

PROJECT 1 – LIGHTWEIGHT PUBLISH-SUBSCRIBE APPLICATION PROTOCOL

INDEX

Assumptions	2
Topology.txt	3
Project1.h	4
Project1AppC.nc	6
Project1C.nc	7
Definition of components for connecting to Node-RED	7
Event: Boot.booted()	7
Event: AMControl.startDone(error_t err)	7
Event: AMControl.stopDone(error_t err)	7
Event: MilliTimer.fired()	7
Event: Receive.receive(message_t* bufPtr, void* payload, uint8_t len)	7
Event: AMSend.sendDone(message_t* bufPtr, error_t error)	8
Node-RED flow	9
Receive from TinyOS	9
Parse message	9
ThingSpeak	9

[ThingSpeak public channel](#)

ASSUMPTIONS

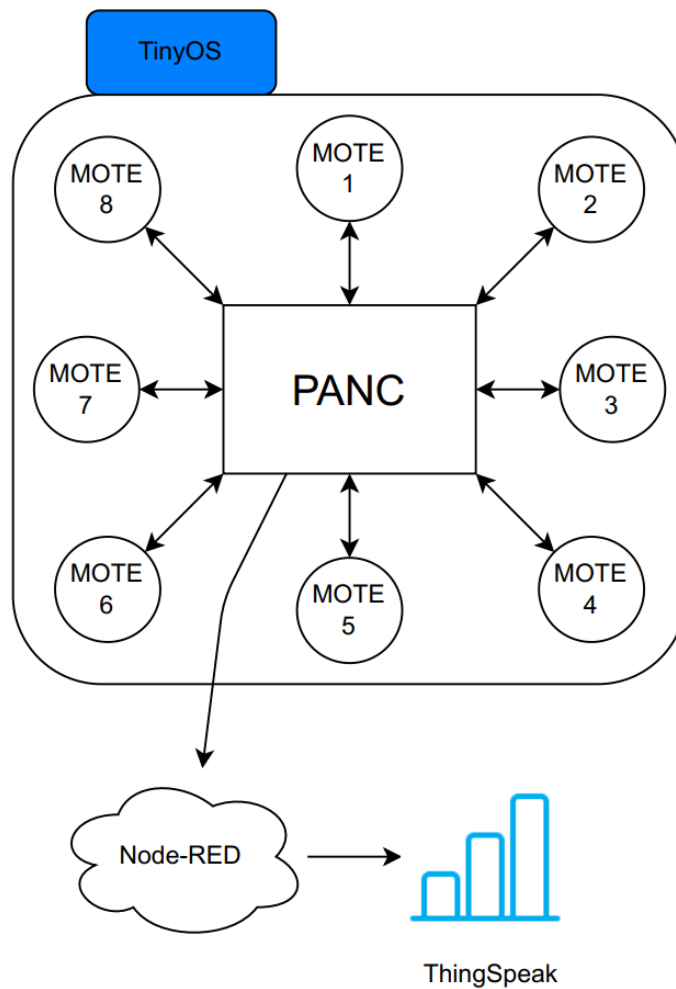
1. The node indicated as 1, in the file topology.txt, is the PAN Coordinator.
2. To handle the retransmission of CONN and CONNACK messages, we decided to resend a CONN message in both cases, whether CONN is lost or CONNACK is lost.
3. To handle the retransmission of SUB and SUBACK messages, we decided to resend a SUB message in both cases, whether SUB is lost or SUBACK is lost.
4. To handle the retransmission we used some arrays that help us to keep track of which nodes received any messages.
5. We left some lines commented to help you in understanding what is going on between nodes and PAN. (e.g. line 129 – 130 – 131 – Project1.nc | you can see step-by-step which nodes have received CONNACK, SUBACK and what topic they have chosen)
6. The choice of the topic for the SUB messages is fixed. It is equal to remainder of the division between the node ID and 3 since we have 3 different topics.
7. We put a sleep of 16 seconds in the code (line n. 248 – Project1C.nc). In this way we have emulated a periodic sending to Node-RED and it can correctly manage the messages.
8. Due to the sleep function mentioned above, we set the number of events in the Python file (line 149 – RunSimulationScript.py) to 10000. In this way the execution lasts 8 minutes with 20 PUB messages more or less.
9. We used a TCP connection between TinyOS and Node-RED.

