Environmental Monitoring using IoT Devices - Contiki NG/Node Red

Bruno Morelli Cristian Lo Muto Vincenzo Martelli

Contents

1 Introduction

IoT devices are used to measure temperature and humidity every T=10 sec. A sliding window is applied that computes the average of the last six readings. Should the value of the average exceed a certain threshold K, the raw readings are reported instead of the average obtained from the sliding window. The average or the readings are reported to the backend. At the backend, information on the hottest, coolest, and most/least humid day of the month is kept in a log that is periodically communicated via email to a speci1c address. This log is persistently stored so rebooting the backend will not make the system lose the data gathered until that time.

2 Approach and Assumptions

This section outlines the methodology and underlying assumptions employed in this project to achieve the desired outcomes.

2.1 Contiki NG

Our approach for simulating IoT devices that use Contiki NG involves a network of COOJA Motes that communicate using the *RPL Tree Protocol*. The root of the tree communicates with the external network through an IPV6 border router, which serves as the gateway for the IoT devices to connect to the internet. The simulation assumes that the devices are constantly reachable from the border router, either directly or through multiple hops, to simulate a real-world scenario where IoT devices can move around and still maintain their connectivity. Temperature and humidity data (both averages or raw reading) are communicated using the MQTT protocol. QOS is set to "0" for two main technical considerations: Firstly, since the application is not safety critical in nature, the loss of occasional readings is deemed acceptable. QoS 0, also known as "at most once" delivery, provides a best-effort approach where messages are delivered to the broker without any acknowledgment or guarantee of successful delivery. In scenarios where occasional data loss does not significantly impact the overall functionality or purpose of the application, QoS 0 serves as a suitable choice. Furthermore, selecting QoS 0 reduces the overhead on the IoT devices themselves. QoS levels higher than 0, such as QoS 1 ("at least once") or QoS 2 ("exactly once"), introduce additional complexity and overhead due to the need for acknowledgment, retransmission, and duplicate suppression mechanisms. These additional operations require more processing power, memory, and network resources on the IoT devices, potentially straining their limited capabilities. By opting for QoS 0, the overhead on the IoT devices is minimized, ensuring efficient resource utilization and maintaining a lightweight operation. The simulation in this project utilizes the *Constant Loss Unit-Disk Graph Model (CL-UDGM)*, where nodes within a certain range can directly communicate and nodes outside this range experience complete signal loss. The interference range is set to 0 meters, assuming no interference or attenuation between neighboring nodes. This choice simplifies the simulation and enables targeted investigations into specific aspects such as the routing protocol and the MQTT protocol.

2.2 Node-RED

The technology used for the backend is Node Red, this was an easy choice thanks to its ease of use and the integrated nodes to receive the data, send them via email and create a persistent

storage. The persistency was achieved saving the log in a json file after the end of each month. To support intra-month persistency Another json file is saved containing the cumulative temperature and humidity of the current day, the number of readings and the current max and min of the month plus other information that will be seen more in detail later.

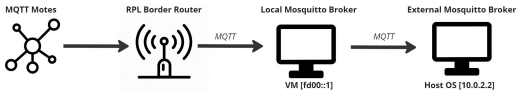
3 Design

3.1 Contiki NG

The application consists of two types of motes: an MQTT Mote and an RPL Border Router Mote

3.1.1 RPL Border Router Mote

This mote is the root of the RPL tree and acts as a bridge towards the external network. More specifically, It receives MQTT messages from the IOT devices in the tree and casts them to a local Mosquitto broker running inside the VM environment. For the sake of project presentation and ease of configuration, the decision was made to recast MQTT messages from the local Mosquitto broker to an external Mosquitto broker running on the host operating system, although it is technically possible to forward the messages to an external broker anywhere on the internet.



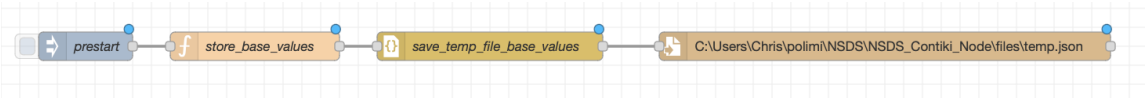
3.1.2 MQTT Mote

The MQTT motes in the application function as a finite state machine that is triggered periodically by a timer. Each time the timer reaches zero, the motes send an MQTT message containing a random temperature and humidity reading. The lasts six readings are kept in a local buffer in order to calculate averages. However, if the calculated average exceeds a predetermined threshold value, denoted as K, the raw temperature and humidity values are reported instead. This threshold-based mechanism allows for special handling of outlier conditions or significant variations in the collected data. By employing this approach, the application achieves the objective of providing both average readings for normal scenarios and raw readings when exceptional conditions are detected.

