HOMELAB PROJECT

IoT Gateway and Wireless Nodes

Part 1: The hardware

Sometimes you want to have specific functions in a home automation system that are not available with any commercial product. That's why the author decided to develop his own system, which features wireless communication between the various nodes and the central gateway. The gateway uses MQTT to send measurement data to an OpenHAB server, which processes and displays the data. His project has now been copied and extended by other users. In a series of two articles, the author describes the key elements of his system. More detailed information and the corresponding software are available for free on the Internet.

By **Hennie Spaninks** (Netherlands)

Home automation systems are enticing. There's something magical about having your lamps go on and off by themselves, a heating system that knows when you are at home, or receiving a message when someone is at your door. There are quite a few systems currently available, but most of them suffer from the following shortcomings:

- They often work in only one direction – you can send commands, but you never know whether they actually arrive. That's not a big issue when you're at home sitting next to a lamp and reading, but when you are on vacation in a foreign country it's nice to know that everything is working as it should.
- They often can only be operated using a specific app, and each vendor has their own app with protocols that do not work with other systems. As a result, you end up with a whole bunch of apps on your smartphone.
- The nice thing about home automation is that you can program them to send commands in response to sensor readings. However, because most commercially available systems are not compatible with each other, you have to use an external service (such as *IfThisThenThat*) to get a working system.
- Many wireless systems are based on Wi-Fi. In modern residential buildings made from reinforced concrete, the range of Wi-Fi signals is very limited.
- Security is not always optimal, so there is a distinct chance that your neighbors will be able to control your lamps.
- All in all, there are plenty of good reasons to roll up your sleeves and build your own home automation system.

Overview

For the end nodes, which means the sensors and actuators, you ideally want reasonably affordable devices which are also energy efficient. The central controller obviously has to support

standard network protocols, including TCP/IP. That is why the author opted for a two-tier system consisting of end nodes, a gateway and a central control unit. The overall system is shown schematically in **Figure 1**.

- For communication with the end nodes, the system uses wireless duplex links operating at a relatively low frequency. In Europe the available frequency bands for this purpose are 433 MHz and 868 MHz. The 433 MHz band is fairly crowded, so we opted for 868 MHz. Transceivers for this band which can be controlled over an SPI bus are available from HopeRF. We chose the version with the highest transmit power: the RFM69HW. This module has integrated hardware encryption, so security is not an issue.
- The RFM69 does not have a TCP/IP stack, so fixed-size data blocks of 66 bytes are exchanged over the radio link. A gateway is necessary to convert the data from the wireless network to TCP/IP and vice versa.
- To distribute the data from the sources (sensor nodes) to the recipients (for example, a smartphone), we opted for a different standard protocol which runs over TCP/IP: MQTT. It is a messaging protocol designed to send short messages to several recipients in a simple manner. Because MQTT is a standard protocol, data streams from our home automation system can also be used in other MQTT-based systems.
  MQTT requires a server to act as the message center, which is called the broker. A recipient (such as a smartphone) can subscribe to a data stream, after which it receives messages for that stream through a push service. To implement our MQTT broker we use Mosquitto [1], an open source MQTT broker implementation which is available for several platforms (including Raspberry Pi).
- Our home automation system should preferably be compatible with devices from other vendors. This means that it must also be able to communicate using other protocols. For this we opted for a system based on OpenHAB [2]. This open source software can convert messages in various protocols to a universal message stream. For example, you can use it to control Philips Hue lamps.
- OpenHAB has a standard app (for Android and iOS) which communicates with your own OpenHAB system over the Internet using a secure connection. That means you don't have to worry about configuring firewalls or encryption for the link between your smartphone and your home automation system. OpenHAB can be accessed on the local area network through a Web browser.
- OpenHAB has an underlying database which makes it easy to make charts of measured values. Creating rules and scripts to automate tasks is easy in OpenHAB. OpenHAB also has external interfaces for sending alerts via email or the OpenHAB app.
- We use readily available and relatively low-cost components: Arduino as the controller for the wireless nodes, and a Raspberry Pi as the platform for Mosquitto and OpenHAB.