TOPOLOGY.TXT

The node 1 is the PAN Coordinator, instead, the nodes from 2 to 9 are the nodes connected to it. (In the picture below, the nodes are mote 1 to mote 8)

```
1 2 -60.0
2 1 -60.0
1 3 -60.0
3 1 -60.0
1 4 -60.0
4 1 -60.0
1 5 -60.0
5 1 -60.0
1 6 -60.0
6 1 -60.0
1 7 -60.0
7 1 -60.0
1 8 -60.0
8 1 -60.0
1 9 -60.0
9 1 -60.0
```

This is a picture of the whole infrastructure:



PROJECT1.H

Here is defined the structure of messages that the PANC and the nodes can exchange each other. As mentioned in the “Assumptions” part, there those arrays used for keeping track of messages received or sent to and by nodes.

```
#ifndef PROJECT1_H
#define PROJECT1_H

#define MESSAGE_BUFFER 5

typedef nx_struct Msg
{
    nx_uint8_t type;
    /*
    * type = 0 -> CONN
    * type = 1 -> CONNACK
    * type = 2 -> SUB
    * type = 3 -> SUBACK
    * type = 4 -> PUB
    */
    nx_uint8_t sender;
    nx_uint8_t dest;
    nx_uint8_t data; // Field containing the value to send to Node-RED
    nx_uint8_t topic;
    /*
    * topic = 0 -> TEMPERATURE
    * topic = 1 -> HUMIDITY
    * topic = 2 -> LUMINOSITY
    */
}msg_t;

uint16_t indexConnReceived[8] = {0}; // Contains which node has correctly sent a
CONN message to PAN
uint16_t indexConnAckReceived[8] = {0}; // Contains which node has correctly
received a CONNACK message
uint16_t indexSubReceived[8] = {0}; // Contains which node has correctly sent a SUB
message to PAN
uint16_t indexSubAckReceived[8] = {0}; // Contains which node has correctly
received a SUBACK message
// Contains the subscribed topic of each node. Filled with 3 because we cannot have a
topic that has "3" as integer
uint16_t indexSubbedTopic[8] = {3, 3, 3, 3, 3, 3, 3, 3}; // Contains PUB messages
// 5x5 because we have 5 variables in our msg struct and 5 rows because it's enough
to contain messages. It's like a queue.
uint16_t buffer[MESSAGE_BUFFER][5] = {{0, 0, 0, 0, 0},
                                         {0, 0, 0, 0, 0},
                                         {0, 0, 0, 0, 0},
```

```
                                {0, 0, 0, 0, 0},  
                                {0, 0, 0, 0, 0}};  
  
enum  
{  
    AM_RADIO_COUNT_MSG = 6,  
};  
  
#endif
```

PROJECT1APPC.NC

Here you can find all the components we used to develop the project.

```
#include "Project1.h"

configuration Project1AppC {}

implementation
{
    components MainC, Project1C as App;
    components new AMSenderC(AM_RADIO_COUNT_MSG);
    components new AMReceiverC(AM_RADIO_COUNT_MSG);
    components new TimerMilliC() as MilliTimer;
    components ActiveMessageC;
    components RandomC;

    App.Boot -> MainC.Boot;
    App.Receive -> AMReceiverC;
    App.AMSend -> AMSenderC;
    App.AMControl -> ActiveMessageC;
    App.MilliTimer -> MilliTimer;
    App.Packet -> AMSenderC;
    App.Random -> RandomC;
}
```

PROJECT1C.NC

In this file we implemented all the functionalities requested.

