Project Documentation

182.753 Internet of Things (VU 4,0) 2017W

Philipp Raich* Mino Sharkhawy† Thomas Weber‡

February 26, 2018

Contents

*philipp.raich@tuwien.ac.at
†mino.sharkhawy@student.tuwien.ac.at
‡e1025654@student.tuwien.ac.at

1 Introduction

This document serves as documentation and lab protocol for the lab part of the *Internet of Things* course at TU Vienna (WS2017). It contains the given assignment, details about the given tasks and questions, and finally a description of the team's submitted work.

2 Assignment

The context of the assignment was given as: Multiple low-power **sensor nodes**/MicroController Units (MCUs) shall be connected to a special purpose, **WiFi-enabled MCU** using a generic Radio Frequency (RF)-module. The WiFi-enabled MCU will act as a **gateway** for the sensor nodes, i.e. it is intended to deliver or save the sensor readings and serve them over WiFi either (a) as ***MQTT* client** to a broker, and thus consumers/subscribers, or (b) as a web server, serving a **homepage** with sensor charts over HTTP.

Further, the RF protocol used between the sensor nodes and the gateway device must focus on **low energy consumption**, to extend the battery life of the sensor nodes, through e.g. Radio Duty Cycling (RDC) and similar measures. The gateway device will be connected to an outlet and does not have to hold up to the same scrutiny. The sensor-nodes and the gateway device to be used will be supplied. The project work consist of writing the necessary software and evaluating the end result.

The RF-modules to be used for the low power radio communication (LP-RF) are based on the 2.4GHz **nRF24L01+**[1], which implements the proprietary "Enhanced ShockBurst" protocol. The sensor-nodes are compatible to "Arduino Pro Mini 3.3V" (ATmega328P@MHz) devices[2].

The given cornerstones and deliverables were determined as following:

Cornerstones

- Design a communication protocol based on the nRF24L01+ "Enhanced SchockBurst" protocol. Consider the energy consumption and limit it to a possible minimum (battery powered sensor nodes).
- Make the sensor data available to the "home network", i.e. to conventional OTS hardware, via WiFi or Ethernet (via gateway device).
- The data should be protected from access by third parties, even if they have access to the home network (authenticated access).

For the sake of focus on the main scope of the project, the following simplifications apply: (a) The sensor readings can be "simulated"/mocked, and (b) security considerations regarding the low-power RF-protocol can be widely ignored.

[1] http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24LU1P
[2] https://store.arduino.cc/arduino-pro-mini

Deliverables

- All the written **code**.
- A **presentation** of the completed project shall be given.
- A clean **documentation** and **lab-protocol** must be submitted (this file). The documentation shall complete the submitted code and explain how the tasks were solved.

3 Solution

3.1 Architecture

We chose to follow variant *(a)* of the assignment, i.e. to map the messages from the sensor-nodes to MQTT on the gateway node (see Fig. 1). Additionally, a dedicated device (BeagleBone Black Industrial[3]) was employed to cover the following tasks: (a) act as a **WiFi Access Point (AP)** (and DHCP), (b) run a **MQTT broker**, and (c) serve as clock source. *hostapd* and *dnsmasq* were used to serve the AP functionality, while *mosquitto* was used as a MQTT v3.1/v3.1.1 compliant MQTT broker.

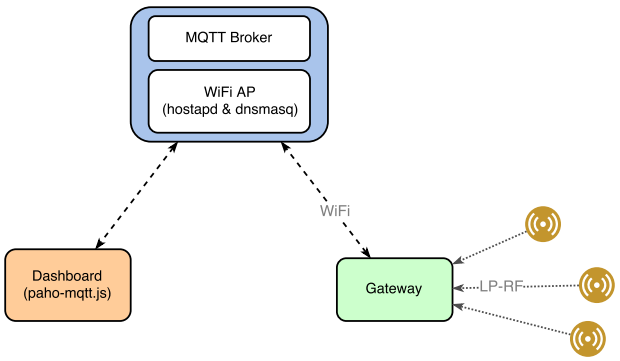

Figure 1: Overview over the chosen architecture.

