UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Non-Opportunistic Data Transfer for IoT and Cyber-Physical Systems with Mostly Sleeping Nodes

Isak Østrem Hellemo

INF-3981 Master's Thesis in Computer Science
July 2022

Abstract

Sensor networks are frequently used to monitor our environment. From monitoring the habitat of seabirds [1], to the structural integrity of bridges [2]. They can also be used to monitor the arctic tundra to help us monitor climate change.

The arctic tundra does however place additional requirements on a monitoring system. Low access to energy sources, human intervention, and networks to transfer the results back, combined with a high likelihood of being destroyed by the environment makes it difficult to successfully retrieve any measurements. The nodes should therefore replicate any measurements among themselves while minimizing the energy consumption.

In this thesis, we describe four approaches to schedule connections to share data between a neighborhood of nodes. We also present the implementation of a simulation to evaluate the approaches based on energy usage, broadcast-latency and broadcast-throughput.

We conclude that scheduling connections in a ring-like or cluster structure has in general the lowest energy usage at the cost of latency and throughput. However, more work should be done to get a more accurate estimation of the energy usage of the systems.

Acknowledgements

I would like to thank my main advisor Professor Otto Anshus, and my co-advisor Associate Professor Issam Raïs for their valuable guidance and feedback.

Thank you to the Distributed Arctic Observatory (DAO) project funded by theResearch Council of Norway, Project number 27067.

I would also like to thank my family and friends for their support and encouragement along the way.

Contents

List of Figures

List of Tables

1 Introduction

Sensor networks are frequently used to monitor our environment. They have been used to monitor seabird nesting environments [1], the impact of cooling solutions in data centers [3], structural health [4] [2], and urban air quality [5]. These systems could also be used to monitor the arctic tundra, gathering data used in climate change models. However, the arctic tundra places extra requirements on a potential monitoring system.

The arctic tundra is a difficult place to deploy sensor nodes. The nodes must be deployed in remote locations where there are no available power sources, and no possibility of human intervention. The nodes must therefore spend long periods of time sleeping to save power. Additionally, the nodes may easily be destroyed by animals, snow, or other events such as flooding. As such, they should also replicate measurements among themselves to increase the likelihood of data being successfully retrieved despite nodes being destroyed.

In this thesis we describe four general non-opportunistic approaches to share data between nodes in a neighborhood by scheduling connections. We implement a simulation to evaluate the four approaches by exploring their energy usage, latency, and throughput. Finally we will discuss the results' implications for choosing an approach for the arctic tundra.

2 Related work

There has been a lot of research on the efficient data dissemination between nodes. Both in terms of minimizing latency, and to reduce the overall energy consumption.

In [6] we looked at the effects of having the nodes opportunistically communicate whenever two nodes were awake at the same time. We found that this caused a high latency for data dissemination when the nodes were mostly sleeping as there was a low likelihood of two nodes being awake at the same time. The latency could be improved by scheduling all the nodes to wake up at the same time each day. However, this caused collisions where a single node was contacted by multiple neighbors simultaneously, which wasted both time and energy. In this thesis, we extend this work by looking at the effects of creating a schedule of when the connections take place.

In [7] Niki Trigoni et al. introduces WaveScheduling, a methodology for trading energy vs latency in a sensor network. The idea is to schedule message transmissions to avoid collisions in the MAC layer. Additionally, the schedule allow nodes to turn off their radio in periods where they are not scheduled to transmit data. They do this by creating a schedule where the node wake up in a wave-like pattern. This lets the nodes sleep for long periods of time, while also minimizing the latency of messages, as the message can "follow the wave". In our work, we assume a radio-technology where there is no interference if multiple nearby nodes transmit simultaneously. Additionally, we focus on transmitting data to all nodes in a neighborhood, where every node is within radio distance of each other.

In [8] Nikolaos A. Pantazis et al. present a Time Division Multiple Access (TDMA) scheduling scheme which achieves high levels of power conservation while reducing the end-to-end transmission time from sensors to a gateway. They do this by creating multi-hop paths from each node to the gateway. Then they create a schedule such that each node is assigned a time-slot at an earlier point than the next node in the path to the gateway. Additionally, instead of instantly sending the content, they first send a wakeup message which tells all the nodes in the path to the gateway to stay active until the data is transmitted. In our work, we look at how to schedule node wakeups to efficiently transmit data to all nodes in a neighborhood.

In [9], H. Sabbineni et al. presents location-aided flooding, an energy efficient data dissemination protocol which uses location information to reduce the number of redundant transmissions. They prevent redundant transmissions by sending the ids of all nodes which has already received the message in a message-header. They also prevent the header becoming too large by only storing the ids of nodes in the current grid. In our work, we do not use any headers to prevent redundant transmissions. Instead we schedule connections such that there is always just one node responsible for sending a message to a receiver, avoiding redundant transmissions.

In [10] Aasem Ahmad et al. presents a distributed TDMA scheduling algorithm for *ZigBEE*-like cluster-tree topologies to meet timeliness and energy demands. They create a tree of clusters, where each cluster is scheduled to be active in an order which allows messages to be passed from a source to a sink node within an end-to-end deadline. In this work, we explore how different topologies will perform in terms of latency and throughput to evaluate potential alternatives to using a cluster topology.

In [11] Michael J. Murphy et al. designed and deployed CO_2 sensors in the arctic tundra. They described the architecture and implementation of the system, and described lessons learned from the deployment. The nodes were deployed in an area with good LTE-M coverage, which allowed the nodes to transfer the measurements back from the tundra. The nodes would wake up daily to take measurements and send the results back. In our work, we do not consider how and when to transfer the data back from the tundra. Instead, we explore how the nodes can replicate the data between themselves to avoid losing data when nodes are destroyed. We also do not rely on the nodes being deployed in an area with a reliable back-haul network, as the nodes will only communicate with each other to replicate the data.

In [12] Issam Raïs et al. report the tradeoffs between successful data dissemination, and energy and uptime overheads resulting from loosely coupled policies. These policies include extending the uptime of a receiver to complete a data transfer, and receivers sharing hints of when the sender will be awake. In our work the nodes do not wake up randomly. Instead, we create fixed schedules of when the different nodes should communicate.

3 System requirements

The arctic tundra is a difficult environment to monitor. The observation nodes must be placed in a remote and harsh environment, where there are limited possibilities for energy generation and human intervention, and a high likelihood that nodes will be damaged or destroyed.

This section describes the requirements for our system, and the assumptions made regarding what technologies will be available.

3.1 Long deployments with limited energy

We expect the nodes to be deployed for long periods of time, with little to no possibility to generate energy while deployed.

The arctic tundra is remote, and it requires a lot of time and resources for humans to travel and deploy the nodes. Current observations of the arctic tundra are done through seasonal expeditions where they collect old and deploy new equipment such as camera traps [11]. As such, the nodes should survive at least one year without human intervention, and preferably longer.

However, there are limited options for energy generation. The nodes will be deployed in different locations depending on what they are measuring. This includes being under snow, rocks, or next to cliffs. These locations combined with bad weather and little sun during the winter makes the use of solar-panels difficult. Additionally, there are regulations which may prevent the deployment of larger installations such as wind-turbines [11]. Because of this, the nodes must survive on battery-power alone the whole deployment, which can only be achieved by aggressively saving energy where possible.