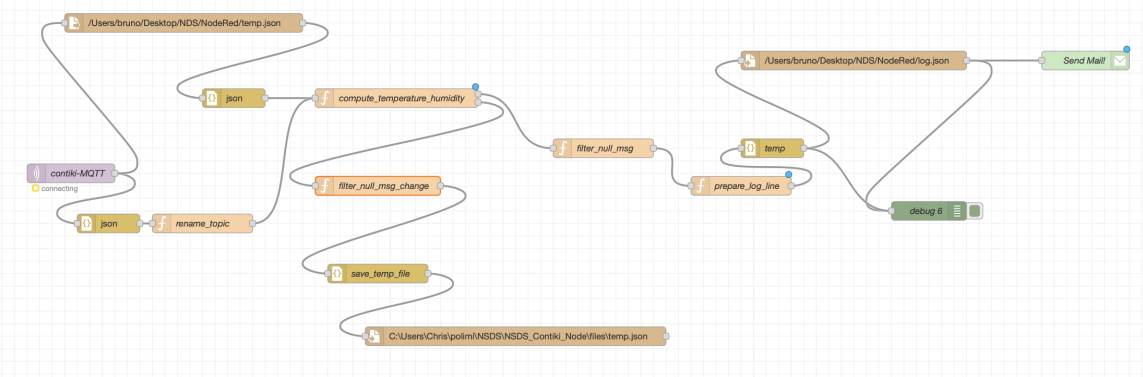
3.2 Node-Red

The back-end consists in two flows:

- **setup flow**: a simple flow that is used only to setup the system before the first round. This flow starts when the 'prestart' node is triggered and consists of a function that creates a message with hard-coded values that are sent to the next nodes which creates a json file out of these values that is eventually saved on storage by the last node of this flow. This file is called 'temp.json' and is saved to the directory specified in the node.



- **main flow**: here is where most of the work happens. The most important node is *compute_temperature_humidity*, it receives the new readings and keeps a running sum of the temperature and humidity of the current day. After every computation it reports these values to a file called *temp.json*. The same happens also for the minimum and maximum values, at the end of each day this node computes the means and compare them with the previous notable values and saves them in the same temp file the new values. This node has also another output exit, this is used at the end of each month to send the the values that have to be written in the log and sent as an email. The log is written on another file called *log.json*.
  In order to provide persistency, every time a message arrives from the front end the *temp.json* file is loaded and sent to the *compute_temperature_humidity* node, this will use these information only if the system had just restarted else it will discard this message. This file will contain the default values or the last values saved before the crash, obviously we must assume that the system does not crash while writing the file.
  Another approach was to connect this branch to a trigger instead of the MQTT receiver. This would have saved reads from disk but would have decreased user experience, with this approach crashes are completely transparent to the user. This could also lead to the lost of the first sensor reading after the restart if it arrives to *compute_temperature_humidity* before the other branch but it was assumed that one read would not change the average value of the day in a significant way.
  It is important to state that all of these writes and reads of file are possible thanks to slow arrival rate of sensor readings.



To go more in depth of the two json file used:

- **temp.json**: this file is composed of the following attributes:
  - **current_day**: self explanatory, default value is 1
  - **current_month**: self explanatory, default value is 1
  - **current_year**: self explanatory, default value is 2000

  - **current_day_temp**: sum of the temperatures read in the current day, default value is 0
  - **current_day_humidity**: sum of the humidity read in the current day, default value is 0
  - **max_temp**: maximum temperature registered in the current month, default value is 0
  - **min_temp**: minimum temperature registered in the current month, default value is 100
  - **max_humidity**: maximum humidity registered in the current month, default value is 0
  - **min_humidity**: minimum humidity registered in the current month, default value is 100
  - **max_temp_day**: day of the month in which the maximum temperature was registered, default value is 0
  - **min_temp_day**: day of the month in which the minimum temperature was registered, default value is 0
  - **max_humidity_day**: day of the month in which the maximum humidity was registered, default value is 0
  - **min_humidity_day**: day of the month in which the minimum humidity was registered, default value is 0
  - **reading**: number of readings received in the current day, default value is 0

- **log.json**: this file is a sequence of strings, there is one line per month with all the information for that month. An example of a line is:
  max temp for 07/2023 is 33°C on day: 1, the min is 32°C on day: 2, max humidity is 100% on day: 2, min humidity is 42% on day: 1.

4 Test

4.1 Installation

The installation process outlined in this section of the documentation covers the setup and configuration of Contiki NG and Node-Red.

