Report Smart Bracelet Project

Alessandro Barbiero 10692413
Roberto Alessandri 10783942

Contiki

Files

- child.c and child2.c → These two scripts are used to start at booting time the Child Bracelet process. The only difference between these two files is the "PRELOADED_KEY";
- parent.c and parent2.c → These two scripts are used to start at booting time the Parent Bracelet process. The only difference between these two files is the "PRELOADED_KEY";
- key-generation.c → This script is used to start at booting time the Key Generation process [Not used in production, see notes];
- smart-bracelet.h → This header contains all the logic of our system and the implementation of various processes.

Methods

- Callback Missing → This method resets the timer and fires the "MISSING" ALARM. It is called by the cTimer when 60 seconds are passed from the last message received;
- Key Generation → Iterate a cycle 20 times for the creation of random characters. For the creation of a character we randomly chose values from 32 to 127, since this is the range of the ASCII printable characters;
- Broadcast Reception → This method works only if the mote hasn't received a Stop Pairing message or the Pairing Phase isn't finished. The method checks if the received message matches the key owned by the device. If this is the case, it ends the Pairing Phase and stores the address of the mote that sent the correct key;
- UniCast Reception → This method distinguishes between two cases:
    - If the mote is in the Pairing Phase and received the "Stop Pairing" message, it ends the Pairing Phase and saves the address of the mote that sent the message;
    - If the mote is in the Operation Mode and it's a Parent, it checks if the address that sent the message is the same as the one previously stored. In this case, it saves the position of the child and checks the status. In case the status is "FALLING" it prints a "FALL" ALARM.

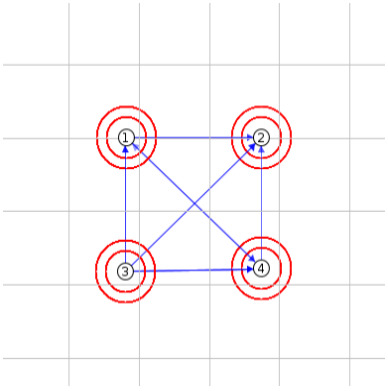
Every time a message is received the cTimer handling the callback for the MISSING alarm is restarted.

- Stop Pairing Message → This method sends the “Stop Pairing” message only if a Unicast message has not been received yet
- Pairing Message → This method sends a broadcast message containing the key of the mote
- Info Message → This method first defines the INFO variables (position x and y, and the status) as random values and then sends them to the saved address. The message is a custom struct made of a struct with two integer values and an enumeration for the status. To obtain the probabilities requested for the status, the remainder of the division by 100 of a random number is calculated and the status value is calulated depending on the range of the remainder. In this way, a range of 30 has a 0.3 probability and a range of 10 has a 0.1 probability.

Processes

- Parent Bracelet → The parent sends a pairing message every 2 seconds until it is in the Pairing Phase. When the pairing ends, closes the broadcast connection, sends the “Stop Pairing” message and starts the “Operation Mode”. In this mode, if it doesn’t receive any unicast message from the address of the child for 60 seconds, prints a “MISSING” ALARM.
- Child Bracelet → The child behaves as the parent until it enters the “Operation Mode”. In this mode, it sends a unicast message to the parent containing its INFO every 10 seconds.

Cooja Results



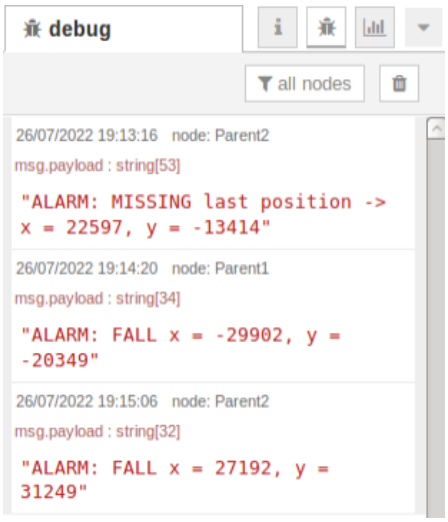
We used 4 sky motes (two parents and two children) running the code of child.c, parent.c, child2.c, parent2.c to check the correct behaviour of the system. The same key is shared between a couple of parent-child. If a mote isn’t in the range of another with the same key it continues the Pairing Phase indefinitely without entering the operation mode.
Once two motes are paired, they stop sending broadcast messages.
During Operation Mode, if two paired motes are put far away from each other for more than a minute a MISSING ALARM from the parent is printed.
If the child sends a message with the FALLING status, the parent fires a FALL ALARM.

The prints of the two parent nodes are forwarded using two Serial Socket Servers listening on ports 60001 and 60003.

Node-RED

The flow on Node-RED consists of two parallel flows.
For each one three nodes are used. Three for one parent and three for the other. For each parent:

- Receive every print with a TCP-input node linked with the related cooja serial server;
- Filter out all the strings that don't contain the "ALARM" sub-string with a function node;
- Print out only the filtered messages with a debug node.



Notes:

- In the beginning, we implemented a method for the automatic generation of a key at construction time. Then we conveyed that it was out of the scope of the project. That's the reason you will find the implementation of the method, but the method seems to be never used. It was used for the generation of the keys.
- We didn't create an actual "Alert Mode", but we implemented the Alert Mode's behaviour inside the Operation Mode. In this way, the bracelets can continue working in the same way also after the communication of the alert.
- "child.c" and "parent.c" share the same key. The same goes for "child2.c" and "parent2.c". The key is implemented through a #define.

For more information and to see the complete code visit our Github repository.