LINFO2146 – Group Project

GUSTIN Théo - 42052000

May 16, 2025

1 Introduction

To tackle this project, I used a two-phase strategy: first, building the tree algorithm and having everything working without any energy factor, and then "simply" taking energy into account. In my GitHub, you can find two folders, *unergised* and *energised*, where you can find the three node C files: `border-router.c`, `computation-node.c`, and `sensor-node.c`. In each folder, there is also a dedicated `Makefile` and `server.py` that are used to interact with the border node.

It is important to note that all my nodes are Cooja motes. In fact, it is required to use Z1 motes for the sensor nodes. The use of Z1 nodes is more "field"-proof since they are real hardware-based devices, whereas Cooja motes are only emulated. In fact, I had to make this choice because I was unable to install the MSP430 toolchain and MSPSim plugin required for Z1 emulation. The only difference I can notice is that the energy cost of actions would differ from the model I used.

In this report, I'll explain both of my implementations, present a suite of test scenarios that I used to ensure the correctness of my work, and finally provide a performance comparison between my two models.

2 Unergised version

The unenergised version comprises the three node types:

1. **Border router**: the root of the tree, relays sensor readings to my computer via the server.py code via a serial socket (default 60001). It is important to note that I always assume that the border router have an ID of 1. I also assume that there will only have one border node.

2. **Computation node**: builds the routing tree via periodic HELLO broadcasts, collects readings from up to $N = 5$ sensors, maintains a fixed-size window of the last 30 values per sensor, computes the slope

$$m \;=\; \frac{N\sum_i i\,v_i - (\sum_i i)(\sum_i v_i)}{N\sum_i i^2 - (\sum_i i)^2}$$

   and sends an OPEN_VALVE command when $m > 0.5$.

3. **Sensor node**: measures a synthetic sensor value every 60s, sends type-2 packets upstream, and opens its LED "valve" on receiving a command.

The header is constitued of 3 bytes : the first one is the type, then the ID and the sensor value or command value.

3 Energy model

We extend each mote with a virtual battery of 1000 units, integrating Contiki's `energest` module to track the energy. I've assign cost value for each action and provide a scenario for LowPowerMode and DeepLowPowerMode. Here are the cost :

Action	**Trigger**	**Energy cost (units)**
CPU active	per second	0.2
Low-power idle (LPM)	per second	0.02
Radio transmit (TX)	per second	1.0
Radio receive/listen (RX)	per second	1.0
HELLO packet (send)	per packet	1.0
Sensor reading (send)	per packet	3.0
Sensor reading (forward RX)	per packet	1.0
Open-valve command (send)	per packet	2.0
Open-valve command (receive)	per packet	1.0
Forward to server (border)	per packet	1.0

Table 1: Energy costs of the various mote actions in our energy-aware model.

Nodes enter LPM when its battery ≤ 30. In this mode, it's recharging at 1 unit per 10 s. We can see that as a "I'm starting to get slower" mode. Once the battery level is below 10, the node enters Deep-LPM mode. In this mode, if it's a sensor, it just sends HELLO but no more data and, on the other hand, if it's a computation node, it only forwards the packets received; it does not do any further calculation. While being in that mode, the node is recharging at 1 unit per 2 s. The node wakes up when its battery reaches 90 % capacity. This can be seen as a "I overheated, let me rest a few minutes." Depending on the ratio, it could lead to a 30 min break; which can be realistic in a real scenario.
I've used "battery" as a name for my energy unit but it can be seen as "CPU capacity" or any other energy-related resource. In real-case scenarios, except for the computation node and server, there are no such limitations. Saving energy in real cases would be to send fewer and lighter packets which I've tried to.

