IOT project documentation

Project number 1:

Preparation: First of all, we need to modify a little bit **RunSimulationScript.py** file add our custom channels, and initialize the nodes (9 nodes in this project). Then we should define the topology of the network and identify which node is connected to which one, here all nodes are connected to node 1(broker). Inside the **Mqtt.h** file, we defined the body of the messages and also some constant variables for declaring the type of the messages. Finally, inside the **MqttAppC.nc** file, we defined the components and the interfaces of the network.

Main code: The rest of the description is related to the **MqttC.nc** file.

Inside the **Boot.booted()** function we initialize the nodes with interested topics and published topics manually.

Connection:

Inside the **send_connect_msg** function, we make a connection message with the corresponding payload and then call the **generate_send** function to send the message. Every 5 seconds, we call the **TimerConn** to check whether the ConnAck message was received or not. If the **ConnAck** is not received, then again **send_connect_msg** will be called.

Inside the **send_connect_msg_ack** function, ConnAck message is created and directed to the broker with a specific payload.

The connection message always is with type =0. The topic and value of this kind of message are meaningless. For ConnAck message type is 1 and again topic and value are meaningless.

Finally, from lines **385** to **412** we check the connection of the node to the broker and if the node is already connected, we do nothing and if not, we try to add the node inside the **connection_status_devices** list. Then the conAck message will be sent. Finally, if the connack message is received successfully then we must stop the timer and avoid sending more connect messages to the broker.

Subscribe:

First, inside the **sub_msg_send** function, we create the subscribe message to send it to the broker. The type of this message is 2 and the destination node is 1 and the topic must be selected. This subscription message will be sent every 5 sec till the node successfully subscribes to the topic.

The second function is **sub_msg_ack**. Broker replies to the node an ACK message with the type of 3.

From lines **414** to **444** we also check the subscribing messages. We add the connected device to the list **connection_status_devices**. Then we try to add the node for the requested topic by modifying the **subscription_matrix** list. Finally, we call the **sub_msg_ack** to send back the ACK message. We check if a SUBACK message is received then we must stop the timer and not send the subscription message anymore.

Publish:

Inside the **send_pub_msg** function, the publish message is created. Here the message value is a random value that is generated and sent to the broker every 3 sec.

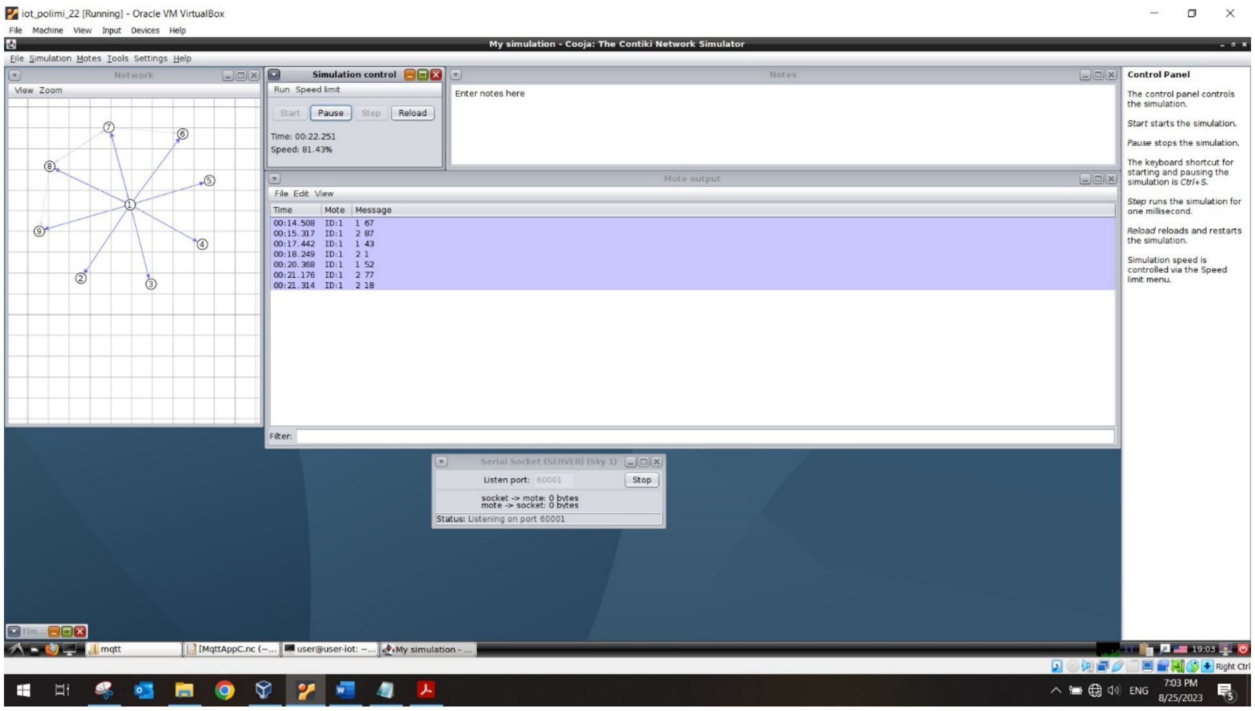
In the "**publish**" function, first, we check the **message_list** to get the topic. Then inside the **message_queue** we find the message to be published. If this list is empty then we simply return nothing. Then we make the packet with proper values and we will send this packet to subscribed nodes. When a message is successfully forwarded to a node we check if we should continue or not, and if we should, the "**publish**" function is called again and the **message_list** will be updated.

