Université catholique de Louvain

Mobile and Embedded Computing

Project report

Group project

CAUDRON Loïc

MOUCHET Antoine

PERREAUX Basile

Github repository:
https://github.com/basilep/contiki_project/

May 2023

1 Introduction

In the context of a building management system, we delve into a hypothetical scenario where a large number of wireless devices, referred to as "sensor nodes", are deployed across various areas of a building. These sensor nodes are equipped with motion sensors that send a message every time someone passes by. Due to the potentially large number of sensor nodes, it becomes imperative to establish an efficient organization and scheduling mechanism to avoid data collisions and ensure smooth data flow. This is where "coordinator nodes" come into play. Coordinator nodes serve as special devices strategically positioned within the network. Their primary role is to schedule the transmission of data for a set of sensor nodes, ensuring that only one node is allowed to send its data at any given time. To put the procedure into more concrete terms, the border router assigns to each coordinator node a particular time-slot. The coordinator node notifies the sensor nodes within its specified area that it is their turn to relay data within their allocated time window. The border router is then used (as a gateway) by the sensor nodes to relay their individual counter values to the external server. A fair and well-organized data transmission process is maintained throughout the network by moving the time-slot to the next coordinator node after the transmission for the current time-slot is complete. In the upcoming sections of this report, we will delve further into the intricacies of the routing algorithm, the time-slot assignment and the server.

2 Routing

For the routing part, we chose to take a *storing mode*, which means that each nodes will have its own routing tables. The addresses of its neighbours (parent and array of children) will be stored as ***linkaddr_t***, making a tree. In the node structure, we also store an integer for each neighbour to manage the availability of parent and children.

We used nullnet single-hop unicast and broadcast to manage communication between nodes. The data sent between nodes is a structure, containing a *step_signal*, the *rank* of the node which sends a message, the random data sent from sensors, the clock and a time-slots array (except for sensors). The nullnet buffer is then pointing to the data structure which allows to send whatever we want from the structure. We use an *input_callback* to manage response whenever a node receives a message.

Three *ctimer* are used to call four functions:

- ***get_in_network***: used to send broadcast connection request.
- ***get_node_availability***: check the availability of neighboring nodes, using an integer number for each. Depending on a threshold, a node may be unreachable and therefore removed from the routing table.

- ***send_clock_request***: send a request to each of the children of the border router to ask them to send their respective clock.
- ***get_sensor_data***: to get data from sensors.

For the routing in itself, when a sensor node arrives on the network, it checks if there is coordinator, if yes it takes the coordinator with the better strength signal (rssi) as parent else it takes the sensor with the best rssi as parent.

We assume that the border router (the root) and the coordinators are always on the network and never down but new sensors can arrive and/or disappear. The border router has the rank 0, coordinators, directly link to the border router, have the rank 1. Sensors have a rank 2 if directly connected to a coordinator, or > 2 if connected to a sensor (a rank is the rank of the parent +1).

2.1 Step signals

The data sent using the nullnet buffer contains a step signal (*uint8_t*) to manage responses according to the data received.

SGN 0 : Connection request

Whenever a node arrives on the network it sends a ***broadcast*** request. For coordinators, just the border router will respond, for sensors, each node in their range (except the border router) will respond with a signal step set to 1.

SGN 1 : Connection response

For coordinators, the response is accepted in any case as just the border router can respond to the connection. For sensors, if the node is not already on the network, it accepts the response and enters in the network. If it receives a second response, it first compares the current rank and new rank (to check if it's a coordinator), if the current parent is a sensor and the response comes from a coordinator, it changes directly the parent, if it's 2 coordinators, it compares the rssi of the response with the rssi of the current parent and take the best one, same process is used for 2 sensors node. A step signal set to 2 is sent to the new parent to acknowledge the connection. If the parent changes, a step signal set to 3 is sent to the previous parent to notify him to remove the child (from its array of children).

SGN 2 : Acknowledgement the connection

When a node receive an acknowledge, it adds the new node to its array of children.

SGN 3 : Remove children

When a node receives a demand to remove a child, it means that one of its children has found a better parent, thus it remove it from its array of children.

SGN 4 and 5 : Check node availability

To manage the availability of the nodes in the routing table, a sensor sends a step signal set to 4 to its neighbours node every time the ***get_node_availability()*** function is executed (every *CHECK_NETWORK* interval). When a node receives this signal, it just responds with a step signal set to 5 to aware that it's still

reachable. Coordinator also check the availability of their children (but not the parent obviously).

