École Polytechnique de Louvain

LINFO2146 – Mobile and Embedded Computing

Project Report : Smart Greenhouse

Members

Maxime Delacroix - 31632000
Antoine Deleux - 37422000

Contents

1 Introduction

This report chronicles the development of a custom routing protocol designed to connect Internet-of-Things (IoT) devices within a simulated greenhouse environment. This project was undertaken for the Mobile and Embedded Computing course at UCLouvain. All the simulations were made using the Cooja software, where the different devices were simulated using Z1 motes.

The chosen network communication framework, Nullnet, while offering basic functionalities, presented some limitations. This report delves into the minor hurdles we encountered with Nullnet and the workarounds we implemented to ensure the network functioned seamlessly. We'll then delve into the design of our routing protocol, including the various packet structures we developed to facilitate communication. Finally, we'll explore how the entire system interacts with an external server, completing the picture of our simulated IoT greenhouse network.

The code we implemented for the various node simulations can be found in our github at the following URL : `https://github.com/MxDx/LINFO2146-Project.git`

2 Nullnet

2.1 Utilisation

To utilize the antenna in the Z1 mote we use the nullnet API of contiki, this API allows us to send broadcast packets and unicast packets.
The broadcasted packets are received by all the neighbours in range of the antenna, a protocol we use to connect a node to the network.
The unicast packets are typically used for direct, one-to-one communication. Unicast packets are sent from a single source node to a specific destination node. This is particularly useful when a node needs to send data to a specific neighbor, rather than broadcasting it to all nodes within the antenna's range. This can help to reduce network congestion and improve overall efficiency. The unicast packets are used once the setup procedure is done to send packet directly to specific neighbours.

2.2 Problems

Unfortunately the nullnet implementation has some issues with the unicast functionality. The unicast packets can be changed to broadcast packets when a lot of motes are in range, we did not find any reason why this behaviour occurred but we saw online that other people were encountering the same problem.
After realising that we could not rely on the unicast packets to always be sent to the right node, we chose, as a workaround, to add a redundant packet destination inside our packet format, see section 3.

3 Packet format

As mentioned previously, we can not rely on the unicast of the nullnet API to reliably send a packet to its destination. Below is the packet structure that is in front of every packet in our network to ensure that the node that receives the packet is the intended destination. You can see the structure of the start of each packet on Listing 1.
Then, we designed our system to work with 2 main types of packets : control packets and data packets. The control packets are used mainly to : establish connections, build the routing tree and send ACK packets. The data packets are used mainly to : send sensor data, send device

commands (e.g. turn the lights on), send keepalive packets and do the communication between a mobile device and sensors.

```
.
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                              src                              |
        +                            (8 Bytes)                          +
        |                                                               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                             dest                              |
        +                            (8 Bytes)                          +
        |                                                               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                         network_packet                        |
        +                              ...                              +
        |                                                               |
.
```

Listing 1: Base packet format

- **dest:** In the sending node the dest field in set to the nexthop in the network, so the receiving node can verify if it is supposed to process it. When a packet is forward in the network the dest field needs to be updated, otherwise it will be discarded.
- **src:** The src field is set by the original sender of the packet, so the other nodes in the network can send responses to the originator.

3.1 Control Packet

```
.
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |T|NTP| CTP | E |                 data (Optional)                 |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.
```

Listing 2: Control packet format

- **T:** First flag used to determine if the network packet is a control packet or a data packet. For the control packet T is set to 0.
- **NTP:** The node type flag is used to identified the type of the originator. This flag is encoded on two bits, because there is 4 type of node in the network, gateway, sub-gateway, node and mobile.
- **CTP:** The control type flag is used to identify the different type of control packets. The flag is encoded on 3 bits, the different types are explained in section 4.
- **E:** This part of the header is currently empty and could be used to improve our control packet later.
- **data:** The data in the control packet is optional, some of the response type require additional information to be sent along the control packet. The len of the data does not need to be sent along because the nullnet API give us the total length of the packet when transmitting.

