MARCO ADRIANO FERRERO

PROJECT REPORT – 10613076
SMART BRACELETS

APPLICATION LOGIC (design choices):

All the logic to realize the PAIRING, OPERATION, and ALERT modes are implemented in the braceletC.nc file. Here I have decided to define three interfaces for timers to manage these phases:

PairingTimer, OperationTimer, and *AlertTimer*

When the application is booted the PairingTimer is used to start a *one-second* periodic timer. This timer will trigger the function for broadcasting its (pre-loaded) pairing key.

When a pairing key is received, it is analyzed: if corresponds to the assigned one (pre-loaded), a PAIRING_CONFIRM message is sent. If this message is ACKed back, the PAIRING phase is concluded, and the paired ID of the connected device is saved locally.

In the case of a CHILD bracelet, now the OPERATION phase starts, turning on the *OperationTimer* and firing the *PairingTimer*. On the other side, in the case of a PARENT bracelet, the *AlarmTimer* is started and if there is a loss of connection after 60 seconds, a MISSING alarm is sent.

APPLICATION FILES:

Run Simulation File (*RunSimulationScript.py*):

This python file allows starting simulation. Here I have defined the four nodes and the radio channels that are required to manage the different phases of simulation as specified in the requirements.

In line 138 a cycle is used to trigger events and simulate all the behaviors for two couples of bracelets:

From cycle 1001 to 2500 node 2 is turned OFF to emulate that the child node goes out of the range, thus forcing a MISSING alert to be sent.

Struct file (braceletMsg.h):

This file has defined the values of constants (PAIRING KEYS, MESSAGE TYPES, KINEMATIC STATUS, and BRACELET ROLES) and the structure of the two types of messages.

<u>Messages format:</u>

- *msg*: sent between nodes. It contains all information needed for the pairing phase (such as key and userID) or operation/alert mode (such as coordinates and kinematic status).
- *sensor_msg*: sent from sensor to mote when a sensor read is performed. It is a simplified version of "*msg*" that only includes kinematic status and coordinates.

Component File (braceletC.nc):

This file gives the complete definition for the logic and implementation of bracelet motes (parent or child).

All the relevant functions in sending or receiving modes have some debug logs that help to understand the behaviors, phases, and operations that are taking place in the simulation

Sensor File (FakeSensorP.nc):

This file provides a function that implements the Read interface, to perform a (fake) measurement.

Using the Random library, two numbers in the range [0,100] are generated, to emulate the x and y coordinates for a (fake) position.

Another number is also generated at each read to realize the state constraint for kinematic status:
P(STANDING) = P(WALKING) = P(RUNNING) = 0.3 ; P(FALLING) = 0.1

Simulation Log (simulation.txt):

This file shows debug messages for a complete simulation. All the behaviors of motes are covered as specified in the requirements.

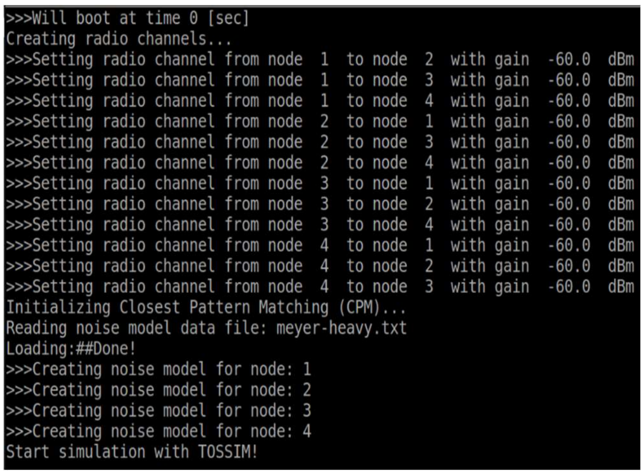
SENSORS ROLES:

In my design choices I have assigned the CHILD roles to nodes 2 and 4, and the PARENT role to nodes 1 and 3. Couples (1,2) and (3,4) have the same pairing keys so that when the keys are sent in broadcast to all the motes, this is the only possible assignment.

SIMULATION ANALYSIS:

All the images in this paragraph are taken from the simulation.txt file, in which it is possible to observe all the logs for a complete simulation.

