AARHUS UNIVERSITET

Master of Science in Computer Engineering

Wireless Sensor Networks

The Agents of WSN Mini-project 6

Due date: 18 December 2024

Participants in the project:

Student Registration Number	**Name**
202010613	Bjarke J. Karlsen
202403561	Kevin Phan
202310943	Shahinoor Razia
202403885	Sanzida Akter Chadne

Contents

1 Introduction

This report presents the design, implementation, and evaluation of a leader election mechanism in a Multi-Agent System (MAS) composed of wireless sensor nodes. The goal is to ensure dynamic, robust, and energy-efficient leader selection in response to network events such as node failures, energy depletion, and topology changes. As modern Wireless Sensor Networks (WSNs) are increasingly applied in fields like environmental monitoring and industrial process control, reliable and autonomous coordination mechanisms are essential for maintaining system functionality despite the inherent unpredictability of wireless channels, limited energy resources, and intermittent connectivity.
In this mini-project, we target a scenario involving at least five wireless sensor nodes, denoted as **A**, **B**, **C**, **D**, and **E**. Initially, node **A** acts as the leader, managing tasks such as data aggregation or synchronization. Over time, the appointed leader may need to change due to various conditions, ensuring that the network continues to operate effectively and efficiently. By implementing and analyzing a leader election protocol, we aim to demonstrate the group's deep understanding of WSN principles, including resource constraints, communication protocols, and distributed decision-making.

1.1 Problem Definition

The central problem addressed in this project is to design a mechanism that can autonomously elect, maintain, and re-elect a leader node under varying operational conditions. Leader elections are triggered by one or more of the following events:

1. **Leader Failure:** If the current leader does not respond within a specified timeout (e.g., due to hardware failure, range issues, or energy depletion), other nodes must detect this absence and initiate a new election.
2. **Energy Threshold:** When the leader's energy level falls below a predefined threshold, it voluntarily relinquishes its leadership to conserve power and prolong the overall network lifetime.
3. **Network Reconfiguration:** The network may require leader rotation for load balancing or improved reliability— for instance, when adding new nodes or encountering sustained high traffic levels.

Our approach must determine clear and justifiable criteria for leader selection. Initially, a simple highest-ID rule may suffice, but integrating energy awareness can further extend network longevity and reduce the frequency of transitions. Thus, we must compare and evaluate both strategies. Additionally, this mechanism should function seamlessly without human intervention, allowing the MAS to adapt and reorganize as conditions change.

1.2 Approach

Leader Election Algorithm

We implement a variant of the Bully algorithm, a well-established leader election method in distributed systems. The Bully algorithm is chosen due to its straightforward, deterministic nature and minimal overhead compared to more complex protocols like Chang & Roberts or consensus-based approaches (e.g., Raft). Its simplicity aligns well with the resource constraints and limited computational capabilities of wireless sensor nodes. In our adaptation, two criteria are considered:

- **Highest ID:** The node with the highest node ID becomes the leader.
- **Highest Energy:** Among candidate nodes, the one with the highest residual energy is selected, aiming to reduce leader turnover and improve network stability.

Communication Protocols and Message Types

The leader election relies on a small set of predefined message types that facilitate decentralized decision-making:

- **MSG_START_ELECTION:** Broadcast by a node that detects leader failure or low leader energy, initiating the election process.
- **MSG_OK:** Sent in response to an election request, indicating that the responding node is operational and a potential leader candidate.
- **MSG_COORDINATOR:** Broadcast by the newly elected leader to inform all nodes of its status.
- **Heartbeat Messages:** Periodically sent by the current leader to confirm its availability and prevent unnecessary elections.

These messages leverage the underlying MAC and routing layers for reliable delivery. When possible, broadcasts are used to reach all nodes simultaneously, while unicast messages target specific nodes to limit unnecessary network traffic. By integrating timeout mechanisms, nodes promptly detect missing responses and conclude the election process efficiently.

Energy Management and Performance Evaluation

Incorporating energy-awareness ensures that leadership roles do not rapidly rotate due to energy depletion, thus prolonging network lifetime. Our approach involves periodically monitoring each node's energy consumption and adjusting the election process accordingly. To evaluate performance, we employ both simulation (e.g., Cooja) and real-world testing on hardware motes. Metrics include energy usage, message complexity, election latency, and memory footprint. By analyzing these factors, we verify that our solution is both practically feasible and optimal for the targeted WSN environment.

2 Architecture

2.1 Bully Algorithm

The Bully Algorithm is a well-established method for leader election in distributed systems. Its simplicity, deterministic behavior, and proven reliability make it an excellent choice for scenarios involving resource-constrained wireless sensor networks (WSNs). The algorithm assumes direct communication is possible among all nodes, simplifying coordination and ensuring timely detection of leader failures.