The header is now constitued of 5 bits as follow : type, high byte of node's rank, low byte of the rank, battery level, and the current mode (active, LPM, DLPM

4 Test cases

We defined eight corner-case scenarios:

1. **Exact thresholds:** verify transitions at 30%, 10%, and wake at 90%. In a configuration with Node 1 = border, Node 4 = sensor. I've got the result :

$$
\begin{aligned}
07:32.236 \quad & ID:4 \quad MODE:Node4:LPM, battery && = 30,0\% \\
13:09.236 \quad & ID:4 \quad MODE:Node4:DEEPLPM, battery && = 9,0\% \\
16:59.236 \quad & ID:4 \quad MODE:Node4:WAKE, battery && = 90,0\% \\
23:09.236 \quad & ID:4 \quad MODE:Node4:LPM, battery && = 29,0\% \\
28:51.236 \quad & ID:4 \quad MODE:Node4:DEEPLPM, battery && = 10,0\% \\
32:45.236 \quad & ID:4 \quad MODE:Node4:WAKE, battery && = 90,0\%
\end{aligned}
$$

2. **Energy-based reparenting:** a sensor with two candidate parents (A, B) should switch to B only when $\text{bat}_B \geq \text{bat}_A + 30$. Here there is 2 computation node, 2 and 4. The sensor

node is Node 5. Nodes 2 and 4 equally distant from 5 leading the energy as the only variable

$00:08.730$	$ID:2$	$TREE:Node2:newparent->1$	$(rank=1,bat=100)$
$00:08.730$	$ID:5$	$TREE:Node5:newparent->1$	$(rank=1,bat=100)$
$00:09.954$	$ID:4$	$TREE:Node4:newparent->5$	$(rank=2,bat=98)$
$02:47.721$	$ID:4$	$TREE:Node4:newparent->2$	$(rank=2,bat=74)$
$08:15.295$	$ID:4$	$TREE:Node4:newparent->5$	$(rank=2,bat=48)$
$15:15.286$	$ID:4$	$TREE:Node4:newparent->2$	$(rank=2,bat=71)$
$20:39.255$	$ID:4$	$TREE:Node4:newparent->5$	$(rank=2,bat=48)$
$28:01.321$	$ID:4$	$TREE:Node4:newparent->2$	$(rank=2,bat=72)$

3. **Deep-LPM forwarding:** comp-node in deep sleep forwards readings without computing slope. The tree is 1 - 2 - 4 (As border - computation - sensor). We can see that when Node 2 is entering DPLM mode at 12:30 min, it forward the Node's 4 packets till the 16:52min when it wakes up

$12:22.899$	$ID:2$	$MODE:Node2:DEEPLPM,battery$	$=10,0$
$13:14.236$	$ID:4$	$MODE:Node4:DEEPLPM,battery$	$=9,0$
$16:52.899$	$ID:2$	$MODE:Node2:WAKE,battery$	$=90,0$
$12:13.274$	$ID:4$	$PROCESS:Node4:sendreading96to$	2
$12:13.305$	$ID:2$	$PROCESS:Node2:slope=0,00sensor$	$=4$
$13:13.274$	$ID:4$	$PROCESS:Node4:sendreading40to$	2
$13:13.291$	$ID:2$	$PROCESS:Node2:forwardsensor4to$	1
$13:13.312$	$ID:1$	$PROCESS:ServergotID=4,value$	$=40$
$14:13.274$	$ID:4$	$PROCESS:Node4:sendreading28$	$to2$
$14:13.284$	$ID:2$	$PROCESS:Node2:forwardsensor4$	$to1$
$14:13.294$	$ID:1$	$PROCESS:ServergotID=4,value$	$=28$
$15:13.274$	$ID:4$	$PROCESS:Node4:sendreading85$	$to2$
$15:13.303$	$ID:2$	$PROCESS:Node2:forwardsensor4$	$to1$
$15:13.307$	$ID:1$	$PROCESS:ServergotID=4,value$	$=85$
$16:13.274$	$ID:4$	$PROCESS:Node4:sendreading19$	$to2$
$16:13.306$	$ID:2$	$PROCESS:Node2:forwardsensor4$	$to1$
$16:13.321$	$ID:1$	$PROCESS:ServergotID=4,value$	$=19$
$17:13.274$	$ID:4$	$PROCESS:Node4:sendreading52$	$to2$
$17:13.307$	$ID:2$	$PROCESS:Node2:slope=0,00sensor$	$=4$

4. **Max-fan-out:** When the computation node received data from 6 different sensors, it correctly handles only 5 sensors. When I manually one sensor node away, its data was replaced by the 6th sensor data after the window time.