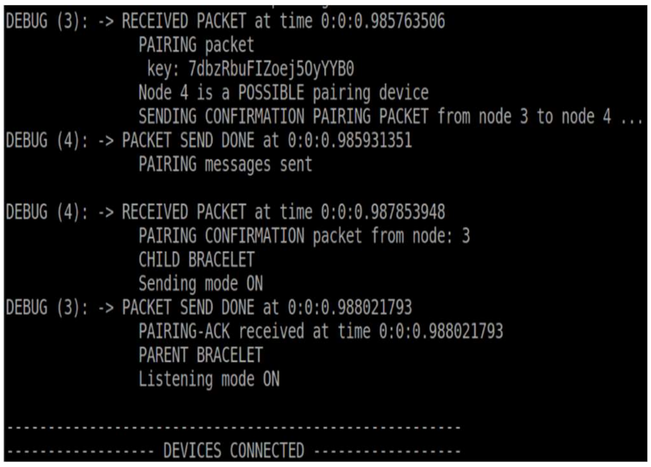
SENSORS BOOT



```
>>>Will boot at time 0 [sec]
Creating radio channels...
>>>Setting radio channel from node  1  to node  2  with gain  -60.0  dBm
>>>Setting radio channel from node  1  to node  3  with gain  -60.0  dBm
>>>Setting radio channel from node  1  to node  4  with gain  -60.0  dBm
>>>Setting radio channel from node  2  to node  1  with gain  -60.0  dBm
>>>Setting radio channel from node  2  to node  3  with gain  -60.0  dBm
>>>Setting radio channel from node  2  to node  4  with gain  -60.0  dBm
>>>Setting radio channel from node  3  to node  1  with gain  -60.0  dBm
>>>Setting radio channel from node  3  to node  2  with gain  -60.0  dBm
>>>Setting radio channel from node  3  to node  4  with gain  -60.0  dBm
>>>Setting radio channel from node  4  to node  1  with gain  -60.0  dBm
>>>Setting radio channel from node  4  to node  2  with gain  -60.0  dBm
>>>Setting radio channel from node  4  to node  3  with gain  -60.0  dBm
Initializing Closest Pattern Matching (CPM)...
Reading noise model data file: meyer-heavy.txt
Loading:##Done!
>>>Creating noise model for node: 1
>>>Creating noise model for node: 2
>>>Creating noise model for node: 3
>>>Creating noise model for node: 4
Start simulation with TOSSIM!
```

First, when the RunSimulationScript.py is run, radio channels and nodes are set and TOSSIM simulation is started.

BROADCAST PAIRING

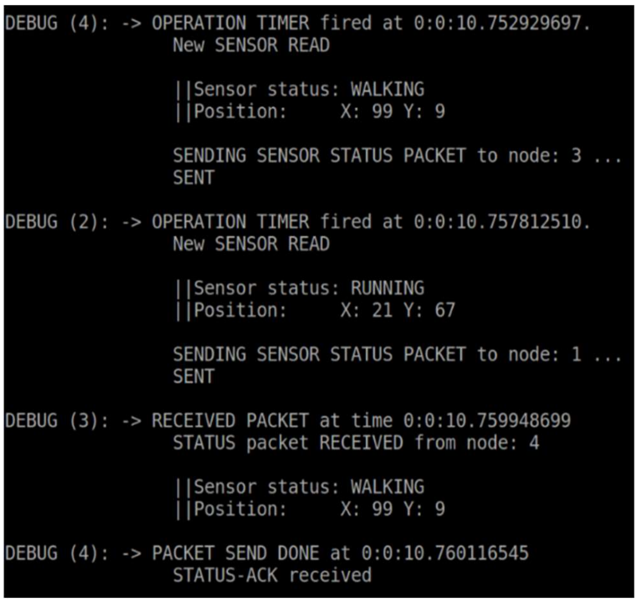


```
DEBUG (3): -> RECEIVED PACKET at time 0:0:0.985763506
               PAIRING packet
                key: 7dbzRbuFIZoej5OyYYB0
               Node 4 is a POSSIBLE pairing device
               SENDING CONFIRMATION PAIRING PACKET from node 3 to node 4 ...
DEBUG (4): -> PACKET SEND DONE at 0:0:0.985931351
               PAIRING messages sent

DEBUG (4): -> RECEIVED PACKET at time 0:0:0.987853948
               PAIRING CONFIRMATION packet from node: 3
               CHILD BRACELET
               Sending mode ON
DEBUG (3): -> PACKET SEND DONE at 0:0:0.988021793
               PAIRING-ACK received at time 0:0:0.988021793
               PARENT BRACELET
               Listening mode ON

-------------------------------------------------------
----------------- DEVICES CONNECTED -----------------
```

When a mote is turned on, it starts to send to all the other addresses a message of type PAIRING, with its specific key in the payload.

In the following screenshot from the *simulation.txt* file is possible to see that node 3 has received a packet (from node 4), that has the same pairing key as the pre-loaded one. Now, node 4 sends back a PAIRING-CONFIRMATION packet to complete the pairing phase with node 4. When pairing confirmation is received from node 4, the CHILD role is assigned, and an ACK is sent back. When node 3 received the ACK message, the PARENT role is assigned.