3.2 Data-replication between nodes

There is a high likelihood that nodes will be destroyed at some point during a deployment. Nodes may be moved or destroyed by avalanches, or animals playing with the equipment. They may also be filled with water from the melting snow. Measurements made by the node should therefore be replicated to other nodes to increase the likelihood of successfully retrieving the data.

Some nodes may also have intermittent access to limited back-haul networks which can be used to transmit data back for analysis. For instance, one might fly a drone near the nodes to download the data, or skiers might have an app installed on their phone which downloads data from any node within connection range. However, the drone or app will most likely only discover a small subset of the nodes, which means that the measurements should be replicated at multiple nodes to increase the likelihood that it is successfully transmitted.

3.2.1 Types of data dissemination

Nodes will take measurements at varying frequencies, and of varying data-sizes. For instance, camera traps might have long periods where there are no animals nearby, before finally taking a large picture which should be replicated. Other nodes might regularly take small CO2 measurements [11] to observe how the CO2 concentration in the air changes. This will cause the amount of data to be replicated at a given moment to vary. We group the replication into three types.

- No replication: There will be periods where nodes are not making any observations which need replication. This will either be because it is saving energy for later, or because there is nothing to measure at the time.

- Small and regular replication: Nodes may regularly make small observations which should be replicated. For instance, it may do daily measurements to see how something changes over a long period of time.

- Bursty replication: Nodes may make several large observations in a short period of time. For instance there might suddenly be a lot of animals in the area which will cause the camera-trap to take multiple large pictures which should be replicated.

Depending on what kind of sensors are being deployed, the system will have different replication needs. The nodes must be energy efficient when there is little to no data being replicated However, it should be able to replicate a large chunk of data within a reasonable time. This is needed because the replication must complete before the next burst to stop the data replication backlog from becoming too large. Additionally, the data should be replicated before the node is destroyed.

3.2.2 Recovering from node failures

As mentioned, there is a high probability of nodes being destroyed or running out of energy. They may also temporarily be covered in snow or other debris, blocking any radio signals. Any data dissemination protocol must therefore handle nodes becoming temporarily or permanently unavailable. In this thesis, we will not explore how to detect, prevent, or recover from such failures, however this must be considered before deploying sensors on the arctic tundra.

3.3 Communication technologies

There already exist several technologies which can be used for communication between sensor nodes. These include *LoRa* [13], *ZigBEE* [14], *BLE* [15] and several others [13]. Each come with different tradeoffs regarding energy consumption, bandwidth and latency.

These technologies will improve, and we expect new technologies to be developed which have even lower energy-costs. Therefore, we will not assume any specific technology in this work. Instead, we will make some assumptions about what future technologies will allow us to do. The assumptions are as follows:

- Nodes can form one-to-one connections with each other, where each node can only have one active connection at time.

- There is little to no radio-interference between connections, even if the nodes are near each other.

4 Spreading abstraction

To fulfill the requirements of our system, we introduce a spreading abstraction which describes how nodes are allowed to spread data. The abstraction defines two terms. A grouping of related nodes called a neighborhood, and an operation called a *broadcast*.

The neighborhood is defined as a set of nodes where every node is within the connection range of every other node. A node is only part of a single neighborhood at a time. Additionally, nodes will not leave or join neighborhoods dynamically. Instead, the neighborhood is defined before they are deployed. If a node is destroyed, or runs out of battery, we still consider it to be part of the neighborhood.

When in a neighborhood, a node is allowed to replicate its data by doing a *broadcast*. A *broadcast* consists of sending a chunk of data (one or more measurements) to all nodes in the neighborhood. We assume that our radio technology only permits one-to-one connections. A *broadcast* in this setting will therefore require nodes to form multiple connections with different neighbors, or forward data on behalf of other nodes.

5 Topologies

We propose four approaches to implement the spreading abstraction using a Time Division Multiple Access (TDMA) based scheme where we schedule connections between nodes.

The idea behind TDMA is to split time into equal-sized time-slots which are grouped into frames. Nodes are assigned to sleep or wake up to connect to a neighbor at specific slots within each frame to avoid transmission-collisions between nodes in a neighborhood.

In our case, we assume that a node can only connect with a single neighbor at a time. We therefore create a connection schedule where there is a maximum of one connection per node per slot. However, we may assign multiple unrelated connections in the same slot. The nodes will wake up for the slots they are assigned a connection, but they will only initiate the connection if they have data to replicate. The nodes will also stay awake for a short period at the start of the slot to allow the neighbor to potentially initiate the connection.

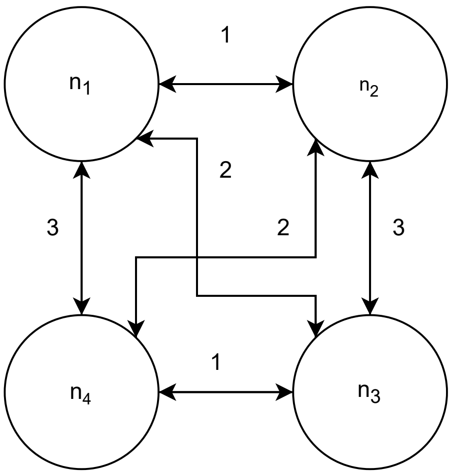


Figure 5.1: Connection schedule for four nodes with a fully connected structure. The number next to each line corresponds to the time-slot for the connection

5.1 Connection schedules

5.1.1 Fully connected

The first approach is to schedule connections between every pair of neighbors. Figure 5.1 shows a neighborhood of four nodes. Each arrow corresponds to a connection between the nodes, and each number corresponds to the time-slot they are assigned to connect. Note that there are multiple parallel connections during the same slots. For instance we see that n_1 and n_2 has a scheduled connection in the same slot as n_3 and n_4

With this scheme, a node will only transfer its own data to neighbors, which means that each node is responsible for completing its own *broadcasts*.

The nodes could forward data on behalf of other nodes. However, this will cause redundant data transfers between nodes unless the nodes send some form of control messages. For instance, in Figure 5.1, if n_1 is performing a *broadcast*, n_2 could in theory forward the message to n_4 during slot 2 reducing the time required for the *broadcast* to complete. However, n_1 has no way of knowing that this occurred, or whether it succeeded. As such n_1 would have to send some form of control message checking what data n_4 has received, which costs energy. In the arctic tundra, energy preservation is generally more important than speed, which makes this approach unfeasible.

There are several approaches to compute a fully connected schedule. One approach is to maximize the number of parallel connections, as this minimizes the time to complete a *broadcast*. However, calculating the optimal schedule for this is an NP-complete problem [16]. Another extreme is to only allow for a single connection in each slot.

Instead of calculating the schedule with the maximum number of parallel connections, we used the algorithm shown in Figure 5.2.

