Anna Carè - Giuseppe Calcagno

Project 1. Smart Bracelets

DESCRIPTION OF THE COMPONENTS AND INITIAL PHASE

We started creating the three files named *Project.h*, *ProjectC.nc* and *ProjectAppC.nc* and we added to the project folder the Tossim dependency.

Inside *Project.h* we wrote the struct of the three different types of messages: *msg_pairing*, *msg_stop_pairing* and *msg_child*. The first is needed for the match phase between the parents and the children and includes the *msg_type*, the pairing key and the sender id. The second is used at the end of the pairing phase to stop the broadcast of the messages and contains only the *msg_type* (since it is an unicast message we don't need other identifiers). The third contains the status and position of the child with also the type of the message. We included in the file some constants useful throughout the code.

Inside *ProjectC.nc* we implemented the following interfaces:

```
/****** INTERFACES *****/
  interface Boot;
  //interfaces for communication
  interface AMSend;
  interface Packet;
  interface SplitControl;
  interface Receive;
  interface PacketAcknowledgements;
  //interface for timers
  interface Timer<TMilli> as MilliTimer;
  //interface for missing timer
  interface Timer<TMilli> as MissingTimer;
  //interface for info timer
  interface Timer<TMilli> as InfoTimer;
  //interace to get a random Number
  interface Random;
```

The *AMSend*, *Packet*, SplitControl, *Receive* and *PacketAcknowledgements* interfaces are used for the communication.
We decided to use 3 timer: *MilliTimer* is used for the pairing phase and it starts when the radio is turned on. When the *msg_stop_pairing* arrives this timer is stopped and the pairing phase is concluded. Then *InfoTimer* is used by the child mote with a period of 10 sec to send messages about his/her status and position. The *MissingTimer* is implemented inside the parent mote and each time the parent receives an info message the timer is resetted. This is used to trigger the missing alarm in that case in which the parent doesn't receive any message in one minute.
The last interface (*Random* interface) is exploited for generating the random status and position of the children.
In file *ProjectAppC.nc* we linked the corresponding components.

IMPLEMENTATION

Inside the *ProjectC.nc* we developed the behavior of the motes.
In the event *Boot.booted()* we setted the key and id of the four motes. The key is implemented as 2 pairs of constant strings of 20 char. After this, we started the radio calling the function *SplitControl.start()*.

In the event *SplitControl.startDone()* we started the *MilliTimer* of all the motes initializing the pairing phase. When the timer is fired we send a broadcast pairing request.
In the event *Receive.receive* we check the type of the arrived message. If the read message is a *msg_pairing* we convert the received key from uint8 to char. If the embedded keys and received key coincides we save the sender address and we send a message for stopping the matching phase.

When the *msg_stop_pairing* is received we stop the *MilliTimer* and we move to operation mode. In this phase if the mote belongs to the child we start the *InfoTimer* that allows the child to send status messages in unicast to the corresponding parent bracelet.

The child status is defined in the *createStatus()* function in which we exploited the random interface. We associate the picked number to a status according to the required probability distribution.

```
void createStatus(){
  uint8_t rnd = (call Random.rand16() % 10) + 1;

    if(rnd<=3){
       ChildStatus=STANDING;
    }
    else if(rnd<=6){
       ChildStatus=WALKING;
    }
    else if(rnd<=9){
       ChildStatus=RUNNING;
    }
    else if(rnd=10){
       ChildStatus=FALLING;
    }
    X = call Random.rand16() % 100;
    Y = call Random.rand16() % 100;
}
```

In *Receive.receive* event if the parent reads a *msg_child* (containing all the child info) he/she stores the child's last position and checks his/her status.
In the case the status is FALLING the parent generates a FALL alarm reporting the last position.
When the parent receives the message his/her timer is resetted (for the first message it is activated).

In *AMSend.sendDone()* event we implemented the ACK check for the unicast messages (*msg_child* and *msg_stop_pairing*).

When the *MissingTimer* expires (the parent doesn't receive a message in one minute) a MISSING alarm is created reporting the last position of the child.

SIMULATION WITH TOSSIM

In order to test the performance of our components we used Tossim.
We took *meyer-heavy.txt* (noise model) from given simulations and we setted the topology of the network using the file *topology.txt* implementing a fully connected topology.
We created the 4 motes and the redirect output to a file (SIMULATION.txt).

IMPLEMENTATION CHOICES

- We decided to split the periodical messages into three timers for the clean of the code and because they are used for very different purposes.

- We decided to implement a QoS “at least 1” using the PacketAcknowledgments interface for the unicast messages and not broadcast one in order to limit the forwarding of the messages. NOTE: in the folder we also attached a simulation where a packet and an ACK is lost to track the behavior of the system in that situation.

- We decided to activate the parent *MissingTimer* only after the arrival of the first message in order to avoid that the parent generates an alarm before the child starts sending info messages.

- We implement the 4 mote following this rule: the first 2 motes are linked between each other, the other two as well. Odd motes belong to the children, even ones to parents.

- Since the behavior of *MissingTimer* is not specified we decided to not stop it after the first MISSING alarm in order to continue notifying the parent about the removal of the child. However, we added in *MissingTimer.fired()* a commented call MissingTimer.stop() to change the behavior of the parent bracelet. In the case the child comes back the timer would be restarted.