# LINGI2144: Project 1

Perdeans Olivier(23821600), Saley Abdou Djafarou(24031600),
Marini Mohamed Samir (01271500)

29 Mai 2020

## 1 Routing of our network

To understand how our routing works, we gave each node different attributes, allowing them to efficiently communicate with their neighbours.

Each node keep in memory it's **parent node** - it's address and it's distance from the server - (expect for the border router, who's parent is the server himself) and it's **distance from the server** which is computed only once, when the node is added to the network.

### 1.1 Initialisation of the network

When a node is started, it first start 2 different processes to find a parent node, compute its distance from server and once this phase is done, this node starts broadcasting 'hello' messages all around him.

#### 1.1.1 Broadcast hello messages

The first process, *broadcast_routing*, establishes the connection to our routing communication channel and sends periodically (every 2 minute) hello message if the node is already connected to the rest of the network and has a path to the server.

#### 1.1.2 Receive hello messages

The second, *recv_hello*, allows the node to receive "hello" messages from other nodes that are already connected to the server. These hello messages are sent by the first process.

During 4minutes, this process allows the node to receive and store all hello messages that are broadcasted around him. Once the timer expired, the node can choose it's parent according to the signal strength of all messages it received (the higher the signal is, the greater are chances for a sender node to be chosen as parent node). Before jumping to the next step, the node has to send a message to it's newly chosen parent containing it's address and the fact that he chose the other node as it's parent. That way, the parent is now aware of having children and knows the address it can use to reach them.

Once the parent is chosen, the node has to compute it's distance from server. Which is quite trivial : the node sets it's distance from server at the same distance of it's parent, increased by 1.

The node is now connected to the network and has a single path to the server. Thanks to the first process mentioned before, it will also start sending hello messages around him to allow other nodes to find a connection to the server through him. We also will start the process in charge of handling data sending (if the node is a sensor node).

## 1.2 Data sending and receiving

This section only concerns the sensor nodes. Each one of them has a specific process handling data sending every minute (*runicast_ data_ process*). This process is started only when the node is connected to the rest of the network and stopped when he lost it's connection to it's parent.

Every minute, the sensor node compute a random value, which is sent to it's parent node in order to reach the server or a computation node.

When a sensor node received data from one of it's child's, it change the "forwarded" byte to true, and then forward the received data to it's parent, in order to reach the server.

## 1.3 Action received from server / computation node

When enabling the data process we just described, allowing the node to send and receive data messages (we will describe later the different formats of messages we used), we start another process, allowing the node to received action messages (this is true for both sensor nodes and computations node). The purpose of action messages, is to ask a sensor node to open it's valve. We will now see how the two type of nodes handles the reception of such a message.

Since action message are broadcast, we need to analyse the sender of the broadcast to know if the message received needs to be forwarded again to reach the destination node.

### 1.3.1 Compute node

The node only process action messages that are broadcast by a node with a lower distance to server. This ensures that there is no "infinite loop" inside the network. If the message satisfies that constraint, the computation node will also forward the message around him.

### 1.3.2 Sensor node

As for the computation node, the sensor node will only process action messages that are broadcast by a node with a lower distance to server.

The sensor node then checks if the action message is addressed to himself.
— If yes, the device will launch the process in charge of opening the valve (in this theoretical case, light the LED's up for 10 minutes).
— Else, the node will broadcast the action message in order to reach is destination node.

## 1.4 Loss of parent and resetting of the network

The system allowing a node to retrieve a new parent and notice that it's parent has died is the same for sensor and compute nodes.

### 1.4.1 Detect loss of parent

The detection is trivial. When sending/transferring data to the parent's node, in case the sending timed out, this means that our parent node is down and our "path" to the server is not valid anymore.

### 1.4.2 Get a new parent

At this time, we just start the recv_hello process, allowing to receive hello messages send periodically by all nodes having valid path to the server.

The main idea is exactly the same as explained in previous section expect that we doesn't re-compute the distance to the server. Once a new parent is found, data process and action processes are started again and the node can work again "normally".