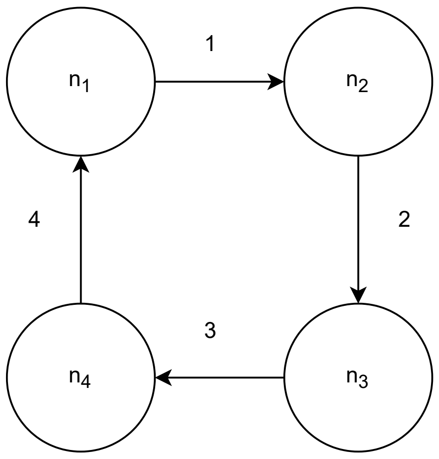
```
func GenerateFullyConnectedSchedule() {

  // For each node in the neighborhood
  for node := 0; node < numNodes; node++ {

    // For each remaining neighbor
    for neighbor := node + 1; neighbor < numNodes; neighbor++ {

      // Check every possible slot
      for slot := 0; ; slot++ {

        // Skip if either node has already used the slot
        if slotUsed(node, slot) || slotUsed(neighbor, slot) {
          continue
        }
        // Assign connection to free slot
        assignConnection(node, neighbor, slot)
        break
      }
    }
  }
}
```

Figure 5.2: Algorithm for generating a fully connected schedule

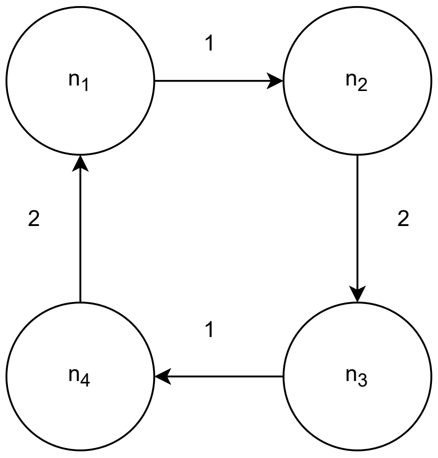
We have not analyzed exactly how much parallelization this algorithm provides. However, when generating schedules for up to 100 nodes, it generated schedules with a maximum of $(N-1) \cdot 2 - 1$ slots where N is the number of nodes.

5.1.2 Ring

The second and third approach uses a ring-structure as shown in Figure 5.3. Each node is placed in a ring where they only form connections with the next and previous node in the ring. With this approach the nodes will both send its own data, but also any data it receives from the previous node in the ring. Using this structure, we achieve a *broadcast* by forwarding the data through



(a) Ring with no parallel connections



(b) Ring with maximum number of parallel connections

Figure 5.3: Connection schedule of four nodes with a ring structure. The number next to each line corresponds to the time-slot for the connection. The arrows point at the direction of the data-flow

the entire ring. The nodes will only transfer data forward through the ring tracking what data has already been sent. This avoids redundant transfers without sending any control messages.

As with the fully connected approach, there are several possible connection-schedules which can follow the ring-structure depending on how many parallel connections we want. We will look at two cases. The case with no parallel connections as shown in Figure 5.3a, and with the maximum number of parallel connections as shown in Figure 5.3b.

In the case of no parallel connections, we see that we require N slots for all the connections. With parallel connections, we need either two or three slots depending on whether there are an odd or even number of nodes. In figure 5.3b we have an even number of nodes, requiring two slots. However, when there are an odd number of nodes, n_1 will be assigned to communicate with both its next and previous on slot 1, which is not allowed. As such, we require a third slot in each frame.

5.1.3 Cluster

The fourth approach uses a cluster-structure shown in Figure 5.4, where all the nodes communicate with a single master. The master is responsible for receiving and *broadcasting* messages from the other nodes in the neighborhood. The

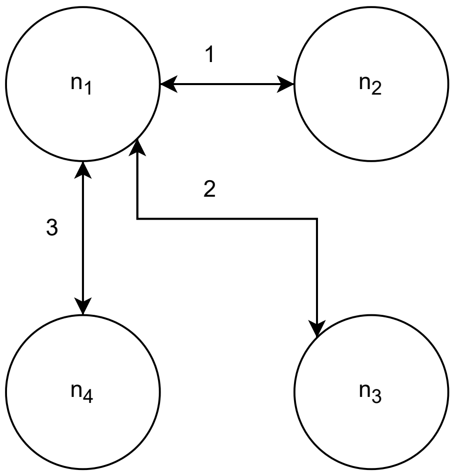


Figure 5.4: Connection schedule for four nodes with a cluster structure, The number next to each line corresponds to the time-slot for the connection

master acts as a normal node, taking measurements and initiating *broadcasts* as any other node. However, it has the additional responsibility of forwarding data and completing *broadcasts* on behalf of its neighbors. This scheme does not allow for any parallel connections, as the master node can only connect to one node at a time. As such, this scheme requires $N - 1$ slots.

6 Methodology

There are several ways to evaluate our four approaches. These include simulating, emulating, or creating a prototype of the systems.

Simulation is described in [17] as the imitation of a real-world process or system which can be studied and used to draw inferences to how the real-world system will behave. This is done by creating a *model* where we make assumptions about how the system operates. Simulations are generally easier to implement that the real world equivalent, which makes it suitable as an analysis tool when designing the actual system since we can more easily test how changes will impact the system performance.

Emulation, and its advantages and disadvantages compared to simulation has been discussed in [18] by Ian McGregor. He describes emulation as being similar to simulation, but where parts of the model is carried out by a real system. He notes that there will always be a difference between the results of a simulation, and the real world equivalent. This difference can be reduced using emulation models, where parts of the model is based on real systems. However, this forces the system to be emulated in real time, as it is based on a real system making it harder and slower to run multiple tests with small changes. Simulations on the other hand can make assumptions and simplifications which allows it to fast forward through time, making it easier to run multiple tests. As such, emulation is more suited to validate the characteristics of the system as it is slower but generally more accurate.

A final possibility is to implement a prototype. However, this is costly and takes long to implement. It also forces you to make choices of which hardware and software to use, which will have a measurable impact on the characteristics of the system.

In our case we do not have the resources to create a prototype of the sensor nodes. Similarly, as we have multiple approaches to test in multiple scenarios, it is unfeasible to create an accurate emulation of the system. As such, we have chosen to evaluate the system by creating a simulation for the different topologies.

We had multiple choices regarding how to simulate the system. There is a large number of simulation frameworks out there such as SimGrid [19], NS-3 and OMNet++[20], each with their own pros and cons. There is also a learning curve to use each of the frameworks as they are usually designed to create detailed statistics of a wide variety of systems. It would therefore take a lot of resources to find the correct framework and learn how to use it. Instead, we chose to implement out own simulator for our specific needs.

7 Simulation

To evaluate the system we implemented a custom simulation for the different connection schedules. The simulation takes a topology and the number of nodes, and computes how often each nodes perform different actions. Specifically, the simulator tracks:

- Time spent sending data
- Time spent listening for and receiving data
- Total amount of data sent
- Number of slots used
- How much simulated time has passed

The basic idea of the simulation, is that it is given a list of all the scheduled connections and their corresponding slots within a frame based on the connection schedule for the topology. It then iterates over each scheduled connection, and simulates the actions taken by the relevant nodes in that slot. The simulation consists of two parts. A schedule generator, and a simulation runner.

7.1 Schedule generation

The schedules are created as a list of all connections which should take place within a frame. They are generated based on the topologies described in chapter 5, and remain static for the entire simulation. The algorithms are deterministic, and will create the same schedule for a given amount of nodes every time.

7.2 Simulation runner

The simulation runner uses the generated schedule, and simulates the node activities. This is done by iterating over the schedule, and simulating the actions taken by each nodes during each slot. This includes activities such as waking up according to the schedule, listening for the potential connection, and performing the data transfer.

7.2.1 Node representation

Each node is represented as an object containing the relevant state for , and the statistics for the activities performed by the node. The state includes:

- Data stored at the node
- Data sent to other nodes
- Statistics including:
  - Time spent sending data
  - Time spent listening for and receiving data

7.2.2 Data representation

The data is represented as a set of *objects*. Each object is given a size, and an identifier in the form of a source-node, and a per-node-unique index. Each object corresponds to a chunk of data which can be replicated between nodes. This might represent a measurement like the air humidity. We assume that an *object* can not be split up, but must be transferred as a whole to neighbors.

