IoT - Home Challenge # 2

Politecnico di Milano

D'introno [939856], Moreno [939974], Zaniolo [927512]

March 29, 2020

Home Challenge 2

Code review

Our code[1] is based on the template SendAck[2]. The structure of the message sendAck.h is composed by two following structures: *mote_req_t* for Request messages and *mote_res_t* for Response messages. It's defined the constant *REQ_PERIOD* for interval time of a periodic request.

Listing 1: sendAck.h. Message structure and timer constant.

```
#define REQ_PERIOD 1000     //  1s
#define AM_MY_MSG 6

#define REQ
typedef nx_struct mote_req{
    nx_uint16_t counter;
} mote_req_t;

#define RESP
typedef nx_struct mote_res{
    nx_uint16_t counter;
    nx_uint16_t meas;
} mote_res_t;
```

In the configuration file, on Listing 2, we added and wired the *PacketAcknowledgements* component to require and check acks by motes. Other components are the same as in the SensorC tutorial.

Listing 2: sendAckAppC.nc. Configuration file to enable packet ackowledgement.

```
configuration sendAckAppC {}
implementation {
  ...
  App.PacketAcknowledgements -> AMSenderC;
  ...
```

[1] The Github repository is available here

[2] The sendAck.zip contains sendAck.h,sendAckAppC.nc, sendAckC.nc,FakeSensorC.nc,FakeSensorP.nc,topology.txt, meyer-heavy.txt, RunSimulationScript.py and simulation.txt

```
}
```

We have added the variable ***bool*** *locked* to disable the channel while the mote is sending a message. In this case, the communication is slow enough to neglect this term, but it's a good practice if there is some delay in the sending channel.

The function *sendReq()* on listing 3 creates the request with the ackowledgement flag turned on and send it to mote2. Note that the counter starts from zero but it's unitary increment it's before the assignment.

Listing 3: sendAckC.nc. Send request with ackowledgement.

```
void sendReq() {
    mote_req_t* req = (mote_req_t*)call Packet.getPayload(&packet, ↩
       sizeof(mote_req_t));

    if(req == NULL){return;}
    req->counter = ++counter;

    call PacketAcknowledgements.requestAck(&packet);
    if (call AMSend.send(2, &packet, sizeof(mote_req_t)) == SUCCESS) {
        dbg("radio_send","  C-%d :: REQUEST SENT\n", counter);
        locked = TRUE;
    }
}
```

The function *sendDone()* on listing 4 checks if the package is sent and an ackowledgement has been sent from the recipient. If that's the case, then the timer stops if it's running.

Listing 4: sendAckC.nc. Check ackowledgement.

```
event void AMSend.sendDone(message_t* buf,error_t err) {
    if(call PacketAcknowledgements.wasAcked(buf)){
        dbg("radio_ack","  [OK] Packet acknoledgment OK\n");
        if(call MilliTimer.isRunning()){
            call MilliTimer.stop();
            dbg("radio_ack","  [OK] Timer stopped\n");
        }
    }else{
        dbgerror("radio_ack","  [x] Packet acknoledgment FAILED\n");
    }
    if (&packet == buf) {
      locked = FALSE;
    }
}
```

The function *receive()* on listing 5 checks the packet type and executes the action depending on this. If it's a request, mote2 assigns the received counter and starts the sensor read task. If it's a response, mote1 just show the value received in the field measurement.

Listing 5: sendAckC.nc. Handle received packets.

```
event message_t* Receive.receive(message_t* buf,void* payload, uint8_t len) {
    dbg("radio_ack","***PACKET RECEIVED ");
    if (len == sizeof(mote_req_t)){
        mote_req_t* req = (mote_req_t*)payload;
        dbg_clear("radio_ack","-> C-%d\n", req->counter);
        counter = req->counter;
        sendRes();
    }
    if (len == sizeof(mote_res_t)){
        mote_res_t* res = (mote_res_t*)payload;
        dbg_clear(
            "role"," -> C-%d => MEASUREMENT = %d\n",
            res->counter,
            res->meas
        );
    }

    return buf;
}
```

The function *readDone()* on listing 6 read the sensor value and execute the send response task.

Listing 6: sendAckC.nc. Read sensor.

```
event void Read.readDone(error_t result, uint16_t data) {
    dbg("role","Measurement READ OK %d\n",data);
    sendResponse(data);
}
```

The function *sendResponse()* on listing 7 creates the packet with the ackowledgement flag turned on and send it to mote1.

Listing 7: sendAckC.nc. Send response.

```
void sendResponse(double meas){
    mote_res_t* res = (mote_res_t*)call Packet.getPayload(&packet, sizeof(mote_res_t));

    if(res == NULL){return;}
    res->counter = counter;
    res->meas = meas;

    call PacketAcknowledgements.requestAck(&packet);
    if (call AMSend.send(1, &packet, sizeof(mote_res_t)) == SUCCESS) {
        dbg("radio_send","C-%d :: RESPONSE SENT\n", counter);
        locked = TRUE;
    }
}
```