Project Report: Smart Bracelets

Alberto Latino - **10600138**, *Lorenzo Mainetti* - **10622242**

Project Scope:
You are requested to design, implement and test a software prototype for a smart bracelet.
The bracelet is worn by a child and her/his parent to keep track of the child's position and trigger alerts when a child goes too far.

The operation of the smart bracelet couple is as follows:

1. **Pairing**:
    - broadcast a 20-char random key
    - upon reception of key checks whether the received random key is equal to the stored one
    - special message is transmitted to stop the pairing
2. **Operation**:
    - parent's bracelet listens for messages and accepts only the ones coming from the child's
    - child's bracelet periodically transmits INFO messages
3. **Alert**:
    - Upon reception of an INFO message, the parent's bracelet reads it. If the kinematic status is FALLING, the bracelet displays a FALL alarm
    - if the parent's bracelet does not receive any message, after one minute from the last received message, a MISSING alarm is sent reporting the last position received

Implementation Choices:
- TinyOS
- TOSSIM
- node-RED

1. TinyOS App implementation

a. bracelet.h

We defined two types of messages, one for the pairing and one for the position. They are represented by the following struct:

- Pairing Messages:
    - msg_type (0 -> KEY_MSG, 1 -> PAIR_ACK)
    - key

- Position Messages:
    - msg_type (2 -> INFO, 3 -> FALL, 4 -> MISSING)
    - status (STANDING, WALKING, RUNNING, FALLING)
    - position: x coordinate
    - position: y coordinate

- Random Key :
  a random key is chosen among a predefined set. Then it is assigned to a couple of nodes in order to make them pair with each other.

b. **braceletC.nc**

In this module we define all the events that manage the behavior of the nodes when the timer starts or a message is received/sent. Each node has 2 modalities, Pairing or Operation, and can be defined as parent or child nodes, based on even or odd node's id. Based on these conditions, the flow is controlled.

Every node starts in pairing modality and if the pairing is successful it switches to Operation mode where the core behavior is implemented. In particular, the flow is made of the following steps:

i. **broadcastKey( ):** this phase is triggered every 10 seconds by a timer. This choice was made in order to guarantee a pairing in case of failure in previous attempts. A pairing message is created with the node's key and is broadcasted (KEY_MSG).

ii. **sendPairAck( ):** this method sends an acknowledgment (PAIR_ACK) to the node with the same key to communicate that the matching was successful on its side.

iii. **receive( ):** this step controls the behavior of a node upon the reception of a message. The first condition that controls the node's behavior is whether the modality is *Pairing* or *Operation.*

   1. If the node is in **Pairing** mode, it looks for pairing messages of the type KEY_MSG or PAIR_ACK. If a KEY_MSG is received the node reads the key and checks if it is equal to its own. If the key matches, a PAIR_ACK message is sent to the other node. In case the received message is a PAIR_ACK, it means another node confirmed the pairing from the other side; in this case the mode switches to Operation and the pairing timer is stopped. If this happens on a child node, a timer that triggers the sending of position messages is started. Once the acknowledgment (`PacketAcknowledgements interface`) of the PAIR_ACK message is received also on the other side, the sender also switches to Operation mode.

   2. If the node is in **Operation** mode, it is known that only a parent node can receive messages from the child. Therefore, the behavior will be the one of the parent nodes. It reads the content of messages. In case the content is a Falling message, an alarm is triggered. A timer is triggered in order to check that messages are arriving at most no more than every 60 seconds. In case this timer is triggered, the display alarm method is called.

iv. **displayAlarm( ):** this method is called in case no messages were received from the child within 60 seconds or the last message received indicated the child had fallen.

v. **readDone( ):** This event is triggered only on the child when a new status is read from the FakeSensor, which generates a status with the assigned probability (Falling -> 0.1, Walking/Standing/Running -> 0.3). The position (x, y) is instead randomly selected.
This gathered information is then sent to its parent node.

c. **braceletAppC.nc**

Our application implements the following TinyOS components:

```
components MainC, braceletC as App;
components new TimerMilliC() as PairingTimer;
components new TimerMilliC() as OperationTimer;
components new TimerMilliC() as AlertTimer;
components new AMSenderC(AM_MSG);
components new AMReceiverC(AM_MSG);
components new FakeSensorC();
components ActiveMessageC;
```

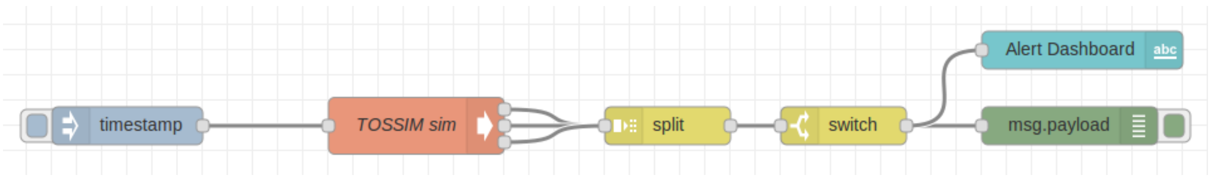
In this file we declared them and then we wired them accordingly.

2. **TOSSIM simulation**
We compiled our TinyOS implementation adding the '*sim*' keyword and we simulated the interaction between two pairs of bracelets (parent - child) using the RunSimulationScript.py.
In order to test the detection of the MISSING alarm, the turnOff() method is called on one of the two children bracelets.

3. **node-RED simulation**
We attached our TOSSIM simulation to node-RED, and we used the following nodes in order to display the most recent alarm on the node-RED dashboard:

- **Inject Node**: it is used to manually trigger the flow, no repetition is set
- **Exec Node**: it executes the RunSimulationScript.py script with the python command
- **Split Node**: the output of the simulation is than split using the DEBUG keyword, obtaining all the different debug messages
- **Switch Node**: it filters out only those messages that contains the ALARM keyword
- **UI Text Node**: it displays on a text dashboard the last alarm received

We also added a **Debug Node** to print the msg.payload and to check that everything works as expected.


timestamp
TOSSIM sim
split
switch
Alert Dashboard
abc
msg.payload