7.2.3 Simulated connection

When simulating a connection, we do the following. The internal state of the two nodes are checked to see if there is any data to transmit. This would for instance be the case if one of the nodes are currently performing a *broadcast*, and the current neighbor has not yet received said *broadcast*. We also check the nodes for what data it has already sent, to prevent redundant data transfers.

If neither node has any data to transmit, we simulate that the nodes wake up, wait a set amount of time for the neighbor to initiate the connection (which in this case does not happen), and fall asleep again. We then increment the listen-time counter accordingly.

If one or more of the nodes has data to transmit, we instead simulate that the nodes wake up, connect, and transfer data. We assume that there is no form of radio interference, and that the nodes are able to fully utilize the bandwidth of the connection between them. We also assume that the nodes connect instantly, allowing them to utilize the whole slot. We can therefore calculate exactly how much time is required to transfer the data between the nodes. If both nodes have data to send, the bandwidth is shared equally, allowing each to send as much as they receive. After calculating how much is sent by each node, we increment the transfer- and listen-time counters correspondingly.

As mentioned, the *objects* can not be split into smaller chunks for transfer. This means that we only simulate transfers of entire *objects*. The nodes might therefore go to sleep before their slot is over even though they have more data to send if there is not enough remaining time in the slot to transfer a whole *object*. We do however assume that the slots are long enough to transmit at least one *object* per slot.

7.2.4 Recording the results

When the simulation is complete, the state of each node is recorded for later analysis. The total amount of data broadcast, the number of slot used, and the amount of simulated time which has passed is also stored.

8 Evaluation

8.1 Metrics

The different approaches are evaluated using the following metrics.

- Energy used by all nodes while idle for 24 hours. The amount of energy used by the nodes during 24 hours while no *broadcasts* are performed. This includes the total amount of energy used, and the energy used by the node with the highest energy consumption.

- Energy used during one *broadcast* of 100mb. This includes the total amount of energy used, and the energy used by the node with the highest energy consumption.

- Energy used when all nodes *broadcast* 100mb. This includes the total amount of energy used, and the energy used by the node with the highest energy consumption.

- Latency for *broadcasts* of one byte. This is the time passed from when the *broadcast* is started until it is completed.

- Latency for *broadcasts* of 100mb. This is the time passed from the first *broadcast* is started until all nodes has successfully completed their *broadcast*.

- Throughput for a single node. The number of bytes *broadcast* per second by a single node while all other nodes are silent.
- Throughput for all nodes. The number of bytes *broadcast* per second when all nodes are *broadcasting* simultaneously.

8.2 Experiments

To evaluate the topologies, we perform five experiments for each topology. These consisted of running the simulation for:

- 24 hours with no *broadcasts*.
- A single *broadcast* of 1 byte.
- Every node performing a *broadcast* of 1 byte.
- A single *broadcast* of 100mb.
- Every node performing a *broadcast* of 100mb.

8.2.1 Experiment parameters

The simulations do not consider any specific technologies. Only that there is some way to transfer data between the nodes. However, during the simulations we use the network bandwidth and energy usage from related literature.

Table 8.1 shows the parameters used in the simulations. As in [12], we consider Raspberry Pi Zeros which communicate over *LoRa*, where we simulate the energy consumption of sending and receiving data as being equal.

A slot length of 10 minutes was chosen for the following reasons. It is a long slot which allows the nodes to transfer up to 30mb of data during a single connection. Additionally, it lets the nodes sleep for long periods of time before having the next scheduled connection. However, it also short enough that a whole frame will complete within one day even if the neighborhood grows to hundreds of nodes.

The nodes with scheduled connections will also have a listen period of 10 seconds before falling asleep again when there is no data to transmit. We chose 10 seconds as related work has shown that the internal clock of a node may scew up to 10 seconds per day [21].

When estimating the total energy used by a node, we calculate the amount of time spent in the listening- and transferring-states, and their corresponding energy costs.

Bandwidth	50 kbps [12]
Transfer energy	0.56W [12]
Listen energy	0.56W [12]
Listen period	10s
Slot length	10 min

Table 8.1: Simulation parameters

8.3 Results

8.3.1 Energy usage while idle

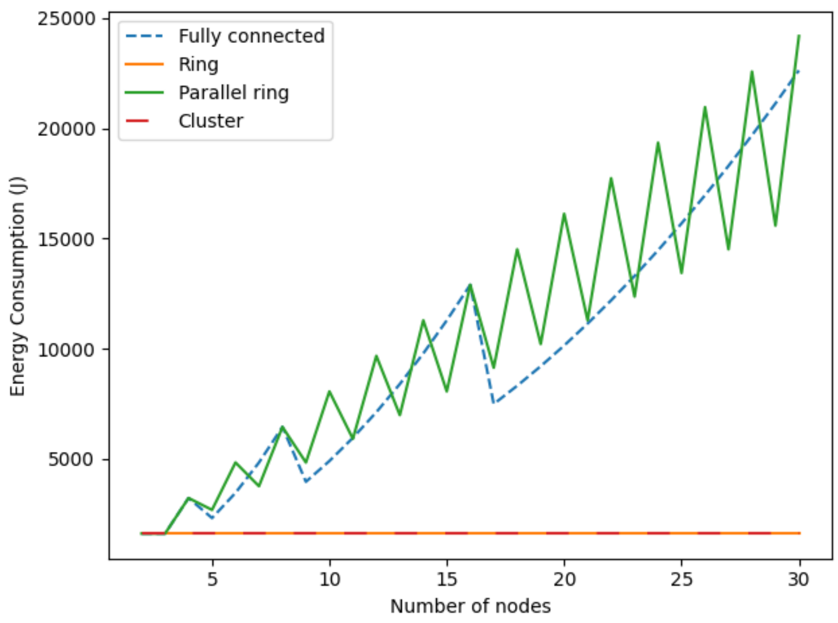


Figure 8.1: Total amount of energy used by all nodes in 24 hours with no *broadcasts*

Figure 8.1 shows the total energy used by the neighborhood when deployed for 24 hours while performing no *broadcasts*. The energy usage in this case only consists of the nodes listening for possible connections at the start of their slots. As such, the energy used corresponds to how many parallel connections are scheduled. We see that the fully connected and the parallel ring topologies has a high energy usage, with the parallel ring being the most expensive as they schedule multiple connections per slot. The cluster and ring topologies on the other hand has a low energy usage, as it only schedules a single connection per slot.

Note that the variation in the parallel ring corresponds to whether there is an odd or even number of nodes in the neighborhood, where an even number of nodes allows for more parallel connections. The fully connected schedule also varies depending on how many parallel connections our algorithm generates.

