UCLouvain

ÉCOLE POLYTECHNIQUE DE LOUVAIN

LINGI2146 : Mobile and Embedded Computing Project Report

De Keersmaeker	François	7367-1600
Gobeaux	Alexandre	4219-1600
van de Walle	Nicolas	2790-1600

https://www.github.com/fdekeers/LINGI2146

Louvain-la-Neuve
May 18, 2020

Introduction

The context of this project is a network of small sensor devices in a building, that measure air quality. These devices organize themselves into a tree network following a routing protocol described later. When they are installed in this network, they send the measured value once per minute to a server, connected to the network by a border router node that is the root of the tree. The server then analyzes this data, and can ask to the devices to open a ventilation valve. To avoid too many network traffic, some computation nodes are present in the network. When they receive data from a sensor node, these computation nodes can do the analysis and the request to open the valve themselves (if their maximum capacity is not reached).

This report is divided in 4 parts. The first one will describe the messages that are exchanged in the network and with the server. Then, the routing protocol used by the nodes to form the tree will be explained. After that, the functioning of the computation nodes will be described. Finally, the server and its communication with the border router are detailed.

1 Message format in sensor network

There are 5 types of messages in our network :

- DIS messages : routing messages, broadcasted by new nodes to request information from nodes that are already installed in the tree network;
- DIO messages : routing messages, broadcasted by nodes that are already in the tree to give information to actual and potential children;
- DAO messages : routing messages, sent by nodes to their parent, to inform them that they are now one of their children, or as keep-alive messages;
- DATA messages : sent once per minute by all sensor nodes in the network towards the server (they contain the air quality value, a random number in our simulation);
- OPEN messages : sent by the server or a computation node to a sensor node that needs to open its valve for 10 minutes.

The usage and sending policy of the 3 first types of messages will be described in more depth in the following section about the routing protocol.

For simplicity, we decided to use a different structure for each type of message, keeping only the necessary information for each type. This lets us use these structures in the cases where a precise type of message is needed, and also avoids to carry useless information in network packets, which is an advantage in a resource constrained network. The drawback of this design choice is that we need to define more structures, which induces a slightly bigger source code, but since each of the structure has no more than 3 attributes, this is not a big problem.

All types of messages start with a number, encoded on 8 bits, that represent the type. This number has the value 0 for DATA, 1 for OPEN, 2 for DIS, 3 for DIO, and 4 for DAO messages. The smallest number of bits to encode these 5 types is thus 4, but we chose to use integers on 8 bits because they are easier to program with, and are the shortest implementation of integers provided in the C language. In addition to this type number, the content of the message varies depending on its type :

- DIS : nothing else, this type of message does not need any other information.
- DIO : the rank of the sending node, an integer on 8 bits.
- DAO : the address of the node that sent it, encoded on 16 bits.

- DATA : the address of the node that sent it, encoded on 16 bits, and the random value used to simulate the measure of the sensor node, an integer on 16 bits.
- OPEN : the address of the destination node, encoded on 16 bits.

The maximum size of the messages exchanged in the network is thus 40 bits, which is really short and suits this kind of lossy and resource constrained network.

2 Routing

The routing protocol used by the nodes to build a tree-shaped network is based on RPL, with some simplifications. Unlike RPL, a node can only have one parent, and some mechanisms for repairing the tree are not present (like the global repair with new versions of the tree). This section will describe the protocol in detail.

At the beginning, only the border router node is present in the tree, as its root. It sets its own rank to 0. The rank represents the number of hops to this border router, and will be used by other nodes in the network to avoid cycles, by not being able to choose a parent that has a rank greater than or equal to their own rank. When a new node arrives in the network, it has a rank of value `INFINITE_RANK` (i.e. 255). It periodically sends DIS messages, with a short period. These messages are broadcasted to request information about a potential parent in the communication range. When another node receives a DIS message, it ignores it if it isn't itself already in the tree. Otherwise, it broadcasts a DIO message, containing its rank, to inform the requesting node. The node that is not in the tree will then choose among all the potential parents, first by choosing the one with the lower rank, and if several nodes have the same rank, by choosing the one with the best signal quality. It then sets its own rank by incrementing by one the value of the rank of the parent it just chose. We implemented this policy so that a node always chooses the parent with the smaller number of hops to the border router. Finally, the new node sends a DAO message in unicast to its new parent, to inform it that it just attached itself, so that the parent knows that it can directly reach this node. Then, this DAO message, containing the address of the child, is forwarded towards the root, to populate or update the routing tables of all the nodes on the path (those are implemented with a hashmap, to have constant access time). Each node stores in its routing table the address of the child, the address of the next hop node towards this child, and the timestamp of the DAO message. Every new node in the network does this procedure to join the tree.