Definition of components for connecting to Node-RED

We used two additional libraries, and we defined two global variables in order to set the server address and the service port.

```
#include <arpa/inet.h>
#include <sys/socket.h>

#define SERVER_IP "127.0.0.1" // Localhost IP address
#define SERVER_PORT 1234 // Port number for the server
```

Event: Boot.booted()

It starts the system through the AMControl interface.

Event: AMControl.startDone(error_t err)

If there is no error, it prints on the simulation output and calls a Timer to start the execution.

Event: AMControl.stopDone(error_t err)

If called, it prints a message on the simulation output.

Event: MilliTimer.fired()

When called from the “AMControl.startDone” event, every node create a CONN message and send it to the PANC.

Event: Receive.receive(message_t* bufPtr, void* payload, uint8_t len)

This is the core event where all the messages are handled depending on which receive them.

This event is made of two main parts:

1. the PANC receives
 - a. If it receives a CONN message, it writes the node that sent the message in the array called “indexConnReceived” and, after this, it sends a CONNACK message to that node.
 - b. If it receives a SUB message, it writes the node that sent the message in the array called “indexSubReceived” and, after this, it sends a SUBACK message to that node.
 - c. If it receives a PUB message, it starts looking for an empty row in the matrix “buffer” because we want to save the message inside it. Then, it sends the PUB to all nodes that have a subscription to the same message topic (the retransmission is not handled, QoS = 0). In the end, it opens a connection to Node-RED and sends the message.

2. The nodes receive
 - a. If a node receives a CONNACK message and the relative position in the array “indexConnAckReceived” is zero (e.g. node 3 – position 2 in the array == 0), the same position will be marked with the node ID in order to remember which nodes have already received it. Then, it creates a SUB message and sends it to the PANC. (the choice of the topic is explained in the assumption section above)
 - b. If a node receives a SUBACK message and the relative position in the array “indexSubAckReceived” is zero (e.g. node 3 – position 2 in the array == 0), the same position will be marked with the node ID in order to remember which nodes have already received it.
Once this is done, there is a check to be sure that the node is effectively connected and subscribed. After this check, it can create a PUB message (using QoS = 0) with a random payload as requested and send it to the PANC.

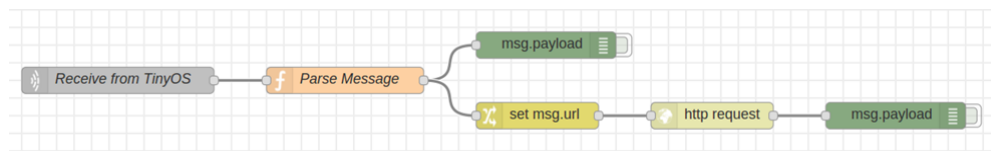
Event: AMSend.sendDone(message_t* bufPtr, error_t error)

It checks that messages are sent successfully, and it handles the retransmissions of messages if they get lost.

There are two checks:

1. if the respective node position in array “indexConnReceived” is zero or if the respective node position in array “indexConnAckReceived” is zero, this means that the message gets lost, so the node will resend a CONN message to the PAN.
2. if the respective node position in array “indexSubReceived” is zero or if the respective node position in array “indexSubAckReceived” is zero and, if the respective node position in array “indexConnAckReceived” is NOT zero, this means that the node has received a CONNACK but not a SUBACK or the PAN has not received a SUB from the node, so the node will resend a SUB message to the PAN.

NODE-RED FLOW



Receive from TinyOS

It is the component that use TCP protocol and listens to the port “1234” that is the one set for Node-RED.

Parse message

It parses messages to extract the topic and the payload and creates the URL for the http request to our ThingSpeak public channel.

THINGSPEAK

[Link to our public channel](#)

In the charts below, we set the max number of the chart points as 10, just for a better visualization.

Project IoT

Channel ID: **2185815**
Author: **mw0000028800397**
Access: Public

☒ Export recent data

MATLAB Analysis

MATLAB Visualization