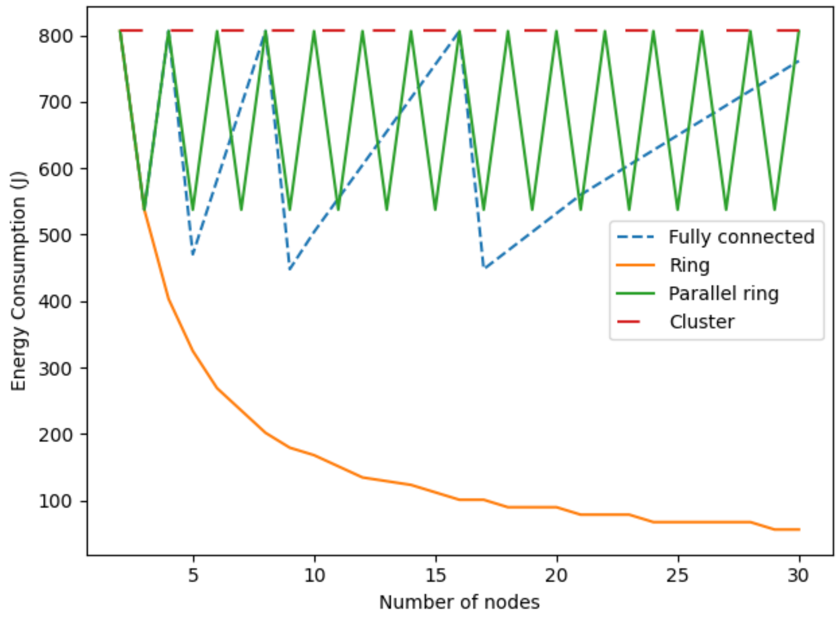


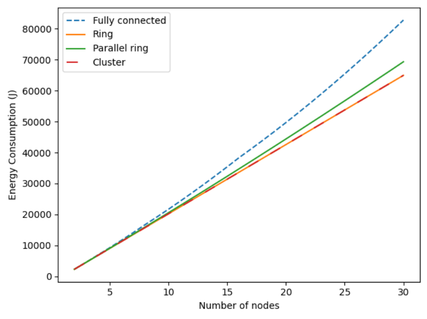
Figure 8.2: Maximum amount of energy used by a node in 24 hours with no *broadcasts*

Figure 8.2 shows the energy used by the node which used the most energy when deployed for 24 hours while no *broadcasts* were performed. This corresponds to the node which has the most scheduled connections per frame, as they spend the most time listening for possible connections.

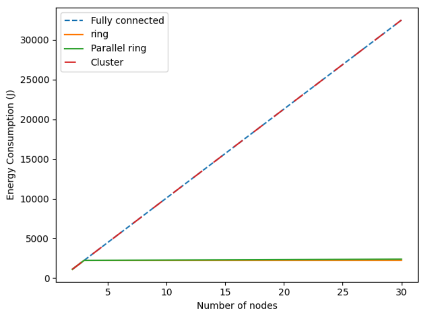
We see that the cluster-topology has the single node which uses the most energy. Specifically the master-node. This is because it must be active for every slot in every frame. We also see that the energy required is constant, no matter how many nodes are in the neighborhood. The parallel ring, and the fully connected topology also has a high maximum energy usage. However, it varies depending on the size of the neighborhood, as some sizes creates schedules where nodes are inactive for some slots.

Finally, the ring-topology uses the least amount of energy during a day. This is because each node only wakes up twice for each frame. Once for the connection with the next, and once for the previous node in the ring. As the neighborhood size increases, the time between each connection for a node increases, which again reduces the amount of energy used by each node.

8.3.2 Energy usage under load



(a) Total amount of energy used by all the nodes



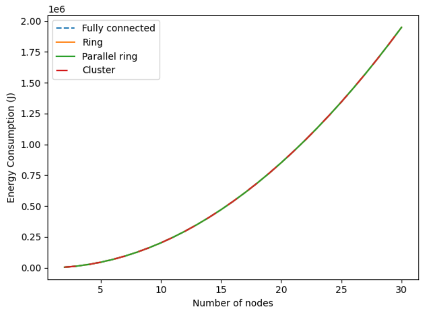
(b) Maximum amount of energy used by a node

Figure 8.3: Energy usage when a single node *broadcasts* 100mb

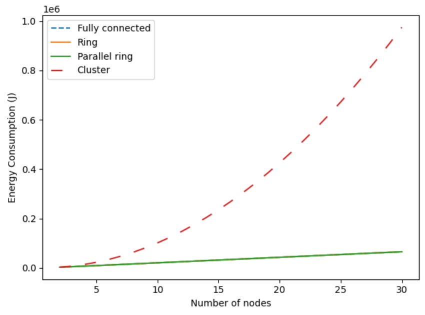
Figure 8.3 shows the energy used when a single node performs a *broadcast* of 100mb. Figure 8.3a shows the total energy used by all the nodes. We see that the schedules use a similar amount of energy. However as the number of nodes increases we see that the fully connected topology uses more energy. This is because nodes will have to wake up for scheduled connections with every neighbor, even though only one of the neighbors are actually performing a *broadcast*.

Figure 8.3b shows the energy used by the node which used the most energy. We see that the fully connected and cluster topologies require a single node (the sender in the fully connected, and the master in the cluster) to do all of the data transfers.

Figure 8.4 shows the energy used when every node performs a *broadcast* of 100mb each. Figure 8.4a shows the total amount of energy used by all of the nodes. We see that all the topologies in total use the same amount of energy since they are sharing the same amount of data with no redundant sends. Figure 8.4b shows the energy used by the node which used the most energy. Here we see that the cluster topology requires the master node to use significantly more energy as it has to perform all the *broadcasts* on behalf of the rest. On the other hand, the other topologies has a much lower, and therefore



(a) Total amount of energy used by all nodes



(b) Maximum amount of energy used by a node

Figure 8.4: Energy usage when every node performs a *broadcast* of 100mb

more evenly distributed per node energy usage.

8.3.3 Broadcast latency

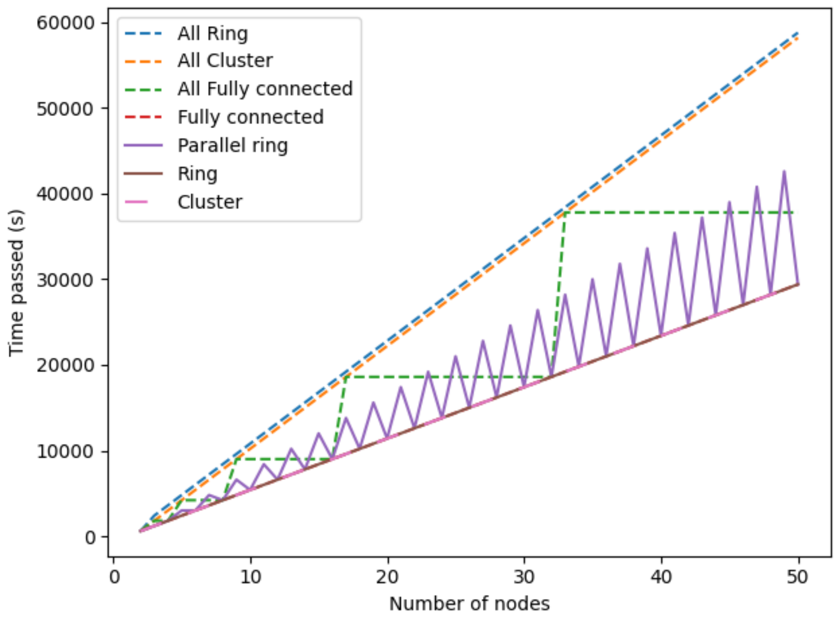


Figure 8.5: Time passed when nodes perform a *broadcast* of 1 byte. *All* refers to the case when every node is performing a *broadcast* each. Otherwise it is only a single *broadcast*

Figure 8.5 shows the time to complete a *broadcast* of 1 byte. The labels starting with *all* is for when every node is performing a *broadcast* simultaneously. The *broadcasts* happen concurrently, where the nodes does not wait for other *broadcast* to complete before starting its own. The time is measured from the when the first *broadcast* begins, until they have all completed. The parallel ring had the same latency for both scenarios.