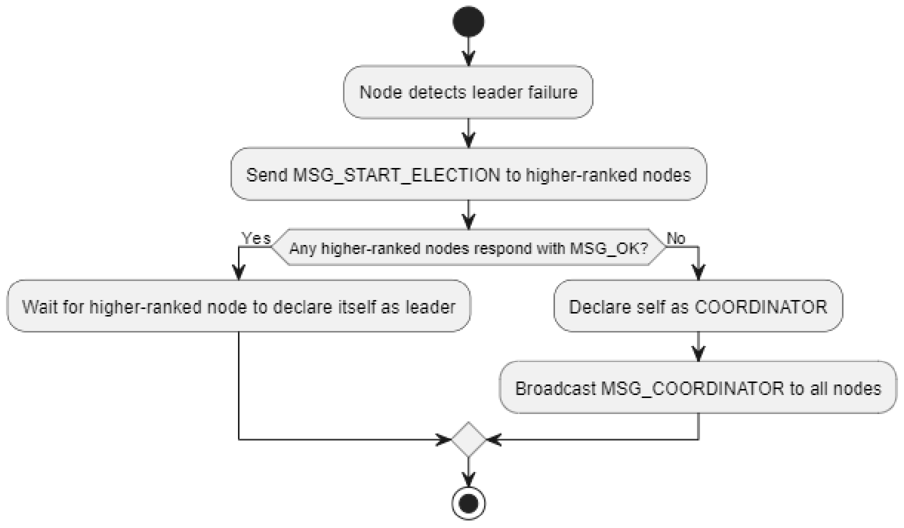

Node detects leader failure
Send MSG_START_ELECTION to higher-ranked nodes
Yes
Any higher-ranked nodes respond with MSG_OK?
No
Wait for higher-ranked node to declare itself as leader
Declare self as COORDINATOR
Broadcast MSG_COORDINATOR to all nodes

Figure 1: Overview for the core logic for the Bully Algorithms

This flowchart illustrates the leader election process in a distributed system:

1. Detection: A node detects the failure of the current leader.
2. Election Start: It sends a MSG_START_ELECTION to higherranked nodes.
3. Response Check:
   - Yes: If any higherranked node responds with MSG_OK, the node waits for that node to declare itself as the leader.
   - No: If no higherranked nodes respond, the node declares itself as the COORDINATOR (new leader).
4. Broadcast: The new coordinator broadcasts MSG_COORDINATOR to all nodes.
5. Completion: The election process ends.

In our adaptation of the Bully Algorithm, we introduce energy-awareness to suit resource-constrained WSNs. Specifically, we integrate energy thresholds that allow only nodes with sufficient remaining energy to participate in elections. This helps reduce the frequency of leader changes and prolongs network lifetime. While this is not part of the original Bully Algorithm, this extension makes it more suitable for energy-limited environments like those found in IoT and WSN deployments.
Though the Bully Algorithm is conceptually simple and deterministic, its scalability can be challenging in very large networks since it involves communication with all higher-ranked nodes during elections. However, for small to medium-sized WSNs, it offers a straightforward approach to leader election without complex consensus mechanisms.

2.1.1 Mesh-Under vs Route-Over

Mesh-under routing occurs at the link layer (Layer 2), where multi-hop forwarding is managed locally without exposing these hops to the network layer. From the network layer's perspective, the entire network appears as a single hop. While this reduces network-layer overhead and simplifies IP-level processing, it limits the layer's ability to optimize routing decisions across intermediate hops.

Route-over routing, on the other hand, operates at the network layer (Layer 3). Each node acts as an IP router and is typically managed by a routing protocol such as RPL, providing greater flexibility and the ability to optimize routes. However, this also increases complexity and overhead at the network layer.
In our system, we use mesh-under routing to keep the upper layers simpler. By handling multi-hop forwarding at the link layer, we avoid implementing complex IP-based routing protocols, allowing us to focus on core functionality such as leader election, sensor data collection, and energy management without incurring additional network-layer complexity.