From lines **446** to **481** we are controlling the publish messages. When the broker receives a Publish message, the connection must be checked. Then we append the topic to **message_list** array. We save the early coming message inside this array and they must be published first. Then we save the message in **message_queue**. Finally, the publish function is called to publish the message to the subscribed nodes.

Important: Lines 472 and 473 in MqttC.nc and 27,28 in MqttAppC.nc must be commented for Tossim simulation and uncomment for cooja and node-red.

Cooja and Thingspeak simulation:

We need to configure the cooja environment and set 9 nodes (1 PAN coordination and 8 sensors) as sky motes. Also, we put the serial socket (Server) on port 60001 and started the simulation. Below is a picture of cooja environment:



iot_polimi_22 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
Simulation control
Run Speed limit
Start Pause Step Reload
Notes
Enter notes here
Mote output
File Edit View
Filter:
Serial Socket (SERVER) (Sky 1)
Stop

```
Time: 00:22.251
Speed: 81.43%
```

```
Time       Mote   Message
00:14.508  ID:1   1 67
00:15.317  ID:1   2 87
00:17.442  ID:1   1 43
00:18.249  ID:1   2 1
00:20.368  ID:1   1 52
00:21.176  ID:1   2 77
00:21.314  ID:1   2 18
```

```
Listen port: 60001
socket -> mote: 0 bytes
mote -> socket: 0 bytes
Status: Listening on port 60001
```

Control Panel

The control panel controls the simulation.

Start starts the simulation.

Pause stops the simulation.

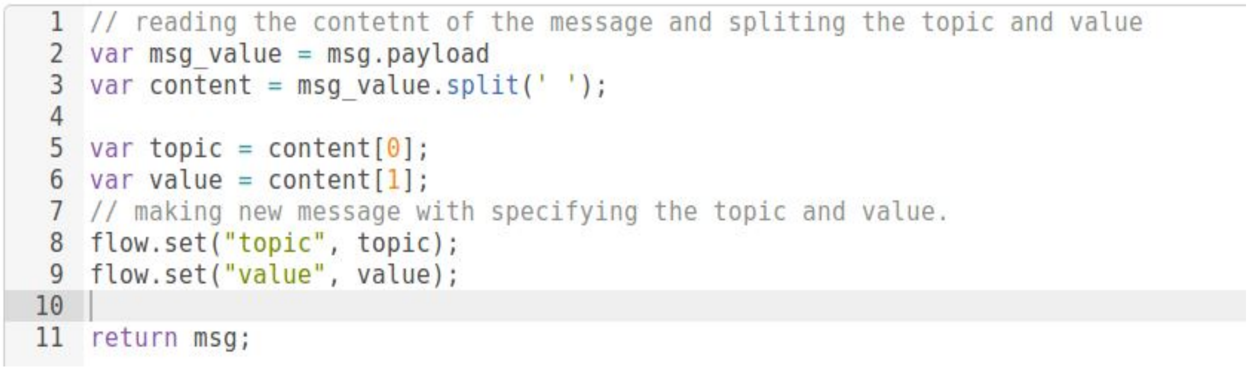
The keyboard shortcut for starting and pausing the simulation is *Ctrl+S*.