We see that performing a single *broadcast* is quickest. out of the single *broadcasts*, the cluster, ring, and fully connected approaches takes the same amount of time. The parallel ring however takes longer depending on whether there is an odd or even amount of nodes.

Performing multiple *broadcasts* is generally slower. This is because we require multiple frames for the *broadcast* to complete. For instance, when every node performs a *broadcast* in a ring-topology, the last node in the ring can only start its *broadcast* at the end of the first frame. We can therefore see that the ring and cluster topologies takes twice as much time to complete all the *broadcasts* than their single-*broadcast* alternatives.

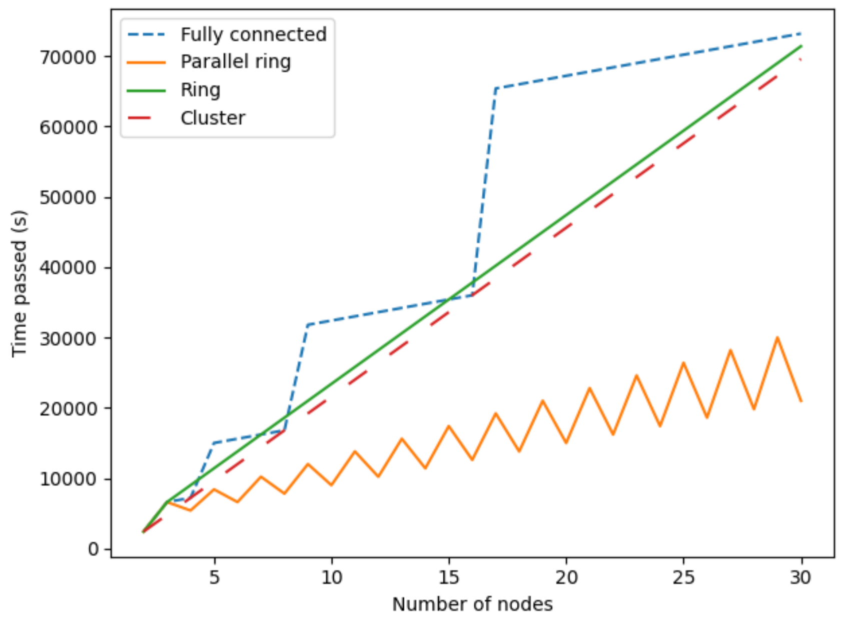


Figure 8.6: Time passed when for a single *broadcast* of 100mb

Figure 8.6 shows the time taken for a single node to *broadcast* 100mb. We see that the parallel ring is the fastest, and the fully connected is generally slowest. We also see that the ring and cluster topologies are very similar, however the ring cluster is slightly quicker.

Figure 8.7 shows the time required for every node to complete a *broadcast* of 100mb. We see that the ring and clustered topologies become significantly slower as we increase the number of neighbors. This is because all the data has

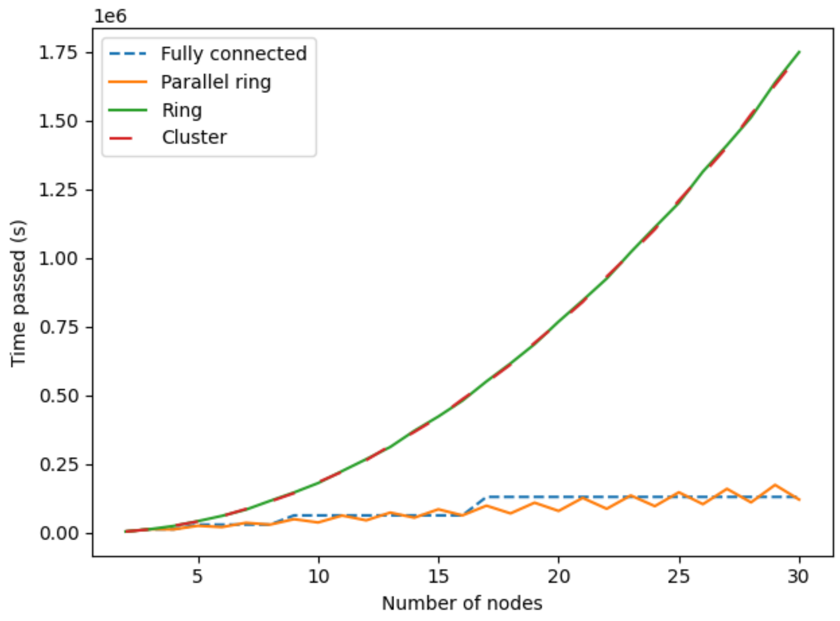


Figure 8.7: Time passed when every node performs a *broadcast* of 100mb

to be passed through a single node.

8.3.4 Throughput

Figure 8.8 shows the amount of data which can be *broadcast* per second by a single node. We see that overall, increasing the number of nodes in the neighborhood will reduce the throughput. This is because it takes more connections per *broadcast*. However, we see in Figure 8.9, that when multiple nodes perform a *broadcast* simultaneously, then the overall throughput does not decrease for the fully connected and parallel ring topologies. This is because they utilize the multiple parallel connections in the schedules. The ring and cluster topologies on the other hand do not have any parallel connections, which reduces the throughput.

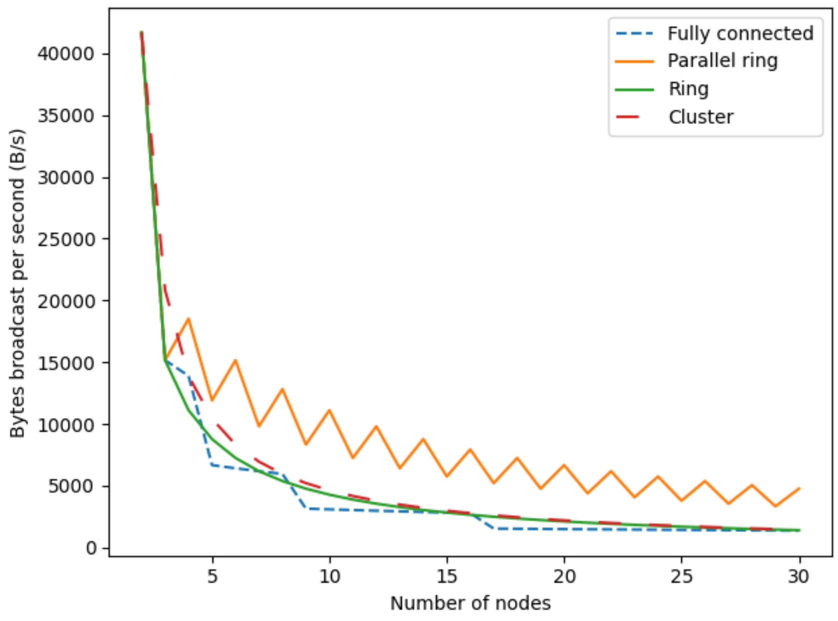


Figure 8.8: The number of bytes/s which can be *broadcast* when only one node is *broadcasting* at a time