2.2 Communicate between nodes

We have used the MAC layer (Medium Access Control), as it is part of the Data Link Layer in the OSI model which is essential for organising data into addressed frames, managing access to the communication medium, detecting errors, and controlling data flow to prevent congestion. In wireless networks, efficient communication is ensured by avoiding collisions, coordinating transmissions, and prioritising critical traffic. Contiki OS primarily uses CSMA-based MAC protocols aligned with the IEEE 802.15.4 standard for low-power wireless communication, with some implementations adopting TDMA for synchronised transmissions. Key features include beacon-enabled modes, guaranteed time slots, acknowledgements, integration with NullNet and Netstack for flexible networking, radio duty cycling for energy conservation, and interaction with 6LoWPAN for IPv6 communication. The communication workflow involves frame creation, carrier sensing, transmitting with acknowledgments, receiving and processing frames, handling errors through retries, and managing energy by controlling the radio's power state. Enhancements like scheduled transmissions, adaptive backoff, security measures, and mesh networking support further optimise performance. In general, the MAC layer in Contiki OS is vital for reliable, efficient, and energy-conscious communication in resource-constrained IoT environments.

2.3 CSMA/CA

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is a Medium Access Control (MAC) mechanism commonly used in low-power wireless networks to control how nodes access the shared communication medium. In contrast to IEEE 802.11 WLANs, which may employ RTS/CTS handshakes, IEEE 802.15.4-based systems and Contiki OS's default MAC layers typically rely solely on carrier sensing and random backoff. Before transmitting, a node listens to the channel (clear channel assessment) to ensure it is free. If the channel is busy, the node waits for a random backoff period before attempting again. Once the medium is clear, it transmits its data. Upon successful reception, the receiver returns an acknowledgment (ACK), confirming proper delivery at the MAC layer.
In our Contiki OS-based project, CSMA/CA is essential for managing node access to the shared wireless medium, minimizing collisions, and enhancing data transmission reliability. It aligns with the IEEE 802.15.4 standard, integrates seamlessly with Contiki's Netstack, and leverages Radio Duty Cycling (RDC) to conserve energy. In our Bully Algorithm-based leader election system, CSMA/CA ensures the timely and reliable delivery of critical messages such as heartbeats and election signals, adapts to increased traffic by adjusting backoff periods, and supports energy management by reducing unnecessary transmissions.
Despite its effectiveness, challenges remain. Densely populated networks can face higher collision probabilities, and careful tuning is required to balance latency and energy consumption. Scalability is also a concern as network size grows. Potential enhancements include scheduled transmissions, adaptive backoff algorithms, and integrated security measures like frame encryption and authentication. Such improvements can further optimize CSMA/CA performance, ensuring efficient, reliable, and energy-conscious communication within resource-constrained IoT environments.

By leveraging Contiki OS's built-in capabilities, we can concentrate on higher-level protocol implementations, relying on well-established MAC mechanisms for collision avoidance and energy efficiency.

2.3.1 NullNet

NullNet is a minimal, "no-frills" network layer implementation available in Contiki OS. It operates by directly passing packets between the application and the link layer without implementing routing, fragmentation, or protocol-specific logic such as that found in RPL. This lightweight design minimizes code footprint and complexity, helping our Bully Algorithm-based leader election system fit within stringent ROM limitations. While NullNet does not itself provide routing or reliability guarantees, it allows us to integrate only the functionalities we require, conserving resources and enabling energy-efficient communication. By selecting NullNet, we can focus on higher-level protocols and essential IoT functionalities without incurring the overhead of unnecessary network-layer protocols.

3 Implementation

In this section, we go into how we are implementation the multi-agent system.
As mentioned in the architecture section, we decided the way to select a leader is done by utilising the Bully Algorithm. In this algorithm, a mote can be either the leader of all the nodes or be a member, and lastly be the orphan. Each node should be one of these different types and should be able to change depending on the conditions, such as missing a leader. All nodes share the same code that can be found in Appendix x. We are testing two versions, but this will be discussed in more detail later on.

3.1 Implementing a leader election mechanism

To implement the leader election mechanism, we had to take into account who should be able to call for an election. Either it is their leader who relinquishes the role and calls for an election, or a node does not receive a heartbeat message from the leader in a set amount of time. Then the node will become an orphan and try to call for an election. However, for a node to call for election, it has to have more than the threshold of 20% power. If the leader crosses a threshold of 20% the leader relinquishes the responsibility and starts an election.

For selection, a new leader was considered in two different implementations. Firstly, we look at the node with the highest ID. Secondly, we look at the node that has the highest amount of energy, which would reduce the number of elections. In the two code snippets,Listing 1 and Listing 2, below we can see the small difference in the implementation.

Listing 1: Simplified code from the node with highest ID wins

```
    if (node.energy < received_msg->sender_node_data.sender_energy) break
```

Listing 2: Simplified code from the node with highest energy wins

```
      if (node.id < received_msg->sender_node_data.sender_id) break
```

3.2 Evaluating memory resources and system performance

We rely on the Energest library's operational metrics—CPU, LPM, Transmit, and Listen times—because Contiki provides these readings directly, allowing us to assess system performance without altering the code or using specialized profiling tools. Although these metrics do not directly measure memory usage, they reflect how actively nodes operate, which often correlates with handling more complex or larger data structures. This indirect approach avoids intrusive methods that could distort results in a resource-constrained environment. It also makes it easier to compare different setups and can guide us toward more targeted memory analysis if the observed activity suggests that additional investigation is warranted.