When nodes are in the tree, they send periodic DIO messages in broadcast, and DAO messages in unicast to their parent. These messages are used by children to update information about their parent (rank and signal strength), and as keep-alive messages so children don't think their parent is lost, and vice-versa. DAO messages are again forwarded towards the root to update the routing tables. If the rank of the parent has changed, the node updates its own rank by adding 1 to the parent's rank, and then directly broadcasts a DIO message to forward the information to its children, and so on. Moreover, when a node receives a DIO message from another node than its current parent, it may decide to change its parent for the new one, if the new one has a lower rank than the current parent, or the same rank but a better signal strength (over a certain threshold, to avoid non-stop parent switching in fluctuating networks). In this case, the node updates its rank in the same way, and sends a DAO message to the new parent, just like it would do if it was a new node in the network. It also sends a DIO message with its possibly new rank to its children, so that they can update their parent's information and perhaps send a DIO message themselves, and so on.

It is possible that nodes are removed from the network. In this case, they should be removed from all routing tables of nodes on the path towards the root, and the children of these removed nodes should detach from the tree to seek another parent. To this end, 2 timers are set in each node: one for its parent, and one for its children. The first one is started as soon as the node joins the tree, and restarted each time it receives a DIO message coming from its parent. When it expires, this means that the node hasn't received a message from its parent since a long time, and thus that the parent has probably been removed from the network. In this case, the node detaches itself from the tree, by resetting its rank to `INFINITE_RANK`, its routing table, and deleting its parent. It then broadcasts a last DIO message with this `INFINITE_RANK` to notify its children that it has detached itself, and that they should do the same. In the possibility of the loss of this message, the children would either reattach to the same node, when it is

back in the network, or themselves wait for the expiration of their parent timer, and undergo the same procedure. To reattach to the network, a node simply does the same as if it was a new node, i.e. it sends periodic DIS messages to find a parent. The second timer, used to delete unresponsive children from the routing table, triggers every 2 min 30 the deletion of every children that did not send a DAO message during this period from the routing table. This prevents nodes from keeping useless information in memory.

To lower the traffic in the network, we use a trickle timer, as described in the course, to determine the sending time for the periodic DIO and DAO messages used for keep-alive. This timer works as follows : the timer has an initial period T, set to 2 seconds. A random duration is chosen between T and $T/2$, that determines the time after which the message will be sent. At each DIO message sent, T is doubled, up to a maximum value `T_MAX`. This induces the messages to be sent less and less often. However, each time there is a change in the network, T is reset to its initial value. This happens in the following cases :

- When a child is removed from the routing table, i.e. when the node has not received any DAO message from this child before the timeout of the children timer;
- When the node changes something about its parent, i.e. when the node receives a better DIO message from another potential parent and chooses to switch to this new parent, or when the node needs to update the rank of its current parent upon reception of a DIO message from it, advertising a new rank;
- When the node detaches itself from the tree, i.e. when it has not received any DIO message from its parent before the timeout of the parent timer, or when it receives a DIO message from its parent advertising an `INFINITE_RANK`, which means that the parent itself has detached from the tree;
- When a new child is added, i.e. when a DAO message is received from a new child.

Unlike the implementation seen in the course, there is no cancelling of messages since these periodic DIO and DAO messages are needed as keep-alive messages to prevent the parent and children from thinking the node has been removed from the network.

A lot of different timers and delays are used throughout the routing protocol. This last paragraph will sum these timers up, as well as explain the values we chose for them. The sending period of data values by sensor nodes is fixed at 1 per minute in the requirements. We added some randomness to this period, by choosing a random delay between 55 and 65 seconds, to avoid collisions between DATA messages coming from different nodes. Other timer values are based on this first one, to avoid sending unnecessary DATA messages, but still cope with the possibility of loss in this kind of networks. The maximum value for the trickle timer for DIO and DAO messages, `T_MAX`, is set to 20 seconds. Then, the parent timer, that deletes a parent that didn't send a DIO message before its expiration, is set to 50 seconds. In this way, up to 3 DIO messages from the parent can be lost in average before a node deletes it. The parent timer is slightly less than the data sending period, so that sensor nodes don't send DATA messages to a parent that has been removed from he network. Finally, the timer that deletes an unresponsive child from the routing table is set to 2 min 30. This timer doesn't have an impact on the sending of messages, and this operation is costly (all the routing table has to be processed), so we don't want to do it too often.