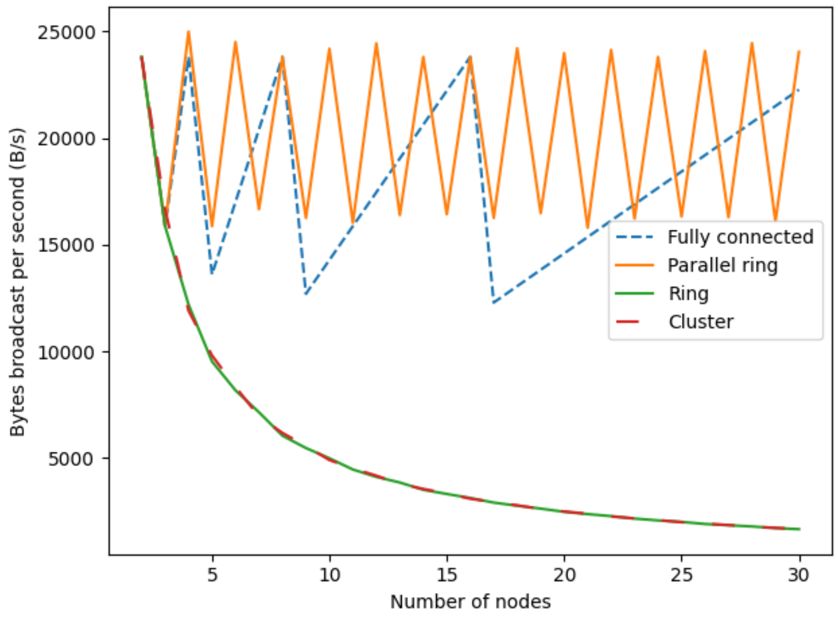


Figure 8.9: The number of bytes/s which can be *broadcast* when all nodes are *broadcasting* simultaneously

9 Discussion

9.1 Implications for spreading systems in the arctic

The results show several aspect which must be considered when deploying a system in the arctic tundra.

First, is the energy usage. This is important as this is the leading factor deciding how long the system can be deployed before requiring human intervention to recharge the nodes. The total amount of energy consumed by all the nodes decides how much energy storage (batteries) must be brought to the field. However, the distribution of the energy consumption is also important. If one node requires significantly more energy than the rest, then it must be given a larger energy-capacity to stay active as long as the rest.

We found that when idle, the ring and cluster topologies overall used the least amount of energy. However, the cluster approach places most of that energy usage on the master node. Similarly, under load, the ring and cluster approach used either the same or less energy as the other topology, but the cluster causes the master node to expend a large amount of energy. This means that when using a cluster approach, we must take into consideration the additional energy consumption of the master. On the other hand, the fully connected and parallel ring approach uses more energy while idle. It should also be noted that the fully connected topology is more *fair*, as in that the node which is initiating the *broadcasts* is the node which expends the most energy. This will in some cases be advantageous, as we can predict which node requires more energy depending on the frequency of the measurements.

The second factor is the latency and throughput of the system. If nodes are making frequent and large measurements which are *broadcast*, the system must have a high enough throughput to transfer all of the data without creating a large backlog. Additionally, if the data is important, we want a low latency to reduce the likelihood of data being lost if a node is destroyed.

We found that when only a single node is performing *broadcasts*, the different topologies has similar characteristics, with the parallel ring having slightly lower latency and higher throughput. However, when multiple nodes are *broadcasting*, the fully connected and parallel ring has the lowest latency and highest throughput since they allow for multiple connections in a single slot. Because of this, if multiple nodes are taking large measurements of high importance, with high frequency, then a fully connected or parallel ring approach should be considered.

In the arctic tundra, the most important aspect is achieving low energy usage. Based on these conditions, the ring topology has several advantages. It uses the least amount of energy while idle, and does not put extra strain on specific nodes in the neighborhood during *broadcasts*. All the while having similar latency and throughput as the cluster approach. However, if we know that there will be a significant amount of data that will be replicated continually, then the more energy-consuming fully connected and parallel ring alternatives should be considered.

9.2 Limitations of simulation

There are several limitations with how the nodes are simulated. The simulation assumes an ideal scenario where the nodes always wake up at the correct time and successfully form a connection with its neighbor instantly. This will in practice be impossible. The nodes will for instance be affected by clock-scew[21] which will cause the nodes to wake up sooner or later than expected, and therefore have to wait for the connection.

We also assume that the bandwidth of the connections are always optimal. However, in practice, the quality of the connection will vary due to several factors. Weather such as rain and snow may block the radio signals, and interference from other radio transmitters may reduce the bandwidth of the connection.

The simulation also does not account for time and energy used by the nodes for start-up and shutdown. Depending on the software/hardware on the node this will vary, however most systems require some form of boot sequence to initialize the hardware, which takes both time and energy.

When exchanging data, the nodes always shared the bandwidth with a perfect 50/50 split (except if a node had no data to send). In practice, this would be hard to do without also sending some form of control-messages which would take time to send and receive.

9.3 Energy estimation

The energy estimation is rudimentary, and does not give a full picture of the actual energy usage of the nodes. In this work we consider the time spent listening and transmitting data. However, there are several other factors which will affect a nodes energy usage. For instance, processing the data received, and tracking what data is already sent uses extra energy. Additionally, the frequent start-ups and shutdowns of nodes will require extra energy usage. We can therefore not use these results directly to estimate how much energy we need to deploy a sensor network. However we can still use it to compare the energy usage between the different possible approaches.

10 Future work

10.1 Measure energy usage

The current work estimated the energy usage based on measurements of certain technologies in related literature. Specifically an idle raspberry pi Zero [12] [22], and the energy-use of *LoRa* [23]. In the future we should create a prototype of the system and measure the actual energy usage as this will be much more accurate than our current simulations.

10.2 Prioritizing broadcasts

Currently, all data *broadcast* is treated identically. This means that there is no way to prioritize important, or urgent data. The system should be extended to allow setting a priority for the data, allowing more important *broadcasts* complete quicker.

10.3 Replication beyond neighborhood

Currently, we only considered *broadcasts* to replicate data within a single neighborhood. However, natural disasters may destroy a whole area. This means that the data should be replicated physically further away from the source node. We should therefore also explore how to replicate the data outside a nodes neighborhood. For instance, we may specify that some nodes must act as bridges between nearby neighborhoods. This will require the connection-schedules to be modified so that the bridge-nodes has available slots to communicate.

10.4 Tiered clusters

The clustered approach currently has a single master node which has the highest energy-requirements and ends up being a bottleneck when there is a high number of *broadcasts*. This could potentially be improved by creating a form of tiered clusters, where multiple nodes share the role of master.

10.5 Dynamically reconfigure schedules

We have currently not explored how the nodes might reconfigure the connection schedules dynamically in response to nodes being destroyed or running out of battery. This is especially important for the cluster and ring-like approaches, as these have single points of failure before a *broadcast* will be impossible. Reconfiguring the connection schedules will also let us spread the energy usage more evenly in the cluster topology, as this will allow us to dynamically change the master. We should therefore look into ways of allowing the nodes coordinate and reconfigure their schedules.

10.6 Optimal fully connected schedule

In our current experiments, we have used a simple heuristic to create the connection schedule for the fully connected topology. However, we have not analyzed exactly how many parallel connections this algorithm generates. There may also be algorithms which produce a higher number of connections per slot, and smaller frame-sizes. We have also not evaluated the energy cost of these algorithms which may be useful when reconfiguring the schedules dynamically. We should therefore explore algorithms to create the fully connected connection schedules, and evaluate their energy requirements.