3.3 Reading the temperature

To read the temperature, the implementation is based on the sht11-sensor.h library, which allows us to read the sensor values. Each member reads the temperature and humidity values and converts them to readable values. Then the member sends it to the leader. However, the values we read are not temperature and humidity at first and have to be converted as in the source (Sensirion 2005). To convert sensor outputs into temperature and humidity, use the following linear equations and constants. The temperature is calculated using the formula:
Temperature conversion

$$T = d_1 + d_2 \cdot SO_T$$

Where:

- SO_T: Sensor Output

The constants for d_1 based on the supply voltage ($VDD = 3.0\,\mathrm{V}$) are:

$$d_1 = -39.6\,^\circ C$$

The constant for d_2 based on the sensor resolution (14-bit) is:

$$d_2 = 0.01\,^\circ C$$

Humidity Conversion
The relative humidity is calculated using the formula:

$$RH_{\text{linear}} = c_1 + c_2 \cdot SO_{RH} + c_3 \cdot SO_{RH}^2$$

Where:

- SO_{RH}: Sensor Output

The constants for c_1, c_2, c_3 based on the sensor resolution (12-bit) are:

$$c_1 = -2.0468, \quad c_2 = 0.0367, \quad c_3 = -1.5955 \times 10^{-6}$$

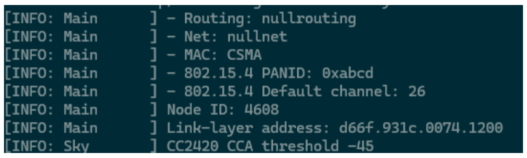
3.4 Changes over the time

When creating the project, we have made a lot of adjustments for serveal reason, first, the simulation does not behave as the node. Some part of the code behaved differently as excepted.

First of all, we started the project thinking we had to do routing for a bigger system and the nodes did not have to have a direct connect. So we started to implement it using RPL and looking at how the worked. After we had a implementation where the connection worked, we found that we did not have enough ROM if we started to log the states in the code.

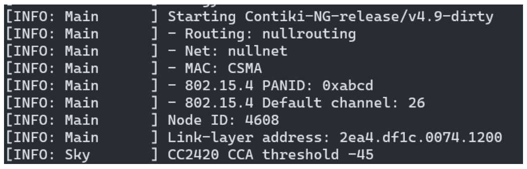
Sending packets with Nullnet presents some challenges. We had to modify the protocol for our transmissions. Initially, we sent the Node ID as a 16-bit integer, but upon receiving by the other node, the value changed from 3 to 768. Therefore, we decided to format the integer. This issue might be due to packet size constraints or conversion to an 8-bit integer pointer as the packet is being sent. For temperature and humidity, we separated the integer and fractional parts before sending. We also adjusted the message type, which was initially an enum but was changed to an integer.

The node_id for the nodes is created using the link layers last two bytes, and we encountered errors with the having the same last two bytes, so we had to make some adjustments to the code and not only depend on the node_id.



```
[INFO: Main      ] - Routing: nullrouting
[INFO: Main      ] - Net: nullnet
[INFO: Main      ] - MAC: CSMA
[INFO: Main      ] - 802.15.4 PANID: 0xabcd
[INFO: Main      ] - 802.15.4 Default channel: 26
[INFO: Main      ] Node ID: 4608
[INFO: Main      ] Link-layer address: d66f.931c.0074.1200
[INFO: Sky       ] CC2420 CCA threshold -45
```

(a) Log output showing the initialization of a Contiki-NG network node 1



```
[INFO: Main      ] Starting Contiki-NG-release/v4.9-dirty
[INFO: Main      ] - Routing: nullrouting
[INFO: Main      ] - Net: nullnet
[INFO: Main      ] - MAC: CSMA
[INFO: Main      ] - 802.15.4 PANID: 0xabcd
[INFO: Main      ] - 802.15.4 Default channel: 26
[INFO: Main      ] Node ID: 4608
[INFO: Main      ] Link-layer address: 2ea4.df1c.0074.1200
[INFO: Sky       ] CC2420 CCA threshold -45
```

(b) Log output showing the initialization of a Contiki-NG network node 2

4 Simulation

After implementing, we tested it on both the COOJA simulation and the TelosB mote. Testing began with the COOJA simulation, which revealed the need for adjustments to work on the TelosB mote.