Diverging from the task description, a HTML **dashboard** was developed *additionally* to the MQTT broker, which would receive the data from the MQTT broker via a JavaScript MQTT client (over websockets).

This architecture of loosely coupled components allows for a robust implementation that avoids interferences between components, which are a common threat on Internet of Things (IoT) solutions that employ "jack-of-all-trades" components. Further, established libraries and packages are available, and the team can concentrate on the main tasks of the project. Moving as much as logic as possible from particularly the underpowered gateway was considered a main priority.

[3]https://beagleboard.org/arrowbbbi

3.2 Security

The following security threats were identified:

- eavesdropping (confidentiality),
- data manipulation/message forgery (authenticity).

To protect the MQTT messages/connections against these threats, **Transport Layer Security (TLS) with client certificates** were used, to authenticate the participating endpoints and protect the transmitted data. Additionally, **user-name and password authentication** was used for MQTT client authentication with the broker (see MQTT configuration in Section 3.3.4).

To protect the LP-RF protocol between nodes and gateway, keyed-Hash Message Authentication Code (HMAC) with timestamped messages was used (see Section 3.3.1), using pre-shared keys deployed at programming. The timestamped messages allow for detection of replay attacks, since old/repeated messages can be discarded. Please note that HMAC will not protect against eavesdropping.

3.3 Parts

The parts of the solution thus are:

- A low-power RF protocol for communication between sensors and gateway (Section 3.3.1)
- Sensor node program (Section 3.3.2)
- Gateway program, with MQTT topic mapping (Section 3.3.3)
- A MQTT broker and respective configuration (Section 3.3.4)
- A HTTP Dashboard (Section 3.3.5)

3.3.1 Low Power RF Protocol

A enhanced library around the "Enhanced ShockBurst"-API was designed, specific to the requirements of the given task. The public functions of the API are listed in listing 1.

Listing 1: "RF Wrapper Library API"

```
int rflib_sensor_init(uint16_t cepin, uint16_t cspin, uint8_t channel,
          uint64_t address, uint8_t delay, uint8_t retransmits);
void rflib_sensor_tx_pre(void);
int rflib_sensor_tx(const struct rflib_msg_t *msg, struct rflib_msg_t *ackmsg);
void rflib_sensor_tx_post(void);

int rflib_coordinator_init(uint16_t cepin, uint16_t cspin, uint8_t channel,
         const uint64_t *addresses, uint8_t n_addresses,
         uint8_t delay, uint8_t retransmits);
int rflib_coordinator_available(void);
void rflib_coordinator_read(struct rflib_msg_t *msg);
int rflib_coordinator_set_reply(uint8_t address_idx, struct rflib_msg_t *msg);
void rflib_coordinator_clear_reply(void);
```

The API is split into *sensor* and *coordinator* specific functions, to accommodate the distinction between highly constrained sensor nodes and the "unconstrained" gateway device, which acts as

coordinator. The sensor-specific interface thus includes the `rflib_sensor_tx_pre()` and `rflib_sensor_tx_post()` functions, that allow the sensor to execute specific actions before and after sending data. In particular, the RF-chip on sensor nodes is **disabled** between sending messages, as are all unnecessary parts of the system, e.g. any connected sensors, and the attached RTC-chip is set to wake the system at the latest configurable time, or when the next sensor message is due, whichever comes earlier. This allows to reduce the power consumption to a minimum.

Message Authentication HMAC was employed to authenticate and verify the integrity of individual messages, using the Secure Hash Algorithms (SHA)-1 function (HMAC-SHA-1) with pre-shared keys. The keys are deployed on the nodes during programming. This protects the messages sent over the LP-RF network against message forging. Replay attacks are averted by including a timestamp in the message (see listing 2 and Section 3.3.1).

Energy Considerations Special care had to be taken to avoid unnecessary energy consumption by the RF-module on the sensor nodes, which together with the peculiarities of the "Enhanced ShockBurst"-protocol lead to the following measures:

- The RF-module on the sensor nodes was **disabled whenever possible**, and only enabled for data transmissions.
- Sensors were disabled/shut-off when not needed, and only enabled when actively taking measurements.
- Data can still be received by the sensor nodes by "piggybacking" on ACKs (i.e. in response to packets from the sensor) (cf. Section 3.3.1).
- The interval at which the sensor nodes wake up for transmissions is configurable.