11 Conclusion

In this thesis we have introduced four non-opportunistic approaches to share data between nodes in a neighborhood by scheduling connections. These were evaluated using a simulation to calculate the energy consumption, latency and throughput of the networks while idle, while performing *broadcasts* of 1 byte, and while performing *broadcasts* of 100mb.

The experiments showed that a ring or clustered approach will use the least amount of energy at the cost of higher latency and lower throughput. On the other hand, the parallel ring and fully connected topologies are more costly while idle, but allows for lower latency and higher throughput during high loads. It also showed that a clustered and fully connected approaches place more of the energy requirements on a specific nodes, which must be taken into consideration when deciding the energy-capacity of each node.

In the arctic tundra, energy conservation is most important. As such, a ring or clustered approach should be considered unless the amount of data to *broadcast* is high.

Bibliography

[1] Alan Mainwaring et al. “Wireless Sensor Networks for Habitat Monitoring.” In: *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*. WSNA ’02. Atlanta, Georgia, USA: Association for Computing Machinery, 2002, pp. 88–97. ISBN: 1581135890. DOI: 10.1145/570738.570751. URL: https://doi.org/10.1145/570738.570751.

[2] Sukun Kim et al. “Wireless Sensor Networks for Structural Health Monitoring.” In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. SenSys ’06. Boulder, Colorado, USA: Association for Computing Machinery, 2006, pp. 427–428. ISBN: 1595933433. DOI: 10.1145/1182807.1182889. URL: https://doi.org/10.1145/1182807.1182889.

[3] Chenhe Li et al. “To Monitor or Not: Lessons from Deploying Wireless Sensor Networks in Data Centers.” In: *Proceedings of the 7th International Workshop on Real-World Embedded Wireless Systems and Networks*. RealWSN’18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 43–48. ISBN: 9781450360487. DOI: 10.1145/3277883.3277887. URL: https://doi.org/10.1145/3277883.3277887.

[4] Ning Xu et al. “A Wireless Sensor Network For Structural Monitoring.” In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. SenSys ’04. Baltimore, MD, USA: Association for Computing Machinery, 2004, pp. 13–24. ISBN: 1581138792. DOI: 10.1145/1031495.1031498. URL: https://doi.org/10.1145/1031495.1031498.

[5] Xinwei Fang and Iain Bate. “Issues of Using Wireless Sensor Network to Monitor Urban Air Quality.” In: *Proceedings of the First ACM International Workshop on the Engineering of Reliable, Robust, and Secure Embedded Wireless Sensing Systems*. FAILSAFE’17. Delft, Netherlands: Association for Computing Machinery, 2017, pp. 32–39. ISBN: 9781450354820. DOI: 10.1145/3143337.3143339. URL: https://doi.org/10.1145/3143337.3143339.

[6] Isak Østrem Hellemo. “Opportunistic data transfer for IoT and Cyber-Physical Systems with mostly sleeping nodes.” Unpublished Capstone Project. 2021.

[7] Niki Trigoni et al. “WaveScheduling: Energy-Efficient Data Dissemination for Sensor Networks.” In: *Proceeedings of the 1st International Workshop on Data Management for Sensor Networks: In Conjunction with VLDB 2004*. DMSN ’04. Toronto, Canada: Association for Computing Machinery, 2004, pp. 48–57. ISBN: 9781450377959. DOI: 10.1145/1052199.1052209. URL: https://doi.org/10.1145/1052199.1052209.

[8] Nikolaos A. Pantazis et al. “Energy efficiency in wireless sensor networks using sleep mode TDMA scheduling.” In: *Ad Hoc Networks* 7.2 (2009), pp. 322–343. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2008.03.006. URL: https://www.sciencedirect.com/science/article/pii/S1570870508000462.

[9] H. Sabbineni and K. Chakrabarty. “Location-aided flooding: an energy-efficient data dissemination protocol for wireless-sensor networks.” In: *IEEE Transactions on Computers* 54.1 (2005), pp. 36–46. DOI: 10.1109/TC.2005.8.

[10] Aasem Ahmad and Zdenek Hanzalek. “An Energy-Efficient Distributed TDMA Scheduling Algorithm for ZigBee-like Cluster-Tree WSNs.” In: *ACM Trans. Sen. Netw.* 16.1 (Oct. 2019). ISSN: 1550-4859. DOI: 10.1145/3360722. URL: https://doi.org/10.1145/3360722.

[11] Michael J. Murphy et al. “Experiences Building and Deploying Wireless Sensor Nodes for the Arctic Tundra.” In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2021, pp. 376–385. DOI: 10.1109/CCGrid51090.2021.00047.

[12] Issam Raïs, Loic Guegan, and Otto Anshus. “Impact of loosely coupled data dissemination policies for resource challenged environments.” In: *2022 IEEE/ACM 22st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2022.

[13] Kais Mekki et al. “A comparative study of LPWAN technologies for large-scale IoT deployment.” In: *ICT Express* 5.1 (2019), pp. 1–7. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2017.12.005. URL: https://www.sciencedirect.com/science/article/pii/S2405959517302953.

[14] *zigbee The Full-Stack Solution for All Smart Devices*. https://csa-iot.org/all-solutions/zigbee/. Accessed: 2022-07-9.

[15] Elke Mackensen, Matthias Lai, and Thomas M. Wendt. “Bluetooth Low Energy (BLE) based wireless sensors.” In: *SENSORS, 2012 IEEE*. 2012, pp. 1–4. DOI: 10.1109/ICSENS.2012.6411303.

[16] S. Ramanathan and E.L. Lloyd. “Scheduling algorithms for multihop radio networks.” In: *IEEE/ACM Transactions on Networking* 1.2 (1993), pp. 166–177. DOI: 10.1109/90.222924.

[17] Jerry Banks et al. *Discrete-Event System Simluation*. Fifth Edition. Pearson Education Ltd., 2010.

[18] I. McGregor. "The relationship between simulation and emulation." In: vol. 2. Jan. 2003, 1683–1688 vol.2. ISBN: 0-7803-7614-5. DOI: 10.1109/WSC.2002.1166451.

[19] Pedro Velho and Arnaud Legrand. "Accuracy Study and Improvement of Network Simulation in the SimGrid Framework." In: *SIMUTools'09, 2nd International Conference on Simulation Tools and Techniques* (Mar. 2009). DOI: 10.1145/1537614.1537632.

[20] Richa Sharma, Vasudha Vashisht, and Umang Singh. "Modeling and simulation frameworks for Wireless Sensor Networks: A Comparative Study." In: *IET Wireless Sensor Systems* 10 (Oct. 2020). DOI: 10.1049/iet-wss.2020.0046.

[21] Sigurd Karlstad. *Clock Synchronization between Observational Units in the Arctic Tundra*. Master's Thesis. 2021. URL: https://hdl.handle.net/10037/21484.

[22] Jeff Geerling. *Power Consumption Benchmarks*. https://www.pidramble.com/wiki/benchmarks/power-consumption. Accessed: 2022-07-9.

[23] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. "A survey on LPWA technology: LoRa and NB-IoT." In: *ICT Express* 3.1 (2017), pp. 14–21. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2017.03.004. URL: https://www.sciencedirect.com/science/article/pii/S2405959517300061.