4.1 COOJA simulation

COOJA was our main testing tool since it allowed us to deploy a large number of agents. These agents were positioned to maintain direct connectivity with each other. However, the temperature and humidity settings in the simulation did not yield useful data. In Figure 3, we observe one of the configurations in a COOJA simulation. We simulated several scenarios with different numbers of agents, ranging from 5 to 10, to assess the impact on both the leaders' and members' resources. We count the overall resources used and save them in motes as shown in Figure 4. Then each of these files was extracted to a PC.

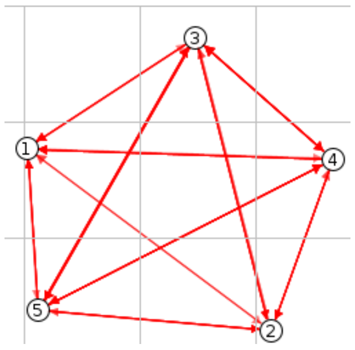
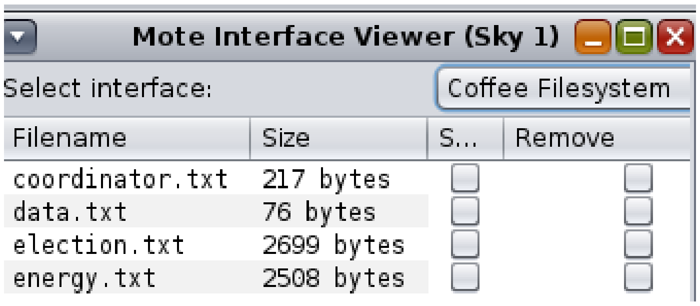


Figure 3: Cooja Simulation - A setup of the motes



Mote Interface Viewer (Sky 1)

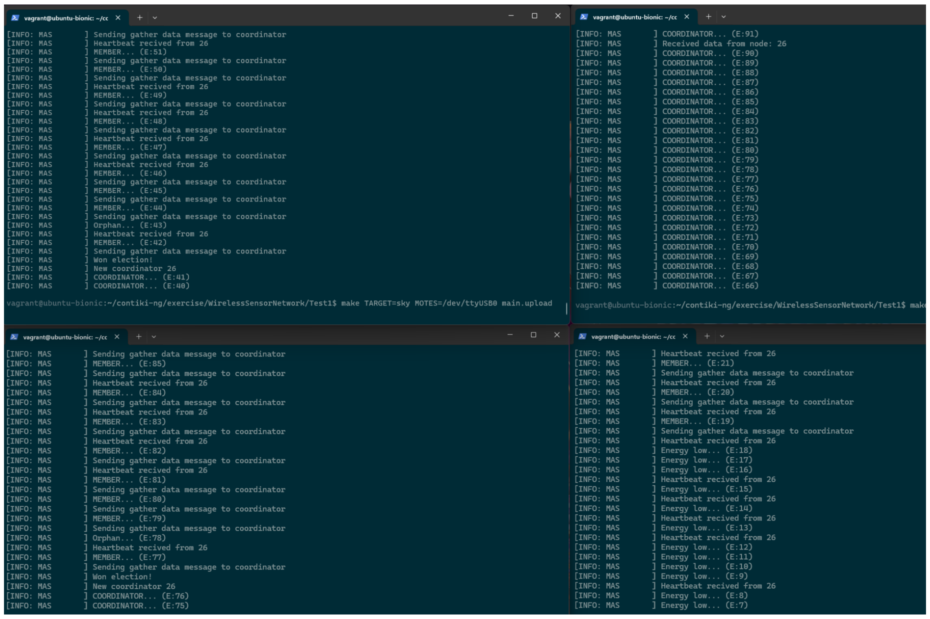
Select interface: Coffee Filesystem

Filename	Size	S...	Remove
coordinator.txt	217 bytes	☐	☐
data.txt	76 bytes	☐	☐
election.txt	2699 bytes	☐	☐
energy.txt	2508 bytes	☐	☐