3 Computation nodes

Concerning the computation nodes, only the data plane is different in comparison to the sensor motes. Indeed, they do not need to create DATA packets (since they are not linked to a sensor) and they can intercept DATA packets going to the root to do the computation instead of the server. This lightens the work of the server and can be very useful either if some nodes are far from the root or if the network consists of many nodes. Since the computation nodes also have physical limitations (physical/virtual memory), they can not choose to do the computation for every node that goes through it when sending data to the server. The number of sensor nodes for which the computation node can do computations is thus limited (e.g. to 5).

Regarding the implementation, we used a simple buffer to store the data. The most important elements contained in each entry of this buffer are the address of the sensor mote its data values, and a timestamp to remove inactive nodes. The mechanism used here to remove the nodes is pretty simple. Since knowing what the buffer contains is only useful when we try to add a DATA value, we can remove nodes that timed out (e.g. after 5 minutes) in a lazy way. In other words, we can remove them when we try to check what is inside the buffer instead of calling a periodic cleaning function as it was done for the routing tables. What happens is that when we want to add a new node in the buffer, we first look if there is an empty place in the buffer or if the node is already contained in the buffer. When iterating through the buffer, if we encounter a used cell with a node that timed out, we clear the used flag of the cell. Its data will be erased when another child mote will take this place in the buffer.

Let us now detail a bit more what happens when the computation node receives a DATA packet.

- If the node sending the DATA packet is already contained in the buffer, it adds the new value (potentially erasing an old value if 30 values were already contained) in the values buffer. If enough values (e.g. 10) are already contained in the buffer for this node, the slope is computed and an OPEN message is sent if the slope is above a given threshold (e.g. 0).
- If it is a new node and there is still a place in the nodes buffer, then it inserts the new node along with its first value. In this case, no computation would be done since we chose to wait for at least a given number of values before computing the slope value.
- If the node is not contained in the buffer and it is already full, the DATA packet should be forwarded to the server (and this packet will eventually be caught by the server or another computation node).

Note: when computing the slope, to reduce the memory usage of the nodes, we decided to consider a linear x-axis ranging from 0 to 29 (unlike the server which, as it will be detailed later, takes the packet time into account).

4 Computing server and Border Router

To handle the DATA that have been sent by the sensor nodes and arrived to the border router (i.e. root of the tree), we set up a Python server. This server is responsible of computing the slope of the least square regression of the values sent by the sensors. If this slope exceeds a certain threshold, it sends and OPEN message to the concerned mote. Otherwise, nothing more happens.

Concretely, the border router and the server communicate over a TCP socket (which requires the enabling of the serial socket server on the root mote). As the communication is done using text (printf over the socket connection), the border router dis-encapsulates the DATA packets and sends them as text using the following format: **[MESSAGE_TYPE]/[SOURCE_MOTE]/[DATA]**. [MESSAGE_TYPE] is an integer value corresponding to the type of message (DATA=0, OPEN=1), [SOURCE_MOTE] is the address of the sensor mote, and [DATA] is the data value.

When the server receives this message, it parses it and adds the current timestamp, together with the data value in the list associated to the sensor mote. If it has enough values (i.e. 10 in our case) it computes the slope of the least squares regression of the last (maximum) 30 values depending on the timestamps. If it is over the threshold (which can be given as command line parameter when starting the server), an OPEN message is sent to the corresponding mote via the border router with the following format **[MESSAGE_TYPE=1]/[DEST_MOTE]**.

If we do not receive DATA from a mote for 30 minutes (unlike 5 minutes for computation nodes which have less memory), we remove it from the server to spare memory.

To run the server, open a terminal and navigate to the `server` folder. Then, run the following command:

path/to/server/: `python3 server.py 127.0.0.1 [PORT OF ROOT] [THRESHOLD]`

The port corresponds to the one displayed when starting the serial socket server in Cooja. The threshold is the minimum slope value to send an OPEN message to the sensor motes (set to 0 as default).