3.2 Data packet

```
.
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |T|U|  MGP  |MFL|           LEN_TOPIC           |    LEN_DATA   |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |    LEN_DATA   |                                               |
        +-+-+-+-+-+-+-+-+                                               +
        |                         dest (Optional)                       |
        +                                               +-+-+-+-+-+-+-+-+
        |                                               |               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               +
        |                             topic                             |
        +                              ...                              +
        |                                                               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                             data                              |
        +                              ...                              +
        |                                                               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.
```

Listing 3: Control packet format

- **T:** First flag used to determine if the network packet is a control packet or a data packet. For the data packet T is set to 1.
- **U:** Up flag. Determines if the data packet is climbing (1) or going down (0) the tree.
- **MGP:** Multicast Group flag. Indicates in which multicast group the packets needs to be forwarded. This is set to 0 for unicast communication.
- **MFL:** Mobile Flags. Used only when a mobile device (phone) is connected to the network. Set to 0 most of the time.
- **LEN_TOPIC:** Length of the topic field.
- **LEN_DATA:** Length of the data field.
- **dest:** MAC address of the ultimate destination of the packet. Mostly used when a data packet goes down the tree.
- **topic:** Name of the category to which the data belongs (e.g. "irrigation").
- **data:** Data content of the packet.

4 Routing

The routing is structured as a tree where every node has a unique parent and can have one or multiple children. The top of the tree is the gateway and can only have sub gateways as children.

4.1 Connection establishment

When a node is starting or has lost its connection to his parent, it starts a setup process. While the node has not found a parent, it will broadcast a control packet with the control type flag set to **SETUP**.
The neighbouring nodes that receive the packet will respond with a **RESPONSE** control packet according to the conditions below.

- The node has to be already attached to the network tree to respond (the gateway is always attached).
- The gateway responds only to the sub-gateways and sub-gateways respond only to nodes

When the response arrives to the node requesting connection, it will use the decision process described in the the project statement (e.g. choose a parent according to connection strength). When the parent is chosen, the node will keep it in memory and send a **SETUP_ACK** control packet to it with its multicast group and its MAC address in the optional data.
The node that receives the **SETUP_ACK** control packet will add this new child in its children routing table, along with the address of the child, the address from which the child has been learned and the multicast group of the child. When adding a child in the table, the node checks if the child address has not been seen before, if it has, it will send a **CHILD_RM** to the previously known route to remove the child from the old routing tables (because a newer path has been found).
Once the child is added, the parent will forward the child information up the tree, so that all the nodes above it in the tree can add the new route to their forwarding table.
Gateway specificity The gateway will attribute a barn number to all its sub-gateways on their first setup in order to identify the different greenhouses. A sub-gateway that loses its connection and try to reconnect will be given the same barn number as before.

4.2 Keepalive and Data Transfer

The first part of data transfer is sending data up the tree. Sending data down the tree is exclusively done using multicasting so it will be detailed in the next section.
When sending data up, the **UP** flag is set to 1 and the **dest (Optional)** field is left empty. When receiving such a packet (and if the packet is not related to communications with the mobile device), a node or a sub-gateway will automatically forward it to its only parent.

When the main gateway receives a data packet, it will generate a control packet with the **CTP** flag set to **DATA_ACK** and send it back down the tree. When the nodes or sub-gateways receive such a packet, they forward it down using their routing tables until it reaches its destination. This serves as acknowledgment packet and tells the sender if they are still connected to the network.

As design choice, we've decided that the devices sending data frequently didn't need special keep_alive packets since they already frequently get acknowledgment packets back (e.g. light sensors). The devices that don't send data frequently will send periodically a dummy data packet with the topic set to "/keep_alive" so that they can also know if they are still connected to the network. Each node has an internal counter that gets incremented when sending a data packet and decremented when receiving an acknowledgment packet. If the counter exceeds a set value[1], the node is considered disconnected from the network and will go back to the process explained in subsection 4.1

[1]To modify this threshold, you can modify the **UNACK_TRESH** variable in the project code.