Figure 4: Cooja simulation - show stored the files

4.2 TelosB Motes

As we only had four TelosB motes, we were unable to successfully test this with more motes. In the run for on the mote, we wanted to see if the experimentation worked and if the collection of data also worked. We connected all motes to one PC and then read the serial output. The data was then save to files and then data was processed.
Since we only had four TelosB motes, we could not test with more motes. During the test run, our aim was to verify both the implementation and data collection processes. All motes were connected to one PC and we read the serial output, which is



```
vagrant@ubuntu-bionic: ~/cc
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:51)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] MEMBER... (E:50)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:49)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:48)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:47)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:46)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] MEMBER... (E:45)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] MEMBER... (E:44)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Orphan... (E:43)
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:42)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Won election!
[INFO: MAS       ] New coordinator 26
[INFO: MAS       ] COORDINATOR... (E:41)
[INFO: MAS       ] COORDINATOR... (E:40)

vagrant@ubuntu-bionic:~/contiki-ng/exercise/WirelessSensorNetwork/Test1$ make TARGET=sky MOTES=/dev/ttyUSB0 main.upload
```

```
vagrant@ubuntu-bionic: ~/cc
[INFO: MAS       ] COORDINATOR... (E:91)
[INFO: MAS       ] Received data from node: 26
[INFO: MAS       ] COORDINATOR... (E:90)
[INFO: MAS       ] COORDINATOR... (E:89)
[INFO: MAS       ] COORDINATOR... (E:88)
[INFO: MAS       ] COORDINATOR... (E:87)
[INFO: MAS       ] COORDINATOR... (E:86)
[INFO: MAS       ] COORDINATOR... (E:85)
[INFO: MAS       ] COORDINATOR... (E:84)
[INFO: MAS       ] COORDINATOR... (E:83)
[INFO: MAS       ] COORDINATOR... (E:82)
[INFO: MAS       ] COORDINATOR... (E:81)
[INFO: MAS       ] COORDINATOR... (E:80)
[INFO: MAS       ] COORDINATOR... (E:79)
[INFO: MAS       ] COORDINATOR... (E:78)
[INFO: MAS       ] COORDINATOR... (E:77)
[INFO: MAS       ] COORDINATOR... (E:76)
[INFO: MAS       ] COORDINATOR... (E:75)
[INFO: MAS       ] COORDINATOR... (E:74)
[INFO: MAS       ] COORDINATOR... (E:73)
[INFO: MAS       ] COORDINATOR... (E:72)
[INFO: MAS       ] COORDINATOR... (E:71)
[INFO: MAS       ] COORDINATOR... (E:70)
[INFO: MAS       ] COORDINATOR... (E:69)
[INFO: MAS       ] COORDINATOR... (E:68)
[INFO: MAS       ] COORDINATOR... (E:67)
[INFO: MAS       ] COORDINATOR... (E:66)

vagrant@ubuntu-bionic:~/contiki-ng/exercise/WirelessSensorNetwork/Test1$ make
```

```
vagrant@ubuntu-bionic: ~/cc
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] MEMBER... (E:85)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:84)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:83)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:82)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:81)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] MEMBER... (E:80)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] MEMBER... (E:79)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Orphan... (E:78)
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:77)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Won election!
[INFO: MAS       ] New coordinator 26
[INFO: MAS       ] COORDINATOR... (E:76)
[INFO: MAS       ] COORDINATOR... (E:75)
```

```
vagrant@ubuntu-bionic: ~/cc
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:21)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:20)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] MEMBER... (E:19)
[INFO: MAS       ] Sending gather data message to coordinator
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] Energy low... (E:18)
[INFO: MAS       ] Energy low... (E:17)
[INFO: MAS       ] Energy low... (E:16)
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] Energy low... (E:15)
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] Energy low... (E:14)
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] Energy low... (E:13)
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] Energy low... (E:12)
[INFO: MAS       ] Energy low... (E:11)
[INFO: MAS       ] Energy low... (E:10)
[INFO: MAS       ] Energy low... (E:9)
[INFO: MAS       ] Heartbeat recived from 26
[INFO: MAS       ] Energy low... (E:8)
[INFO: MAS       ] Energy low... (E:7)
```

Figure 5: Serial outpot of the fou r TelosB motes running

5 Result and discussion

All result is made with `energest_type_time` and is not converted to second.

5.1 Efficiency of leader election mechanism

The bully algorithms work on who has the highest ID to be the leader. This is fine but, however, if we want to see if we can optimise it we can look at the energy the node has and select the leader based on which has the most energy. This leads to less changes in the leader and is therefore more memory efficient. We are testing both solutions and comparing which is the most memory efficient and has the least number of different leaders.
However, note that we do not take battery level into account. The decrease in battery should be based on the CPU, LPM, Transmit and Listen time. This will provide a biased result. We did not take it into account when making the system make this test general worse.
This test is preform with having 10 motes
The Table 1, compares resource usage metrics between the 'Member or Orphan' and 'Coordinator' roles. Shows differences in resource usage across two contexts: 'Use ID' and 'Use Battery Level'.