### 1.5 Delay added to timers

In order to spread the load in the network, we placed a random timer (between 1 and 30sec) that shifts the start

## 2 Message format in our network

We use a total of four types of message format, each of which is exchanged on a dedicated communication channel.

```c
    struct BROADCAST_ROUTING {
      linkaddr_t addr;
      int dist_to_server;
    };
```

```c
    struct RUNICAST_ROUTING {
      linkaddr_t addr;
      bool isChild;
    };
```

**BROADCAST_ROUTING** messages which are exchanged on channel 129 :
These are indeed messages regularly sent by a source so that new arriving nodes can be integrated into the network by choosing a suitable parent based on the dist_to_server.

**RUNICAST_ROUTING** messages which are exchanged on channel 144 :
This message is sent by a child node to its parent node to specify that he has chosen him as new parent. It will allow the parent to update his child list table.

```c
struct RUNICAST_DATA {
  linkaddr_t addr;
  bool forwarded;
  int min;
  int val;
};
```

```c
struct BROADCAST_ACTION{
  linkaddr_t dest_addr;
  int dist_to_server;
};
```

**RUNICAST_DATA** messages which are exchanged on channel 164 :
Structure containing the data sent by the sensor nodes. This packet will be systematically sent to the parent.

**BROADCAST_ACTION** messages which are exchanged on channel 139 :
These messages are used to specify a node in the network to open its valve for 10 minutes

## 3 Computation Node, how it works ?

The computation node find it's parent thanks to the same processes as the sensor nodes (explained before). We will here explain how the computation node stores and computes the value received from maximum 5 sensors.

To achieve this goal, we implemented a table of pointers (size 5). Each entry of this table is a pointer that refers to the first data received from a certain sensor node. Those data are stored in a linked-list which size will never exceed 30 (since we compute those data every 30 measurements received). To keep in memory which index of this table corresponds to which sensor node address, we used a table of node addresses. The index of the address in this table is the same index used in the other table to store the data received from this node.

When receiving a data, we first check if the node at the origin of the data is known from our table. If yes, we add the new measurement to the correct list and then we check if the size of this list is equal to 30. If that's the case, we compute the slope of the least square line, compare it to our threshold and send action broadcast if that's necessary . We then empty the measurement's list for this node.

If the node is not in our table yet two cases are possible.

— The table is already full (we have measurements for 5 nodes already), then the data is simply forwarded to the parent of the computation node in order to reach the server.
— The table has available place, we then add the address to our address table, and add the measurement received to our measurements table at the right index.

In order to remove measurements of dead nodes and make place for other sensor nodes, every time we received a data that's added to our measurement's table, we update the stored timestamp for the address of that node. We build a process that checks every 2 minutes that this timestamp has been updated at least once. If that not the case, we delete all measurements received from that node, and make place in our table for another node!

## 4 Implementation of the server

As the border router uses Contiki and Contiki has a generic interface for line-based serial communication we took advantage of it and use it to communicate with the server which uses a socket to open a communication towards the serial socket. We made the server in python in order to benefit from the high level functionalities that the language offers.

First, the border router retrieves the data that the sensors or computational nodes send to it, and transfers that data to the server through a message in the following form :
DATA : `[source_address], [minute], [sensor_value]\n`
Example : « `DATA : 5.0, 2, 29\n` »

These messages are received by the server byte by byte, so the server waits for the end of line character `'\n'` to retrieve the entire message. Then, the server calls the *handle_line* method which will extract the received data from the message by matching the received line with a regex we defined beforehand.

We then store this data and wait for the reception of 30 samples from the same source. Once the 30 values have been received, we remove them from the memory and we calculate the slope using the method of least squares, if this slope is higher than a certain threshold, we send a message to the border router which will request the relevant sensor to open its valve.

We decided to send a message to the border router only when the valve has to be opened. This decision allows us to use a very simple message format from server to border router which is as follows :
`[Address of the node that needs to open its valve]`
`Example : « 5.0 »`

## 5 Implementation choices

Delaying the data transmission between nodes was decided in order to spread the load on our network. The use of reliable unicast for the data transfer from the sensors is critical as the server/computational nodes needs this data to instruct the sensors whether or not to open the valves.

Network discovery messages is implemented with broadcast since in case of possible lost, the receiver simply needs to wait a little bit more for the other wave of messages without any impact on the existing network.

We've chosen a linked list to keep track of children's nodes in order to avoid any buffer overflow.