SGN 6, 7, 8 : Clock request and Berkeley algorithm execution

In order to request the clocks of each of its children to perform the Berkeley algorithm, the border router sends a step signal initialized to 6 each time the ***send_clock_request()*** function is executed (every BERKELEY_INTERVAL interval). When a coordinator receives this signal, it responds by sending its requested clock and the step signal 7 to indicate that it is sending its clock. Once the border router has retrieved the set of clocks from its children, it executes the Berkeley algorithm to obtain the new synchronized clock and sends it to its children with the step signal 8 to indicate that it is the new clock they should use.

SGN 9 : time-slots allocation The border router sends a time-slot array [begin_clock, end_clock] to each coordinator where they will be able to send data from the sensors to the border router. The allocation of time-slots is made using a fixed *TIME_WINDOW* and a synchronized clock (with the Berkeley algorithm).

SGN 10 : rank change When a sensor change its parent it may change of rank, this signal is used to notify the children of the node to also change the rank.

SGN 11 and 12 : sensors data When a coordinator is in its allocated time-slot, it uses polling to get data from sensors. Every *TIME_WINDOW/10*, a step signal set to 11 is sent to its children, the children then calculate a random number between 1 and 5 to know if they have to send data. If they send data (1 in 5 chances), a new number between 1 and 100 is chosen randomly. This data is sent to the parent, using a step signal set to 12 (when a node gets something on signal 12, it directly sends the data to its parent, allowing the data to go to the root). Every sensor also sends a signal set to 11 to notify their children that they can send data.

3 Berkeley Time Synchronization Algorithm

We have chosen to implement a Berkeley algorithm in its simplest form. Indeed, we have, for ease, changed the way to update the synchronized clock of the coordinators by using a variable storing a clock compensation. We also did not take into account the propagation time of the messages with the Cristian's algorithm. Here are the different steps of the implemented Berkeley algorithm:

1. The border route, considered as the master node, sends a clock request to each of its children (coordinator) thanks to the variable $step\_signal = 6$.

2. Each coordinator responds to this request with its current clock if it is not yet synchronized or the previously synchronized clock with the variable $step\_signal = 7$.

3. The border router retrieves each clock and as soon as the number of clocks is equal to the number of its children, it calculates the average time difference between its clock and the received ones, symbolized by the variable

clock_compensation. It updates its own clock with this clock compensation and sends the new synchronized clock to each of its children (coordinator) with the signal *step_signal = 8*.

4. Each coordinator, instead of modifying their clock, calculates the compensation they have to take into account with respect to their own clock to arrive at the synchronized clock. This compensation will be used every time a clock request is received or for the assignment of time-slots.

We are aware that the main improvement would be to implement Cristian's algorithm where, in this case, we would not just send the request with *step_signal = 6* but also the current clock of the master so that the children can respond with this clock as well as their own clock so that the master can take into account the propagation time which potentially causes a slight drift of the clocks.

4 Server

The server establishes a connection to the border router using sockets. The server tries to connect to the IP and port given when launching it. Therefore, those should be the IP and port of the border router. Within the server, there is a global dictionary that serves as a storage for node counters. Each key in the dictionary represents a node, and the corresponding value is its counter of people. Whenever the server receives a message containing data regarding a node and its counter, it updates the global dictionary.
There is the possibility to add an argument "save" when launching the server. If it is specified to true, then the server will try to load the dictionary from the file "global_counter_save.json" on start-up and it will save the dictionary to this file if the connection is lost.

5 Message format

The message sent from the node (through the border router) to the server is built as follow : "magic2023-[node id],[node counter]". Each node has a different ID which allows us to uniquely identify it. Moreover, each time the node sends data it resets its counter[1]. The "magic2023" part is useful because it allows us to filter the useful data from the rest.

6 Notes

To be able to communicate with more than two children, you need to change the file *contiki-ng/os/net/mac/csma/csma-output.c* at the ligne 112 ***#define CSMA_MAX_NEIGHBOR_QUEUES 4***, here we change the number to 4 (default is 2) as the assistant said that we can assume that the node will not have more than 4 children.

[1] Yes, it is a very busy building. The counter is generated randomly.