Step runs the simulation for one millisecond.

Reload reloads and restarts the simulation.

Simulation speed is controlled via the Speed limit menu.

Then we should open the node-red, and make a connection with cooja. We need to read the messages and send values to the Thingspeak server. By doing so, we make a TCP connection to the local host with port 60001. Then we have a function for message preparation, and we read the message and split the topic and value. Below is the code:

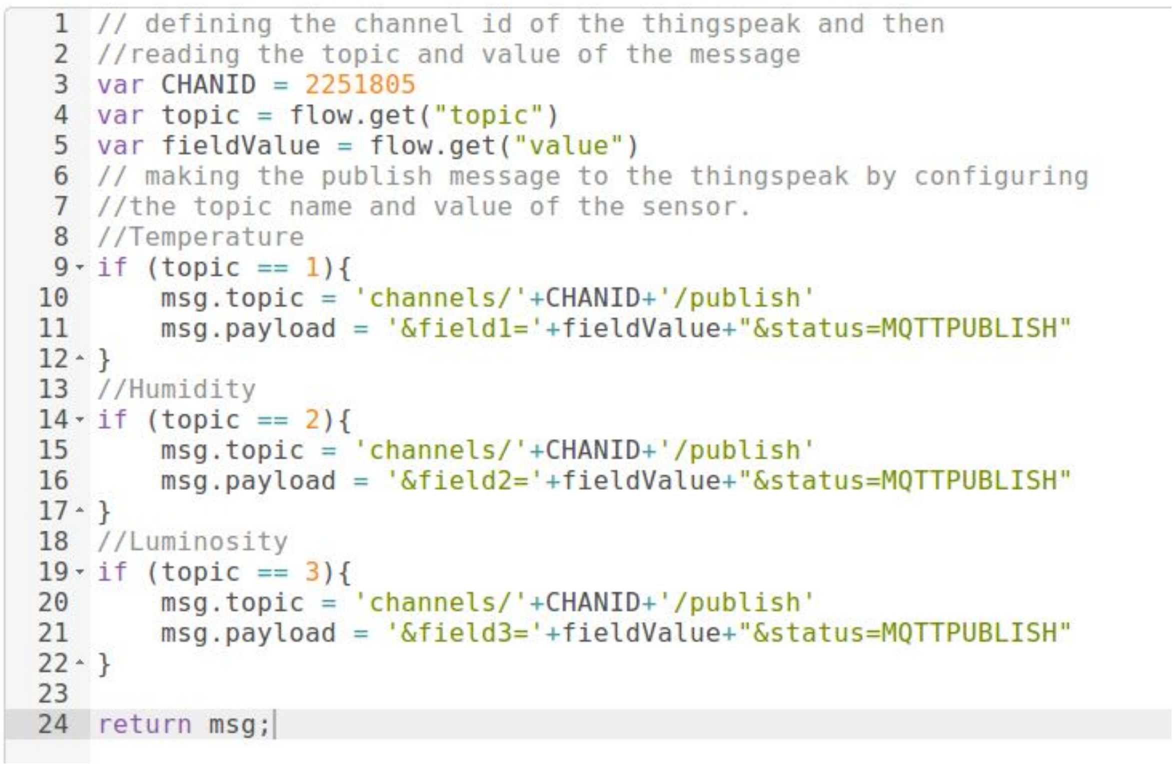


```
// reading the contetnt of the message and spliting the topic and value
var msg_value = msg.payload
var content = msg_value.split(' ');

var topic = content[0];
var value = content[1];
// making new message with specifying the topic and value.
flow.set("topic", topic);
flow.set("value", value);

return msg;
```