Message and Data Format Protocol Buffers[4] were used to format the messages across RF nodes and gate. listing 2 shows the corresponding structure.

Listing 2: "Message Format as Protocol Buffer definition"

```
message Command {
  enum CommandType {
    NEW_UPDATE_INTERVAL = 1;
  }
  required CommandType type = 1;
  optional sfixed32 param1 = 2;
  optional sfixed32 param2 = 3;
  optional sfixed32 param3 = 4;
}

message N2C {
  required fixed32 timestamp = 1;
  required fixed32 roomNo = 2;
  required uint32 nodeId = 3;
  required uint32 sensorId = 4;
  enum SensorType {
```

[4]https://developers.google.com/protocol-buffers/docs/proto

```
    TEMPERATURE = 1;
    HUMIDITY = 2;
  }
  required SensorType type = 5;
  required float data = 6;
}

message C2N {
  required fixed32 timestamp = 1;
  optional Command command = 2;
}
```

2 different types of messages are possible, N2C for node to coordinator messages (sensor values) and C2N for messages from coordinator to the nodes, which will are appended to the ACKs to the nodes on reception of a message from the nodes. The messages from coordinator to the nodes can *additionally* contain a new update interval, to set the sensor sending frequency of the nodes.

Messaging Sequence To counteract replay attacks (cf. Section 3.2), messages from the coordinator contain a **timestamp** and are rejected by the nodes if the timestamp is *not* newer than the last known timestamp. The coordinator on the other hand will reject messages that contain a timestamp outside of the update interval of the sensors. Since this requires the nodes and coordinator to agree on a common (although not necessarily perfectly accurate) time, it is required that the coordinator • has a monotonous clock source, and • "broadcasts" timestamps to the sensor nodes.

The clock source is constructed by regularly publishing *a* current time to the MQTT topic

```
IoT/Time
```

which the coordinator is subscribed to, in order to update its own time and to send the nodes time updates (via the respective ACKs).

Known Issues

- HMAC will only guard the LP-RF protocol against forgery, and together with timestamps against replay attacks, but not against eavesdropping. This was accepted, since HMAC already exceeds the assignment specification, and encryption was not a requirement.

3.3.2 Sensor Node

The sensor nodes were designed to deliver sensor data (at a configurable frequency) to the coordinator, using the protocol depicted in Section 3.3.1.

For the data to be submitted, sensor mockups (generated data) were used to simulate actual sensors connected to the sensor nodes.

As a proof of concept, an available moisture sensor was connected to the sensor board. The pinout is shown in Figure 2.

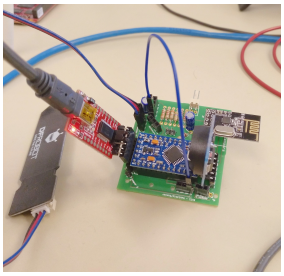

Figure 2: PoC: Moisture Sensor Pinout

Listing 3: "Config struct for the Nodes"

Configuration

```
struct rfnode_config cfg_node1 = {
  /* operations */
  .sensors = sensors_node1,
  .room_number = 1,
  .node_id = 1,
  .wakeup_interval = 30,
  /* communications */
  .address = 0xF0F0F0F0E1LL,
  .channel = 0,
  .delay = 15,
  .retransmits = 15,
  /* security */
  .auth_key = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  /* hardware */
  .cepin = 9,
  .cspin = 10,
  .rtcpin = 2,
  .rtcint = 0,
  /* debug */
  .baud_rate = 57600,
  .debug = DEBUG_ALL,
};
```

3.3.3 Gateway

The main goals of the gateway device are:

- Act as a **border router** between the two distinct networks[5].
- Act as the **coordinator** for the sensor network.
- Map between sensors nodes (respectively their sensors) and MQTT topics.
- Relay control commands to the sensor nodes.

For the communication with the sensor network, the gateway thus is equipped with the same RF-module (nRF24L01+) as the sensor nodes, and uses the RF-wrappers from Section 3.3.1, although the coordinator-specific functions. For this purpose, the pinout shown in Fig. 3 was used.