Category	Use ID				Use Battery Level			
	CPU	LPM	Transmit	Listen	CPU	LPM	Transmit	Listen
Member or Orphan	3854	43802	22	45918	1498	16068	6	16594
Coordinator	21520	259480	116	272556	12501	166219	88	173606
Difference	17666	215678	94	226638	11003	150151	82	157012

Table 1: Comparison of Average Values Using ID and Battery Level

Another thing to see is that the ID had 5 different election while using the battery only had one. This will drastically reduce the amount of load on the network.

5.2 Scalability with increasing number of agents

To evaluate the scalability of the Bully algorithm, we conducted several tests by varying the number of agents and monitored their resource usage during the initial election, as all agents were active at that time. The outcomes are presented in Table 2, which showcases the total resource consumption for leader selection across different numbers of agents (5, 6, and 10). In contrast, Table 3 shows the consumption of resources per agent for the selection of leaders, again considering different agent counts (5, 6, and 10).

Agents	**CPU**	**LPM**	**Transmit**	**Listen**
5	94.098	921.763	588	996.856
6	189.917	1.562.614	1.192	1.729.189
10	2.735.417	16.923.358	29.513	19.568.180

Table 2: Comparison of Election 1 Data Points Across Different Agent Counts in Ticks

Agents	**CPU**	**LPM**	**Transmit**	**Listen**
5	18.820	184.353	118	199.371
6	31.653	260.436	199	288.198
10	273.542	1.692.336	2951	1.956.818

Table 3: Comparison of Average Election Data Points Across Different Agent Counts in Ticks

From Table 2 we can observe how much more activity and pressure there is in the network that would double the amount of agents from 5 to 10. It demonstrates that the bully algorithm is functional in smaller mesh system, however, the agent count will make a substantially grow in resources.

As we add more individual agents, each new node significantly increases system load demands, needing more CPU, LPM, Transmit, and Listen resources per node. The system's computational requirement grows because the number of messages exchanged increases quadratically $O(n^2)$, causing more network traffic and potential latency issues.

5.3 Resources overhead per agent during the process

The data was collected using Contiki's energest_flush() function from the start to the end of the election, measuring energy usage across various activities. We focused our analysis on the first election, ensuring that all nodes had sufficient energy for participation. As shown in Table 4, the memory overhead scales with the number of agents in the election. Adding more nodes leads to a significant increase in resource usage, especially in CPU time, LPM, and listening metrics. This trend underscores the increasing complexity in managing communication and coordination as the size of the network increases.

Scenario	**CPU**	**LPM**	**Transmit**	**Listen**
TelosB 4 Motes (Battery)	48,672.0	242,371.25	485.25	169,080.0
COOJA 10 Motes (Battery)	109,120.8	364,076.3	1,357.0	461,747.4
COOJA 10 Motes (ID)	273,541.7	1,692,335.8	2,951.3	1,956,818.0
COOJA 6 Motes (Battery)	31,652.83	260,435.67	198.67	288,198.17
COOJA 5 Motes (Battery)	94,098.0	921,763.0	588.0	996,856.0

Table 4: Energy Consumption Comparison Across Scenarios

5.4 System resilience to leader failure and energy constraints

The Listing 3 and the log output detail a resilience test, where a leader node manages tasks. The test simulates the leader's failure due to energy constraints, showcasing the ability to elect a new leader. Here's an overview of the process:

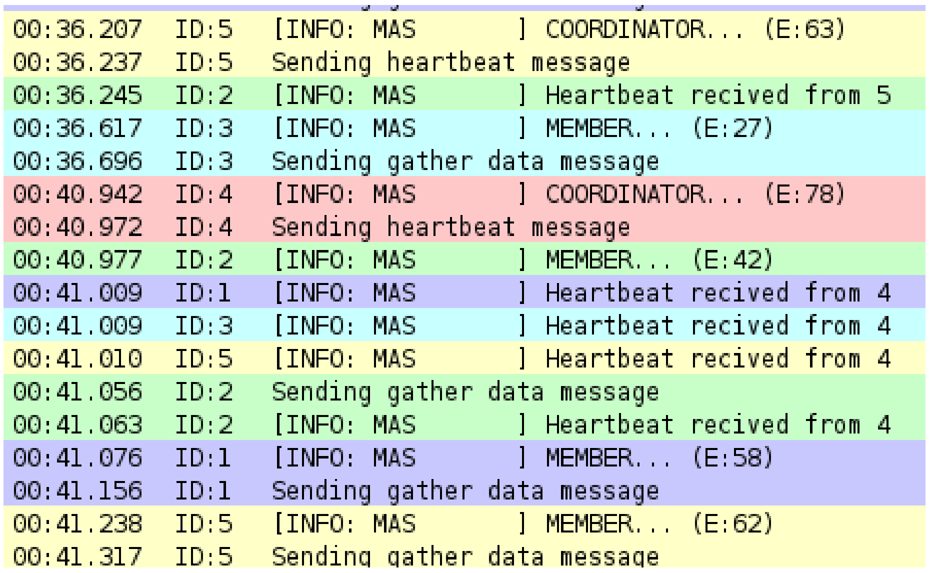
Listing 3: Coordinator Election Log

```
04:16.512   ID:5   [INFO: MAS       ] COORDINATOR... (E:20)
04:16.545   ID:5   [DBG : MAS       ] CPU: 731949, LPM: 7639350, Transmit: 5728, ...
     Listen: 8045400
04:16.580   ID:5   [DBG : MAS       ] Energy data written to file.
04:16.582   ID:5   Sending heartbeat message8743768
04:37.895   ID:5   [DBG : MAS       ] Energy data written to file.

04:37.900   ID:5   [DBG : MAS       ] Energy below threshold.
04:42.910   ID:4   [INFO: MAS       ] [N4] New coordinator 4 (E:30)
04:42.914   ID:4   [INFO: MAS       ] [N4] Won election (E:30)
04:42.949   ID:4   [DBG : MAS       ] Writing to file: 32 - End: CPU: 750802, LPM: ...
     8497680, Transmit: 4274, Listen: 8911107
04:42.985   ID:4   [DBG : MAS       ] Energy data written to file.
04:42.992   ID:5   [INFO: MAS       ] New coordinator 4
```