4.3 Multicasting

To send a packet with multicasting enabled, the flag **U** needs to be set to 0, and the **MGP** flag needs to be set to an existing multicast group[2]. When a node receives a multicast packet, it will forward the packet to all the neighbours that have at least a child with the corresponding multicast address. This implementation makes it so that a node receives **at most** one multicast packet each time one is sent, which is optimal. After the forwarding process the node checks if it also belongs to the targeted multicast group. If so, the packet will be processed and, for the irrigation valve, an ack will be sent back[3]

4.4 Mobile communication

When a Mobile is connected to a sub gateway or a node, at first, it will act the same as any other node. It will also send keep_alive data packets to make sure that it is still connected to the network through the same sub gateway. The exception is that when the mobile device receives a **SETUP** packet from a node not yet connected to the network, it discards it.

When a mobile device successfully connects to the network, it will send three data packets upwards every 30 seconds to request information from the light sensors. These data packets have the **U** flag set to 1 (because the request has to travel up the network first) and the **MGP** flag set to the group of devices the mobile wants information from (in our case : **LIGHT_SENSOR_GROUP**). It will also have the **MFL** flag set to **DATA_QUERY**. Regular nodes will forward this packet up as usual but when it reaches the sub gateway, the **DATA_QUERY** flag is detected and the subgateway changes the **U** flag from the packet to 0 (to send it back down). Because there is a non-null **MGP** flag, the packet is sent down using multicast.

Once the packet reaches the desired node(s), again, the **MFL** flag is detected and the nodes makes an answer. The answer is a data packet with the **U** flag set to 1 to send it back to the sub gateway. The MFL flag is set to **DATA_ANSWER** and the **dest** field is used to store the address of the phone (address gotten because it was the source of the previous packet). It then fills the packet with the requested data and sends it. Like before, it will be intercepted by the sub gateway and like before, the sub gateway will change the flags to send the packet to the phone (using unicast and the routing table this time).

5 Server

The server is a basic python file in the repository that can be run with the commands:

```
python3 --ip docker_ip --port port_number
# Command to run with mqtt support
python3 --ip docker_ip --port port_number --mqtt True
```

It will output in the stdout data arriving to the gateway (excluding keep_alive data), it will trigger the light bulb in a barn if the light sensor fall below a certain level and it turns on the irrigation for X time every Y time. All the different global variable are on the top of the file and can be modified as needed.
To run the file you might need to install the paho-mqtt package in python.

```
# To install the pip package
pip install paho-mqtt
```

[2]The multicast group are set on the custom-routing.h file

[3]This is made so that the server can receive an acknowledgment that the irrigation system has been turned on and off

5.1 Communication

The server connects to the gateway with a socket connection in order to read the stdout of the mote and send data to the mote.
The format of the data looks like this:

```
/{barn_number}/{topic}/={data}
```

For example, when a light sensor of the barn 1 send data the output looks like this:

```
/1/light/=128
```

6 Appendix

A MQTT compatibility

We added the possibility to connect the server to an mqtt broker. If a mqtt server is running on localhost you just have to run the server with the mqtt flag set to True, as mentioned above. The server will send every data from the gateway to the mqtt broker in their specified topic. For example the light data will be sent to the */{barn_number}/light* topic.
The server also subscribes to the mqtt broker and will forward data inputted in the broker to our network. Here are a few example:

```
# Publishing on the light topic from barn 0
mosquitto_pub -t '/0/lights' -l
on              # lights on
off             # lights off
on?5            # lights on for 5 minutes

# Publishing on the irrigation topic for barn 0
mosquitto_pub -t '/0/irrigation' -l
10              # irrigation on for 10 seconds

# Publishing on the irrigation topic for all barn
mosquitto_pub -t '/-1/irrigation' -l
10              # irrigation on for 10 seconds on all barn
```