[5]With reduced capabilities as a non-transparent gateway, since it acts as a proxy and bridges merely on the application-layer.

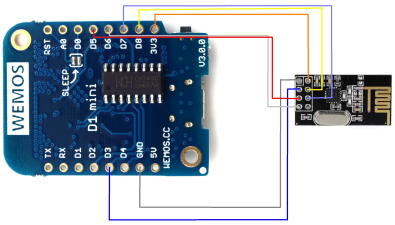

Figure 3: Coordinator Pinout

Configuration The configuration for the gateway is shown in listing 4.

Listing 4: "Config Struct for the Gateway"

```
struct coordinator_config esp_config = {
  .mqtt_server = "192.168.44.1",
  .mqtt_user = "IoT_client",
  .mqtt_password = "leafy_switch_soup",
  .mqtt_topic_timestamp = "IoT/Time",
  .mqtt_topic_update_interval = "IoT/Interval",
  .mqtt_port = 1883,
  .wifi_ssid = "IoT_17_18",
  .wifi_password = "heavy_cat_radiator",
  .address = 0xF0F0F0F0E1LL,
  .channel = 0,
  .delay = 15,
  .retransmits = 15,
  .auth_key = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  .cepin = 0,
  .cspin = 15,
  .baud_rate = 115200,
  .debug = DEBUG_OFF,
};
```

MQTT-Topic Automagic The gateway will enforce the mapping between the individual sensors and the respective MQTT topics, by constructing the topic from the contents of the LP-RF message (see Section 3.3.1) and augmenting it with the predefined topic parts:

```
IoT/Room3/Node5/Sensor2/temperature
```

where `Iot/Room`, `Node` and `Sensor` are predefined, and the effective room number (3), node number (5), sensor number (2) and measurement type (`temperature`) are defined through the message.

Control messages The update frequency of the sensor nodes can be adjusted by sending the desired interval (in seconds) to the topic

```
IoT/Interval
```

Messages sent to this topic are forwarded by the coordinator to the nodes upon reception of a respective message (since messages *to* the nodes are appended to ACKs).

Known Issues

- Due to the energy considerations of the protocol (cf. Section 3.3.1), the sensors will adopt a new interval after (worst case) $2 * old\_interval$ (where $old_interval$ is the previously set interval, e.g. the default interval), since sensors will only receive data from the coordinator after they send data to the coordinator.

3.3.4 Data Broker

Following the given conditions by the chosen exercise variant (see Section 2), MQTT was used as a data broker for all the data from and to the sensor network (cf. MQTT mapping in Section 3.3.3). For this purpose, *mosquitto*[6] was used, since it is widely known and proven. The following changes were made to the default configuration:

- **Access authentication** by username and password.
- A **websockets** listener was enabled on port 1884, to make simple web based clients possible, i.e. JavaScript clients (see Section 3.3.5).
- Secure communication and client authentication through TLS *with client certificates*.

The configuration is shown in listing 5.

Listing 5: "Mosquitto configuration"

```
pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

cafile /etc/mosquitto/ca_certificates/ca.pem
certfile /etc/mosquitto/certs/broker.crt
keyfile /etc/mosquitto/certs/broker.key

require_certificate true

password_file /etc/mosquitto/passwd
allow_anonymous = false

port 1883

listener 1884
protocol websockets

cafile /etc/mosquitto/ca_certificates/ca.pem
certfile /etc/mosquitto/certs/broker.crt
keyfile /etc/mosquitto/certs/broker.key

require_certificate true
```

The data broker additionally runs a *hostapd* and *dnsmasq* to serve a WiFi AP to the gateway device and potential MQTT clients subscribing to the sensor data. A systemd script serves as

[6] https://www.mosquitto.org

clock source for the sensor network and writes the date and time (sources via ntp) to the MQTT topic:

```
IoT/Time
```

Known Issues

- **Historic Data** cannot be stored adequately with MQTT in general, and shouldn't be stored on the message broker. This responsibility is thus shifted to a/the MQTT client, which can then be queried for this data using a different protocol.

