Université Catholique de Louvain École Polytechnique de Louvain

LINFO2146 - Mobile and Embedded Computing

Report

Project

GOMBEER Charles - 31541900

Professor : SADRE Ramin

Assistant : WIAUX Bastien

Academic year 2023 - 2024

Summary -
This document aims to briefly describe the message format in the sensor network, how the routing in the sensor networks works, and how multicasting is implemented.

1 Introduction

In the context of the "LINFO2146 - Mobile and Embedded Computing" course, we are in charge of solving a problem coming from a fictive scenario composed of smart greenhouse fields. The idea is several smart greenhouses scattered across a large plain.
In this plain, each greenhouse contains IoT devices to control the climatic conditions for the plains inside the greenhouse :

- A sub-gateway
- A light sensor and two light bulbs
- One device connected to an external server responsible for managing the greenhouses

And each device is connected to an external server responsible for managing the greenhouses.

The implementation is available at the following link : https://github.com/manonjac/LINFO2146Group-Project/tree/main

2 Message format in the sensor network

Each packet passing through the network is made up of the type, the source, the destination, the length of the payload and the payload of the packet. They can be of different types :

1. **DISCOVERY_PACKET (0):** This packet is used by a node to discover its neighbors and announce its presence in the network. It is broadcast to all nearby nodes.
2. **DISCOVERY_RESPONSE_PACKET (1):** When a node receives a discovery packet, it responds with this packet type. It serves to inform the discovering node about the responder's presence, and the role (useful to select a parent).
3. **CHILD_PACKET (2):** A node uses this packet to announce to another node that it wishes to establish a child-parent relationship.
4. **WHO_IS_THERE_PACKET (3):** This packet is used to query child and underlying nodes, to build the routing table.
5. **WHO_IS_THERE_RESPONSE_PACKET (4):** In response to a WHO_IS_THERE_PACKET, a node sends this packet to his parent.
6. **KEEP_ALIVE_PACKET (6):** This packet is sent periodically from a child node to its parent to maintain the connection between them. It serves to keep the child node active in the parent's routing table.
7. **ACK_KEEP_ALIVE_PACKET (7):** Upon receiving a KEEP_ALIVE_PACKET, a direct parent node responds with this acknowledgment.
8. **DATA_PACKET (8):** Light sensors use this packet type to send their collected data (i.e. light level) to the server.
9. **COMMAND_PACKET (9):** The server sends this type of packet to issue instructions to devices in the network (light bulb and irrigation system). It's used to control various aspects of the system, such as activating lights or irrigation systems based on the collected data.
10. **ACK_PACKET (10):** After receiving a COMMAND_PACKET, the device sends this acknowledgment packet back to the server. It confirms the receipt of the command and may also indicate whether the command was successfully executed.

The format of the payload can be different depending on the type of message and the sensor.

2.1 COMMAND PACKET

The payload sent by the server is `Node type number | Action| Additional data`.

- Node type number : see below.
- Action : is very simple : it's 1 to represent START.
- Additional data : for the irrigation system and light bulb, it's the number of second to reamin on.

For example, the payload sends by the server to the light bulb sensor to remains on for 1h is `4|1|3600`

2.2 DATA PACKET

The TYPE_LIGHT_SENSOR send payload as following : `5| Light level (0 to 128)`.

2.3 ACK PACKET

The motes can send ack and finish message in the format :

- `Node type number | Action| Additional data |1` to advertise the server that the command has been received and started.
- `Node type number | Action| Additional data |2` to advertise the server that the command has been finished.

3 Node role

There are six different node type :

1. TYPE_GATEWAY
2. TYPE_SUBGATEWAY
3. TYPE_MOBILE
4. TYPE_LIGHT_BULB
5. TYPE_LIGHT_SENSOR
6. TYPE_IRRIGATION_SYSTEM

4 Routing in the network

The routing protocol that we developed for the sensor network has been crafted to meet the specific needs of our smart greenhouse scenario. While drawing inspiration from RPL, our protocol introduces features which provide efficient and reliable communication within the network. As its core, our protocol revolves around the concept of a tree structure network, with each node having a single parent. It's far from traditional RPL implementations which allow for multiple parents and complex tree repair mechanisms, our protocol simplifies the architecture to streamline operation and enhance scalability.

4.1 Discovering and structuring the network

When a node starts up, it first sends out a DISCOVERY_PACKET to identify its neighboring nodes. It then waits for 2 seconds, during which it saves in a neighbor list any nodes that respond with a DISCOVERY_RESPONSE_PACKE
The DISCOVERY_RESPONSE_PACKET contains as payload, the role of the node that responded. After this waiting period, the node selects a parent node from the nodes that responded to it, based on its type and RSSI. For example, devices like TYPE_LIGHT_BULB, TYPE_LIGHT_SENSOR and TYPE_IRRIGATION_SYSTEM will prioritize selecting a SUBGATEWAY as their parent, but if none are available, they will choose other devices. SUBGATEWAY, on the other hand, will choose the GATEWAY, and TYPE_MOBILE will choose the SUBGATEWAY. The node will drop the packet if the rssi is below -70 to prevent error on transmission.

