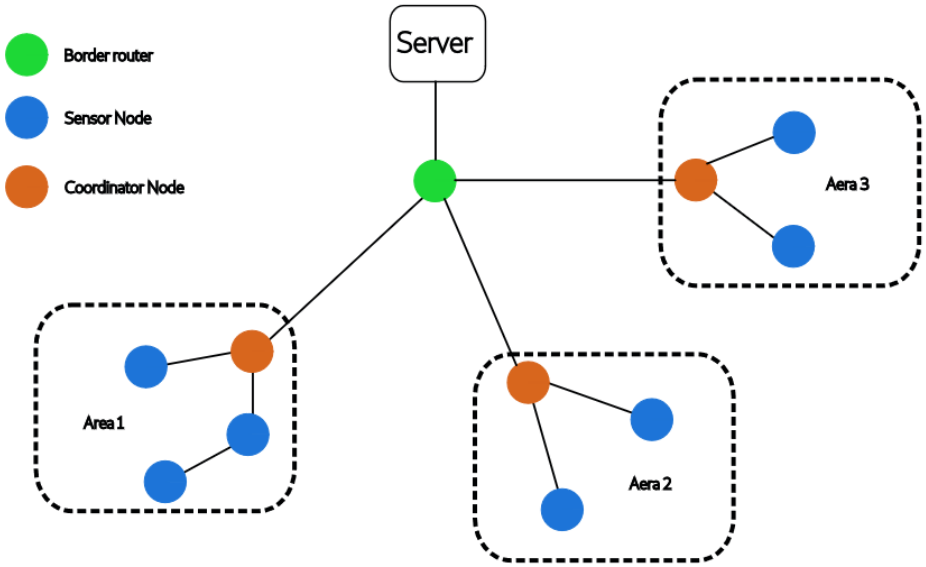
École Polytechnique de Louvain

Projet Report

Projet LINFO2146: Building management system Cooja & Contiki



Membres du groupe:
Alexis Picquet 12182201,
Brieuc Pierre 55481800,
Silya Rachik 65552100,

15-05-2020

1 GitHub repository

`https://github.com/alexispicquet/LINFO2146`

2 Scheduling

2.1 Berkeley Time Synchronization algorithm

For the coordination of the border router and the coordination node we choose to implement the Berkeley Time Synchronization algorithm and the time slot scheduling between the border router and the coordinator nodes.

1. Initialization: The border router has a reference clock. Each computer in the network maintains its own local clock.
2. Exchange of Time Information: The border router periodically sends a request to each coordinator in the network, asking for their current time. The coordinator respond by sending their local time to the border router.
3. Calculation of Clock Offsets: The border router collects the time responses from all the coordinator and calculates the clock offset or time difference between its own clock and the clocks of the other coordinator. The clock offset represents the drift or difference in time between the coordinator's clock and each border router's clock.
4. Adjustment: The border router determines the average clock offset by taking the average of all the individual clock offsets. It then sends the average offset to each coordinator in the network.
5. Clock Adjustment: Upon receiving the average offset from the border router, each coordinator adjusts its local clock by applying the offset value. This adjustment brings the local clocks closer to the border router's clock.
6. Iteration: The algorithm typically runs in multiple iterations to refine the clock synchronization. In each iteration, the border router recalculates the clock offsets based on the adjusted clocks of the coordinator. The process of exchanging time information, calculating offsets, and adjusting clocks is repeated until the desired level of synchronization is achieved.

2.2 Polling scheduling between the coordinator node and the sensors

1. Create a list of messages : First, you need to have a queue of messages that need to be sent to the coordinator node for every sensor nodes.
2. Start the polling loop: The scheduling algorithm starts a loop that continues indefinitely until all messages are sent for every sensor node. Inside this loop, each messages is sent sequentially.
3. Move to the next node: After that all messages are sent, the scheduler moves on to the next node in the list.
4. Repeat the loop: The scheduler continues the loop, repeatedly checking the status of each message that need to be sent until all messages are sent.

3 Message format

The messages are defined as PACKET_T objects containing the following fields:

- ▷ UINT8_T status → indicates which kind of message is being sent: SEARCH for a normal broadcast, PARENT to ask a node to become its parent, DATA to request a sensor's data, ..._ACK for the response to a message. ..._ACK will not require any acknowledgment as this logic could be applied endlessly.