3.3.5 Dashboard

The dashboard is used to visualize the data from the sensors, being dispatched by the MQTT broker (cf. Section 3.3.4 and Section 3.3.3). For this purpose, a small website was developed, using the *paho-mqtt.js*[7] MQTT JavaScript client. This client connects to the broker via websockets, since it runs in the browser and raw tcp sockets are not available for websites/via JavaScript.

For the visualization of the sensor data, the Google Charts library[8] was used, as a simple way of displaying generic charts in webpages using JavaScript and HTML. Unfortunately, using the library requires Internet access, since content is dynamically loaded from the Google servers.

Known Issues

- The Google charts library relies on content loaded from the Google servers, which means that the Dashboard can only display charts if the browser running the dashboard has access to the Internet.
- Relying on the MQTT-broker as data source (cf. Section 3.3.4), the dashboard is restricted to "live data" and would need an additional source to query for historic data, e.g. a time-series database as additional MQTT client. Retained messages might allow to at least display the last known value.

3.4 Evaluation

The setup shown in Fig. 4 was used to *approximately*[9] measure the power consumption of the nodes during typical operation[10]. The circuit was powered with a dedicated power supply, set to 3.3V to avoid destroying the attached RF-module and the FTDI programmer.

Values between **0.74mA** and **1.5mA** were measured.

[7] https://www.eclipse.org/paho/clients/js/
[8] https://developers.google.com/chart/
[9] Since the power consumption of the used test-bed is rapidly fluctuating, the used setup will only approximately show the flowing current.
[10] 1 Emulated Sensor, wakeup interval = 60sec, no serial print, frequency: 8MHz, without led on atmega board, without led on RTC board

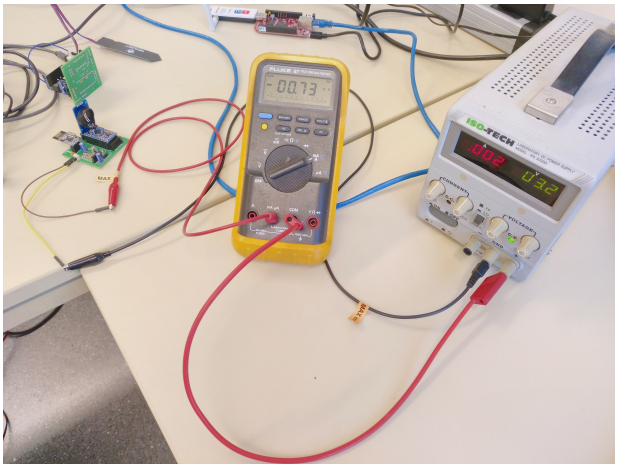

Figure 4: Power measurement setup

4 Discussion & Conclusion

The assignment could be completed according to the given specification while implementing a few minor additional components. The following concluding remarks might be made:

Underpowered Gateway The ESP8266 is slightly underpowered to act as a gateway, which became apparent during development. E.g. smaller key sizes than preferred had to be used for the TLS client certificates.

Established Solutions Established (open) solutions are available for all parts of the LP-RF section of the assignment, from devices (TI MSP430), over operating systems/environments (Contiki, Riot, TinyOS, Thread), to communication protocols (ZigBee, Thread, Sicslowpan, etc.), which offer considerably less power than the components used for the assignment. Similarly, the use of native IPv6 down to the sensor node (e.g. 6LoWPAN, ZigBee, Thread) would have been preferable.

Arduino While the Arduino ecosystem brought considerable simplifications and improvements to embedded systems development, the fragmented and frayed library ecosystem of voluntary contributions with varying degrees of capabilities results in unforeseeable bug hunting sessions, since the quality and active maintenance of libraries is not known beforehand. While great for hobbyists and proof of concept implementations, it is considered less suited for the production in the scope of commercial products or for more elaborate scientific work.

Acronyms

AP Access Point. 1, 2

HMAC keyed-Hash Message Authentication Code. 1, 3
IoT Internet of Things. 1, 2
MCU MicroController Unit. 1
RDC Radio Duty Cycling. 1
RF Radio Frequency. 1, 3, 4
SHA Secure Hash Algorithms. 1, 3
TLS Transport Layer Security. 1, 3

Glossary

MQTT MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport.. 1–6, a, b