Once a parent is selected, the node sends a CHILD_PACKET to inform the parent of their new relationship. Upon receiving this packet, the parent node sends a WHO_IS_THERE_PACKET to the new child. The child then forwards this packet to its own children and responds with a WHO_IS_THERE_RESPONSE_PACKET, containing as payload, the role of the child. The parent updates its routing table with the next hop, the reachable destination through that hop, the clock time, and the role of the destination node. Since each node in the child's underlying network also sends back a WHO_IS_THERE_RESPONSE_PACKET, the parent gains knowledge of all nodes accessible through the new child. The parent will also transfer the packet to his parent. So, parent nodes are aware of subchildren and add them to their routing table.

This enables the gateway and parent to correctly route COMMAND_PACKET to the appropriate nodes within the network.

4.2 Update and maintain connections

To maintain its connection with the parent, a child node periodically sends KEEP_ALIVE_PACKETs, containing his role. Upon receiving this packet, the parent resets the clock associated with that child in its routing table, and update (if any) the role of the child, and responds with an ACK_KEEP_ALIVE_PACKET to confirm that the connection is intact. Then, it will transfer the packet to his parents, so they can do the same and keep the entry in the routing table.

If the child node does not receive the ACK_KEEP_ALIVE_PACKET from the parent within 2 seconds, it will initiate the neighbor discovery process again to select a new parent. Regularly, nodes will scan their routing tables and remove entries for any nodes whose clocks have expired, meaning they haven't sent a KEEP_ALIVE_PACKET in the last 120 seconds, assuming these child nodes are no longer active. This ensures that the network remains optimized at all times.

5 Multicasting implementation

Our multicast implementation is based on broadcasting, in two different ways:

- The server is capable of sending a multicast command to a specific greenhouse by sending a `COMMAND_PACKET` to the subgateway address associated with that greenhouse. Upon receiving the packet, the subgateway will duplicate the packet and forward it to all its child nodes. These child nodes will then decide whether to process the message based on their specific configurations (choose with type).
- The server can send a multicast command to the entire network, which is particularly useful for the irrigation system, by sending a `COMMAND_PACKET` to the address `00:00`. In this case, each node in the network will duplicate the packet and forward it to its child nodes. The end devices within the network will similarly determine whether to process the packet according to their configuration.

6 Server

The server automatically controls the irrigation system and lighting based on light levels. Upon startup, it requests the routing table from the gateway, allowing it to categorize devices by type and by greenhouse (assuming each sub-gateway corresponds to a greenhouse).

Upon startup, a menu is presented that allows us to choose from several actions:

- 1. List greenhouses and devices addresses
- 2. Start irrigation system : to start the irrigation system for a selected number of seconds.
- 3. Start lights : to start the all the lights, the lights of a specific greenhouse or a specific light for a selected number of seconds.
- 4. Start automatic system (irrigation system and light management)

The irrigation system will start every day at 5:00 AM, while the greenhouse lights will turn on for 60 sec if the light level detected in it is below 20.

7 Conclusion

In conclusion, our work involved designing and implementing a simulation of a smart greenhouse ecosystem. We developed a custom communication protocol, enabling various network nodes, servers, gateways, sub-gateways, and sensors to communicate effectively via packets of different types such as discovery, commands, data, keep-alive, acknowledgments.

Moreover, we created our own protocol with specific message format to enable all motes to communicate, with unicast, multicast and broadcast. Using these transmission modes, we facilitate data exchanges and command dissemination within the network. Our approach was guided by the aim to optimize efficiency while ensuring network stability and responsiveness, which involved managing transmission intervals and handling node disconnections. Another important goal achieved is the fact that all motes have been implemented except mobile and the command IRRIGATION_START makes Cooja crash for unknown reason at the end of the project.

Lastly, our exploration of multicast implementation underscored the importance of scalable and adaptable communication strategies to meet the specific requirements even in presence of difficulties during the working process. This project enhanced our understanding of mobile and embedded computing concepts, while providing valuable skills in designing and optimizing IoT networks.

References

[1] **geeksforgeeks.org** - https://www.geeksforgeeks.org/rpl-ipv6-routing-protocol/

[2] **jmainy.free.fr** - http://jmainy.free.fr/guill.web-/Modes.html

[3] **mdpi.com** - https://www.mdpi.com/2076-3417/13/8/4878

[4] **wikipédia** - https://en.wikipedia.org/wiki/Multicast

[5] **wikipédia**-https://en.wikipedia.org/wiki/IPv6_Routing_Protocol_for_Low-Power_and_Lossy_Networks