4.1.1 Node-Red

Once Node Red is running the only thing that has to be done before the system is operational is launching the prestart flow, this will create the temp.json file that is used to load the initial values in the main function. However this should be done just once, if the system fail and is rebooted the prestart should not be launched else the stored information will be overwritten. The recovery after the crash is indeed performed automatically by the system provided that the temp.json file was not moved or deleted by the user.

4.1.2 Contiki NG

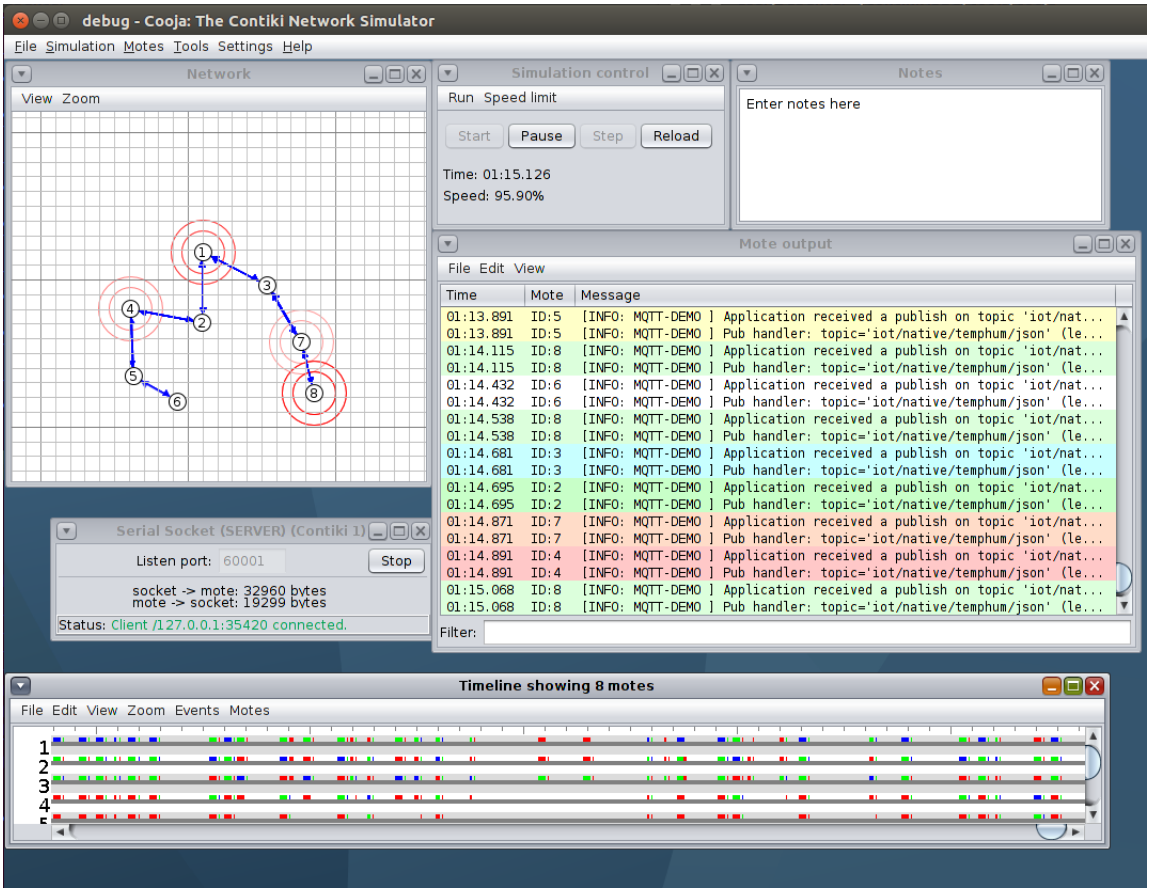
To initiate the COOJA simulator and execute the simulation for the Contiki NG IoT devices:

1. Open a terminal window and navigate to the directory where the COOJA simulator is installed.
2. Run the command "`ant run`" to launch the COOJA simulator.
3. Select the "simulation.csc" file located in the project folder to build the simulation
4. Compile and run "MQTT_Mote" and "RPL_Border_Router".
5. After the simulation has started, launch the client side of the RPL border router by navigating to the directory `contikiNSDS/src/contiki/rpl-border-router` and run the command

   `make TARGET = cooja connect-router-cooja`.
6. Kill the current instance of Mosquitto running in the VM
7. Run another instance of Mosquitto with the configuration file "mosquitto.conf" located in the root folder of the project.

   ```
   mosquitto -c mosquitto.conf
   ```

4.2 Simulation



5 References

- Contiki NG Documentation
- Node-RED Documentation