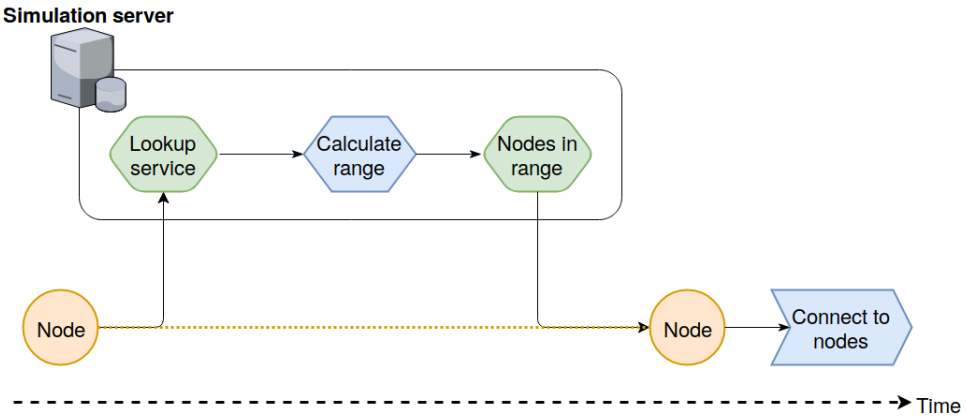


Figure 7.2: Broadcast simulation

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 aCH 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 leader.

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

Why 6??

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 ...**

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 are not safe for concurrent use. If a map is read from and written to from concurrent goroutines, the access must be **synchronized**/protected by a synchronization mechanism. 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 }
```

7.6 Concurrent CH Election And Data Accumulation

To avoid having a CH election occur concurrently as 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. Then will the CH broadcast a request for gathering data and nodes will accumulate and send data to the CH. After gathering data according to the accumulation interval, will the CH start 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.

Our approach to a solution to this interference is to have the two functionalies divided into different phases similar to 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 9.7.

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

The testing environment in our system is built with gopsutil, a psutil for Golang [22]. We use gopsutil 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, described in the section 8.2, where one process is executing one node which is simulating an OU. In order to determine how well the nodes perform in terms of memory, CPU and network usage several experiment scenarios were designed and performed, revealing some of the system's capabilities and limitations. It is important that the cost of running the system is kept to a minimum since it's aimed to execute on small, low-power OUs.

The experiments are ran with a 500 millisecond's measurement interval and system execute around 15 minutes. The experiments are done executing 100 nodes with a broadcast range at 50, as shown in Table 8.1. 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 CHs.

Each node execute their own experiments when launched. Each time interval (500 millisecond) also contains an average value which is displayed in the graphs.

A CH accumulates data after a given interval and the experience are ran on 4 different accumulation intervals as listed below:

- After 2 accumulations
- After 4 accumulations
- After 6 accumulations
- After a random accumulation between 1 and 6.

8.2.1 Memory Measurements

We measure the memory utilities to examine how much memory is consumed when executing the program. The memory utilities measured, is the total per-

centage of RAM used by the program, and not for individual nodes/processes. Measuring the memory is essential because it needs to be relatively low to be able to make the device function as normal in a real-life environment/scenario.

- **Psutil - VirtualMemoryStat - UsedPercent:** percentage of RAM used by programs.

8.2.2 CPU Measurements

We measure the total CPU usage for the computers 4 cores combined to examine how much computational power is needed to execute the program. This means that the CPU usage is similar for all nodes/processes in the experiment. This as well, is important to keep at a minimum due to execution on devices in real-life environment to avoid draining the battery.

- **Psutil - Percent:** calculates the percentage of CPU used either per CPU or combined.

8.2.3 Network (Socket) Usage

Measuring the process connection is important since the system relies on network usage and communication between nodes and it is important to keep it to a minimum. 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.

- **Psutil - ConnectionsPid:** Return a list of network connections opened by a process.

8.2.4 Number Of Sends To Neighbours

We measure number of sends to neighbours to observe the average of sends between nodes in the system during execution. The number of sends are counted every time the node sends a request to a neighbour, like discovering other nodes, CH election or sending and accumulating data to CH.

8.2.5 Number Of Sends To Cluster Head

Besides measuring number of sends to neighbours, we measured 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.

8.2.6 Cluster Head Count

We also measure how many times nodes have been CHs at each time interval to examine if how many CH there is at each time interval.

8.2.7 Cluster Head Receives Packages

At last we measure how many times a CH receives data from other nodes in the network. This is measured only at each CH

8.3 Results

In this section we will discuss the results of the testing described in the sections above.

8.3.1 Memory Usage

8.3.2 CPU Usage

8.3.3 Network (Socket) Usage

8.3.4 Number Of Sends To Neighbours

8.3.5 Number Of Sends To Cluster Head

8.3.6 Cluster Head Count

8.3.7 Cluster Head Receives Packages

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, would forward the request to the 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/**there is any response**. 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.

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 probably forward the accumulated data in the next gathering phase.

Otto: expand last sentence.. "probably"

9.2 Cluster Head Election

!!! A prior?? explain..

Otto: how much more?

Being a CH is much 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, e.g. by a BS without any knowledge about the network, 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 prior/**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 Ø and 1. This **intellectually** simplified does not take in account many other realistic aspects which would improve the CH election. To improve **HOW?????** this approach and make it more realistic, 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
- Er det bare en function-endring eller er det også spørsmål om du har nødvendig info?

9.2.2 Gossip Information Between Nodes

!! Kommer kanskje an på hvordan vi gjør gossip

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

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

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 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.

Improvements?? JM: en mulig sak hva skjer hvis CH fpr X mndinger samtidig som en eller fler andre vil sende? er dert ch som initere query bestandig eller kan andre bestemme seg for å sende?

9.3 Remember Previous Cluster Head

9.4 Multiple Cluster Heads

The current approach have only 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 leaders 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. Most likely **will the?????** CH election 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/have the same "...", the node should **probably - based on what do you propose this?** 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 needed/necessary.

Multiple leaders - replication of data?

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 therefor 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.

9.6 Data Accumulation

9.6.1 Prioritization Of Data?

9.7 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.

Forklar litt mer om fasene i LEACH og mine, fra Section 7.6...

9.8 Base Station Access

9.9 Scalability?

Note to self: scale with partially centralized network.. how about multiple CH to load balance work and maybe shorter path for nodes to leaders..

9.10 CAP Theorem?

Note to self: evt bygge det litt inn i teksten..

10

Conclusion

In this thesis, we have implemented a system/prototype...

Our experiments showed that the system ...

11

Future Work

- CH election/calculation, multiple leaders to load balance and replication(?)
- Access to BS
- Multiple Super OUs that accumulates data..
- Approach for nodes observing other nodes..

12

Appendix

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`.