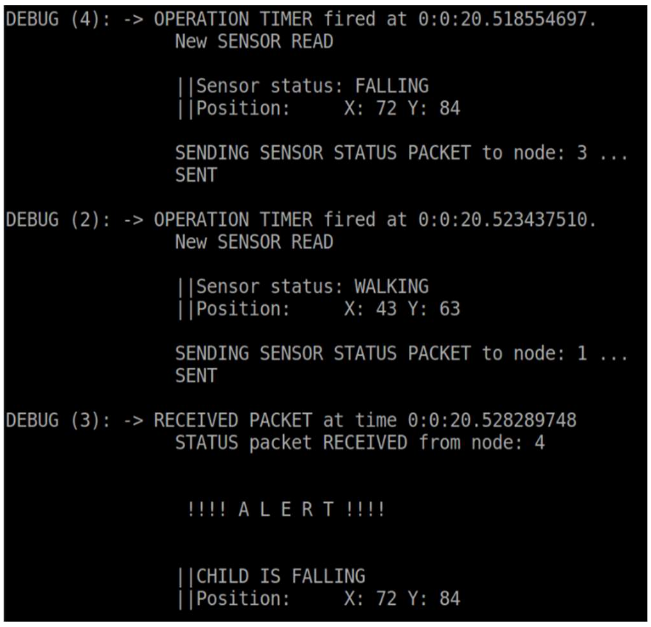
SENSOR READ



```
DEBUG (4): -> OPERATION TIMER fired at 0:0:10.752929697.
            New SENSOR READ

            ||Sensor status: WALKING
            ||Position:     X: 99 Y: 9

            SENDING SENSOR STATUS PACKET to node: 3 ...
            SENT

DEBUG (2): -> OPERATION TIMER fired at 0:0:10.757812510.
            New SENSOR READ

            ||Sensor status: RUNNING
            ||Position:     X: 21 Y: 67

            SENDING SENSOR STATUS PACKET to node: 1 ...
            SENT

DEBUG (3): -> RECEIVED PACKET at time 0:0:10.759948699
            STATUS packet RECEIVED from node: 4

            ||Sensor status: WALKING
            ||Position:     X: 99 Y: 9

DEBUG (4): -> PACKET SEND DONE at 0:0:10.760116545
            STATUS-ACK received
```

From this moment on, the OperationTimer is turned on and it will fire every 10 seconds. In the picture it is possible to observe a sensor read from node 4 (CHILD) sent to node 3 (PARENT). Read operation is completed when the ACK to the status message is received in the CHILD node.

FALLING ALERT

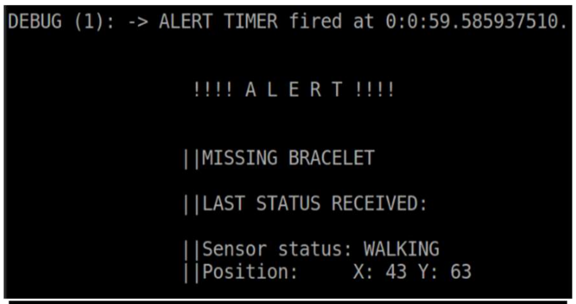


```
DEBUG (4): -> OPERATION TIMER fired at 0:0:20.518554697.
            New SENSOR READ

            ||Sensor status: FALLING
            ||Position:     X: 72 Y: 84

            SENDING SENSOR STATUS PACKET to node: 3 ...
            SENT

DEBUG (2): -> OPERATION TIMER fired at 0:0:20.523437510.
            New SENSOR READ

            ||Sensor status: WALKING
            ||Position:     X: 43 Y: 63

            SENDING SENSOR STATUS PACKET to node: 1 ...
            SENT

DEBUG (3): -> RECEIVED PACKET at time 0:0:20.528289748
            STATUS packet RECEIVED from node: 4


             !!!! A L E R T !!!!


            ||CHILD IS FALLING
            ||Position:     X: 72 Y: 84
```

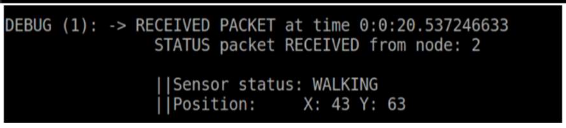
Here it is observable one of the two possible types of alert status: FALLING.

When a sensor read is performed from node 4 (CHILD) and the new status generated from the "fake" sensor is of type FALLING, a status message is sent to the PARENT mote as a normal status. When it is received in the parent node (node 3), a special message is shown in the terminal (as specified in debug function)

MISSING ALERT



```
DEBUG (1): -> ALERT TIMER fired at 0:0:59.585937510.


               !!!! A L E R T !!!!


              ||MISSING BRACELET

              ||LAST STATUS RECEIVED:

              ||Sensor status: WALKING
              ||Position:     X: 43 Y: 63
```



```
DEBUG (1): -> RECEIVED PACKET at time 0:0:20.537246633
            STATUS packet RECEIVED from node: 2

            ||Sensor status: WALKING
            ||Position:     X: 43 Y: 63
```

The last possible status is the MISSING alert. When no message is received in the last 60 seconds, a new ALERT is shown in the terminal with the last status message received. It is possible to simulate this behavior by calling the turnOff() function in the runSimulationScript.py for node 2, to emulate a loss of connection for a CHILD bracelet.