Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

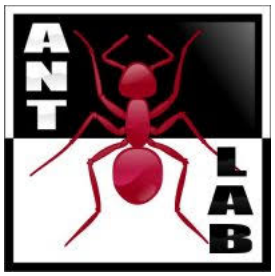
Homework IoT Project

SendAck

Author:
Francesco Monti
Matr: 919755

Supervisor:
Dr. Edoardo Longo
Dr. Matteo Cesana

March 26, 2020



Abstract

This document contains the documentation for the second activity for the course "Internet of Things", Academic Year 2019/2020.
We firstly list the requirements, then we present our implementation.
All the code can be found in the following GitHub repository: https://github.com/Framonti/IoT_Projects

0.1 Requirements

The goal of the project is to implement a simple application for the device Micaz, a small component supporting TinyOS. The application should be deployed on two different motes (Requester and Sender) and implement a reliable communication protocol between them through ACKs messages; unless the Requester is able to obtain an ACK from the Sender, it continues sending messages (and viceversa).

0.2 Implementation

We started from the template provided for the three main files, *sendAck.h*, *sendAckC.nc* and *sendAckAppC.nc*.
In the header file, we just defined the message structure and some constant.
In the App file, we declared the components we used and then wire them.
The most interesting file is *sendAckC.nc*, which contains the application logic.
This is what the implementation does:

- Both the motes are booted and then start their radio, but only the Requester starts a timer
- When the timer on the Requester fires, it calls the function sendReq, which creates a message, sets an AKC calling PAck.requestAck(&packet), sends the message and increments an internal counter.
- The Sender, if booted, receives the message, and manages it while sending back an ACK
- The Requester manages the event AMSend.sendDone; if it receives an ACK, it stops the timer.

- The Sender manages the incoming message, getting the data from a fake sensor and sending a message to the Requester.
- The Requester receives the data, and sends an ACK.
- The Sender checks the ACK from the Requester; if it doesn't receive it, it create a new message, otherwise everything is okay and the simulation stops.

0.3 Simulation

The file *simulation.txt* contains the results of the simulation. To note, the Timer in the Requester is a bit faster than what we would expect (more or less 997 ms instead of 1000); we don't have an explanation for this phenomenon, but nonetheless it doesn't impact the overall behaviour of the implementation.
We see that the Requester tries to send a message, but the first attempts are fruitless, as the second mote hasn't been booted yet. Instead, the sixth message is taken, the ACKs messages are exchanged, as well as the reply with the data.
We simulates our implementation multiple times,and the simulations obtained were comparable.
The simulation therefore validates our implementation.