5. **Silent-sensor expiry:** After the sensor sent 30 readings, if it remained silence for more than 5 minutes, the data window was cleared. The sensor window reset as expected, ensuring no old or incorrect data stayed in memory.

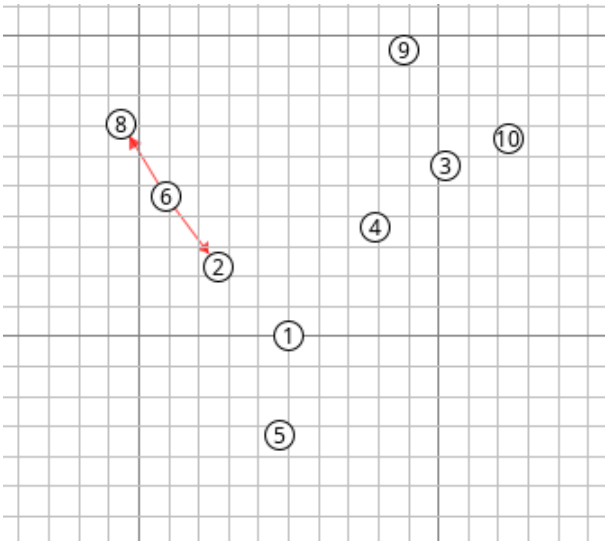
6. **Slope trigger:** With a sensor sending value that increased linearly, the valve command (OPEN_VALVE) was triggered exactly after 30 samples. The slope calculation correctly detect the threshold and activated the valve. Globaly the treshold functionnality is correct

7. **Partition & heal:** After disabling a node in the middle, the affected sensor correctly switched parents (reparented). When the node became active again, sensors switched back, rebuilding the original network tree and resetting the sensor windows again.

I've also tried to test the burst (send a lot of data) for a while and then steady or a steady into burst but I've got issue implementing it.

5 Performances

To compare the two models, we must compare same parameters with the same configuration. Here it is :



In this configuration, 1 is the border router, 2-3-4 are the commputation node and 5-6-8-9-10 are sensor nodes. Here are the results : **Explanation of Results:**

Metric	Unenergised	Energised
HELLO packets (tree management)	2776	5078 (+82.9%)
Sensor readings sent	300	208 (-30.7%)
Readings received by server	60	70 (+16.7%)
Valve openings triggered	30	35 (+16.7%)
Total logged events	5769	5737 (-0.6%)

Table 2: Comparison of key metrics between the energised and unenergised implementations.

- **HELLO packets (tree management):** In the energised version there is significant more HELLO packets (+82.9%) than in the unenergised one. This happen mostly because the node change their state often and they reparent more to optimise the energy usage, resulting in more frequent update of the network topology.

- **Sensor readings sent:** We observed a clear reduction (-30.7%) of sensor reading sent in the energised version. This is due to node entering often into low-power states (LPM or Deep-LPM), which strongly reduce how often the nodes sends data.

- **Readings received by server:** Surprisingly, even if there were fewer sensor readings sent, the energised version had more reading successfully received by the server (+16.7%). It shows that the work is more send to the server due to LPM mode of computation node
- **Valve openings triggered:** We also notice a slight increase (+16.7%) in the number of valve opening events. This probably indicates better accuracy in the slope detection or a reduced packet loss, which allows the energised model to trigger valve commands more consistently.
- **Total logged events:** The total number of logged events (-0.6%) stayed almost the same in both version, showing that the simulation duration and detail of logging were comparable, making our results fair and reliable.

In general, the energised implementation showed a good balance between energy efficiency and performance. It extend the node lifetimes, while keeping good responsiveness and network reliability.