- ▷ UINT8_T type → indicates whether the sender is a coordinator or a sensor node. It is necessary to be able to deal with the message properly.
- ▷ UINT8_T rank → represents the layer of the sender node within the current network (i.e. rank = 2 means the sender is 2 links away from a coordinator node and rank = 0 means the sender is a coordinator node). Without this rank, two sensor nodes could enter a deadlock where each of them asks the other to be his parent. The rank only matters between sensor nodes.
- ▷ UINT8_T rssi → the receiver node stores the signal strength of the message. It indicates the distance between 2 nodes and is used to select the best node as parent.

Messages are stores in lists defined as PKT_LIST_T; (or STRUCT LIST) objects. They are linked lists instead of arrays as the size will depend on the network and because memory usage would be an important criteria for small iot devices. They contain the following fields:

- ▷ PACKET_T* head → points to a message stored (and allocated).
- ▷ STRUCT LIST* next → points to the next message stored if applicable (i.e. a linked list).
- ▷ LINKADDR_T src → stores the address of the node that has sent this message. This is required to be able to send messages back.
- ▷ UINT8_T counter → is used to indicate the number of elapsed timers since it (the node possessing this list) has received a message from the node with the address stored in src.

Two kinds of linked list are used:

neighbours is an ordered list. The index (or position) of a message in a list indicates its potential to become a parent for the node with that list. The lower the index the better the candidate. A node will always try make the best candidate its parent.

tp_ack and kids are unordered lists. The former is used to remember the messages that have been received recently and might trigger events. The latter is a list of the current kids (opposite of parent) of the node with the list.

Both of these list do not allow for 2 messages inside them to have the same address hence the same node. This avoids sending multiple message to the same node at once. It is always sufficient to send a single message with the appropriate field values. For example is a sensor node receives a broadcast (type SEARCH) and a parent request (type PARENT) from another sensor node, it can respond by sending a single PARENT_ACK to accept the parent request.

4 Routing

Initially every node has no knowledge of the network. For this project, the coordinator nodes (CN1) are responsible to make themselves known to their neighbours via a periodical broadcast.

Every sensor node (SN1) receiving this broadcast will attempt to make the coordinator sender his parent (CN1 SEARCH → SN1 PARENT → CN1 PARENT_ACK). If the parenting is successful the node is now situated inside the network (SN1 is the kid of CN1 and inversely CN1 is the parent of SN1).

When a sensor gains knowledge of where he is inside the network, it will broadcast his rank to allow sensor nodes (SN2) to send him parent requests (if SN1 is currently the best candidate for SN2).

Every broadcast is acknowledge so that nodes are aware of their neighbours (bidirectional knowledge).

Any relation (neighbours or parent $\leftrightarrow$ kid) with no message sent in 10 elapsed timer will be removed. The former doesn't trigger any event. The latter will trigger new parent request from the kid node. The removal of neighbours is commented out in the implementation as it caused too much interference as to test small cases (and because we don't check for corrupted messages).

The choice of 10 elapsed timer is to reduce the amount of overhead messages potentially causing many interference.

5 Border router and server

This section describes the implementation of a server application that facilitates communication between nodes in a wireless sensor network. To establish communication between the border router and the server application, we need to create a *SerialSocket* on the bridge node within the Cooja simulation. This Serial Socket acts as a bridge or intermediary between the border router node and the server application. The server application, establishes a connection to the port 60001 of the Serial Socket created in the Cooja environment. This connection allows the server application to send messages to the border router node by writing to its serial interface and receive messages from the border router node by reading from its serial interface. This figures represent the connection between the server and bonrder router.

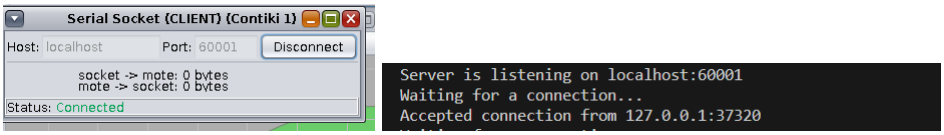


Figure 1: server and border router connection