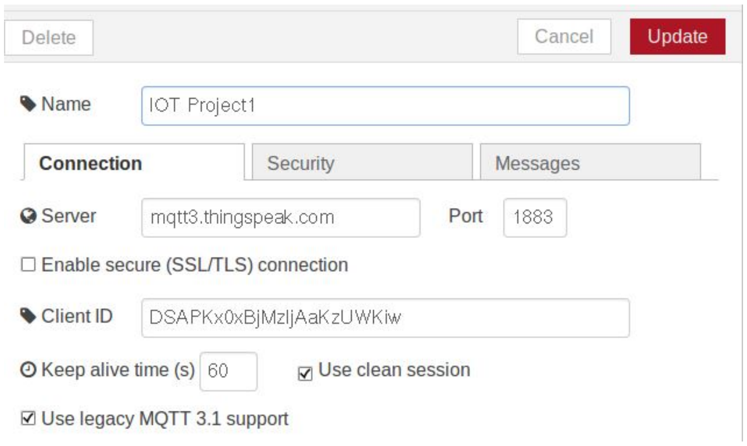
Then we have another function for configuration of the server and making messages compatible with the server requirements, and below is the content of the function:



```
// defining the channel id of the thingspeak and then
//reading the topic and value of the message
var CHANID = 2251805
var topic = flow.get("topic")
var fieldValue = flow.get("value")
// making the publish message to the thingspeak by configuring
//the topic name and value of the sensor.
//Temperature
if (topic == 1){
    msg.topic = 'channels/'+CHANID+'/publish'
    msg.payload = '&field1='+fieldValue+"&status=MQTTPUBLISH"
}
//Humidity
if (topic == 2){
    msg.topic = 'channels/'+CHANID+'/publish'
    msg.payload = '&field2='+fieldValue+"&status=MQTTPUBLISH"
}
//Luminosity
if (topic == 3){
    msg.topic = 'channels/'+CHANID+'/publish'
    msg.payload = '&field3='+fieldValue+"&status=MQTTPUBLISH"
}

return msg;
```

For making a connection with Thingspeak we need to sign in and make a channel. Then we select 3 fields as far we have 3 sensors. Then we must add an mqtt device to this channel. We should configure this mqtt device information inside the mqtt node of the node-red. We should specify the client ID, Username, and password to make the connection. The next 2 screenshots show these configurations:



Name: IOT Project1

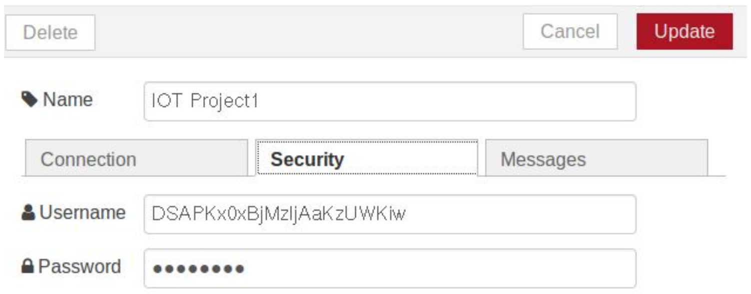
Server: mqtt3.thingspeak.com Port: 1883

☐ Enable secure (SSL/TLS) connection

Client ID: DSAPKx0xBjMzljAaKzUWKiw

Keep alive time (s): 60 ☑ Use clean session

☑ Use legacy MQTT 3.1 support



Delete
Cancel
Update

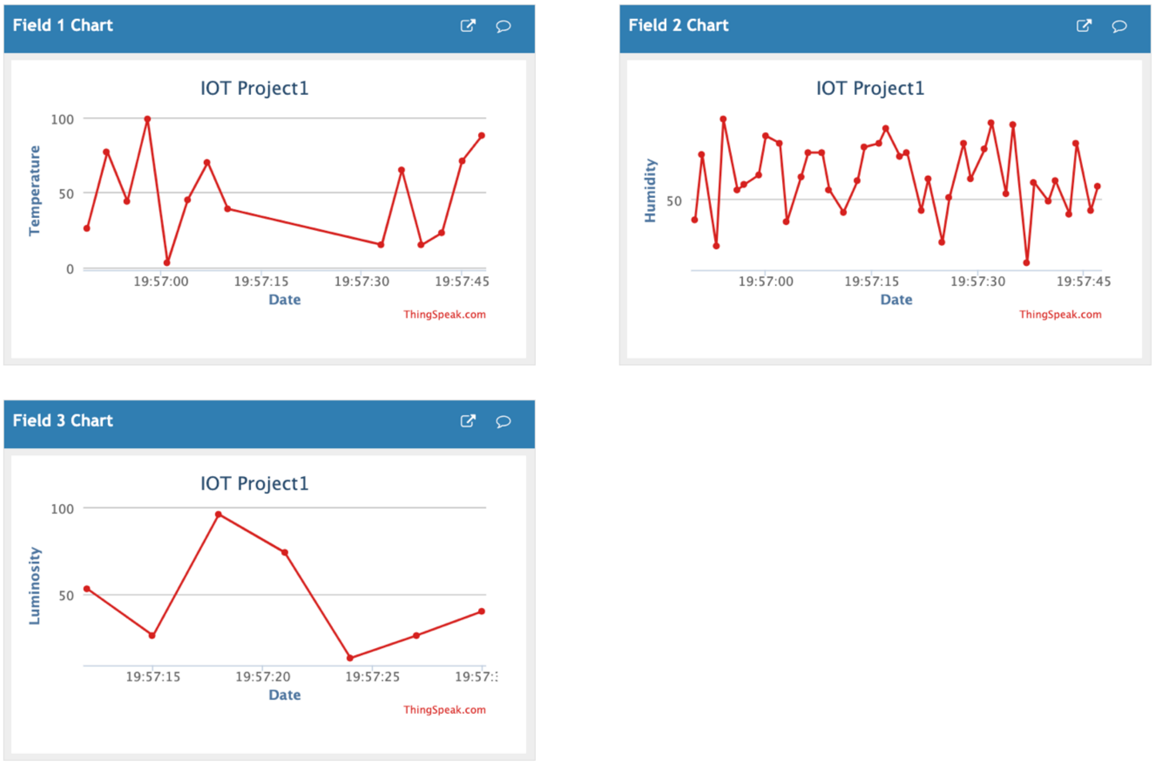
Name: IOT Project1

Connection | Security | Messages

Username: DSAPKx0xBjMzljAaKzUWKiw

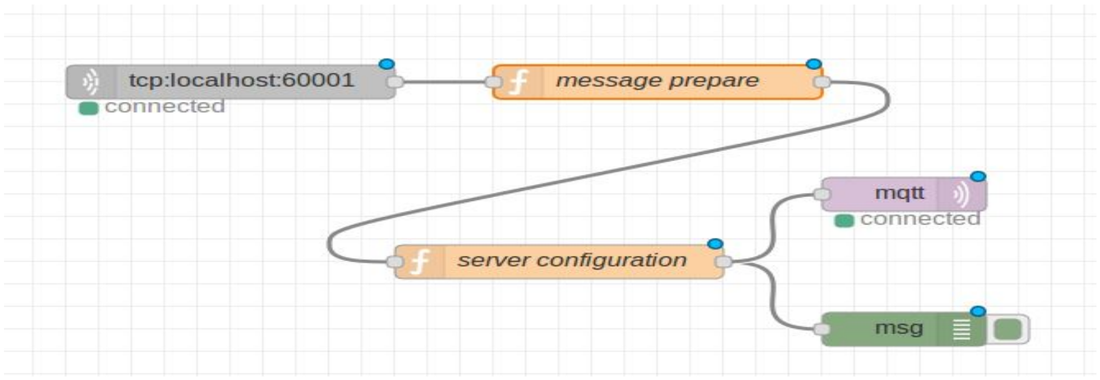
Password: ●●●●●●●●

Finally, the connection with Thingspeak is done correctly and we can see the charts.



And the Thingspeak channel link is this: https://thingspeak.com/channels/2251805

And here is the big picture of node-red flow:


tcp:localhost:60001
connected
message prepare
mqtt
connected
server configuration
msg