5.5 Validate system behaviour under multiple consecutive leader changes

In this scenario, we simulate multiple leader elections using Cooja. Initially, the nodes are separated, causing two nodes to become leaders independently. When they return to range, they resolve leadership through heartbeat messages. In Figure 6, we can see the log of the process. ID5 steps down and does not do anything else, as mentioned earlier we do not send the collected data from a node to another.



```
00:36.207   ID:5   [INFO: MAS       ] COORDINATOR... (E:63)
00:36.237   ID:5   Sending heartbeat message
00:36.245   ID:2   [INFO: MAS       ] Heartbeat recived from 5
00:36.617   ID:3   [INFO: MAS       ] MEMBER... (E:27)
00:36.696   ID:3   Sending gather data message
00:40.942   ID:4   [INFO: MAS       ] COORDINATOR... (E:78)
00:40.972   ID:4   Sending heartbeat message
00:40.977   ID:2   [INFO: MAS       ] MEMBER... (E:42)
00:41.009   ID:1   [INFO: MAS       ] Heartbeat recived from 4
00:41.009   ID:3   [INFO: MAS       ] Heartbeat recived from 4
00:41.010   ID:5   [INFO: MAS       ] Heartbeat recived from 4
00:41.056   ID:2   Sending gather data message
00:41.063   ID:2   [INFO: MAS       ] Heartbeat recived from 4
00:41.076   ID:1   [INFO: MAS       ] MEMBER... (E:58)
00:41.156   ID:1   Sending gather data message
00:41.238   ID:5   [INFO: MAS       ] MEMBER... (E:62)
00:41.317   ID:5   Sending gather data message
```

Figure 6: Cooja simulating - having multiple leaders

6 Conclusion

This work presents a modified, energy-aware Bully algorithm for leader election in wireless sensor networks (WSNs), demonstrating improved network stability, reduced leader changes, and efficient resource usage. The proposed energy-based selection approach, which prioritises nodes with the highest residual energy, proved more efficient than the traditional ID-based method. It minimised leader transitions and reduced network overhead by lowering CPU, low power mode (LPM), transmit, and listen times. This mechanism led to fewer general elections since the node with the highest residual energy was consistently the leader.

The system effectively managed dynamic and resource-limited WSNs and was resilient to leader failures, quickly adapting to energy depletion and other network changes. However, while the approach scaled well for smaller networks, resource usage grew significantly as the number of nodes increased, posing challenges for scalability. Additionally, the energy consumption of the leader—due to more frequent transmissions and listening—was not explicitly considered, which could accelerate its energy depletion over time. Future enhancements could address this by exploring workload distribution or more frequent checks for nodes with higher residual energy, possibly introducing new states to balance responsibilities.

Despite increased resource overhead in larger networks, the approach remained practical for typical WSN deployments. To further optimise performance, future work could include refining communication protocols, employing hierarchical election schemes, integrating dynamic energy monitoring, and improving collision avoidance strategies. Testing on physical devices would also help validate the algorithm's effectiveness in real-world scenarios.

References

Sensirion (2005). *Datasheet SHT1x.* URL: `https://eu.mouser.com/datasheet/2/682/seri_s_a0002858305_1-2291243.pdf`.