Protocol The protocol for the end nodes

Various functions are defined for the end nodes. They include sensors for temperature and humidity, switches, PIR motion detectors for input, and relays and LC displays for output. End nodes can autonomously transmit measurement data at regular intervals. This measurement data can also be sent on request in response to read commands.

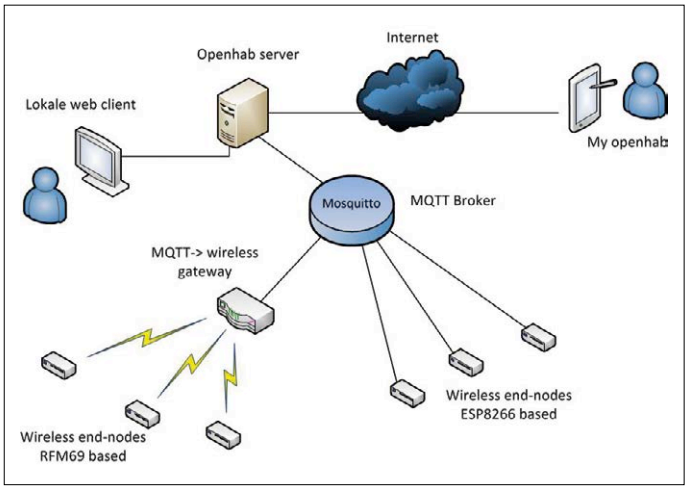

Figure 1. The architecture of the home automation system.

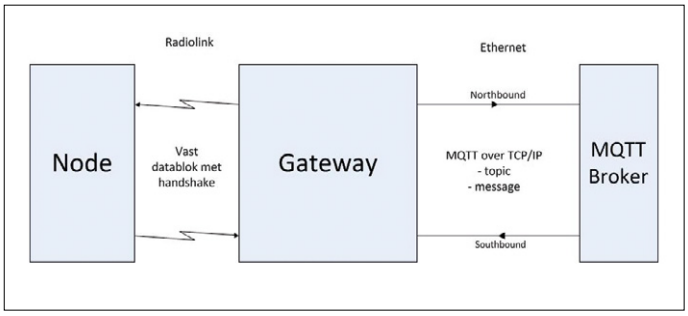

Figure 2. The various data streams passing through the gateway.

Table 1. Device identifiers.

DevID range	Function
0-15	System devices
16-31	Binary output (relay, lamp)
32-39	Integer output (dimmer, PWM)
40–47	Binary input (switch, PIR module)
48-63	Decimal input (temperature, humidity)
64-71	Integer input (keypad, switch)
72	String transfer (LCD display, RFID reader)
73-90	Reserved for future use
90-99	Error messages

The data flow through the gateway is shown in **Figure 2**. End nodes can be programmed for various functions. To ensure that the gateway handles communications with each end node correctly, there is a fixed allocation of functions to devices. Based on the device identifier (DevID), the gateway knows how to handle the incoming data. **Table 1** shows an overview of the DevIDs recognized by the gateway.
Devices 0 to 15 provide system functions. They are described in **Table 2**.
The error messages generated by the gateway are listed in **Table 3**.

Table 2. System functions.			
DevID	**Name**	**RW**	**Function**
00	Uptime	R	Minutes since node start
01	TxInterval	RW	Transmit interval in seconds (0 = no periodic transmission)
02	RSSI	R	Radio signal field strength
03	Version	R	Software version of the end node
04	Voltage	R	Battery voltage
05	ACK	RW	Flag for acknowledging sent commands
06	Toggle	RW	Flag for toggle function of switch on end node
07	Timer	RW	Flag for timer function of pushbutton on end node
08	Btnpress	RW	Flag for sending button press message
09	TXreply	R	Number of repeats necessary on the radio link

Table 3. Error messages.		
Error ID	**Name**	**Description**
90	Link error	The radio link is interrupted.
91	Syntax error	There is a syntax error in the MQTT message.
92	Invalid device	The addressed DevID is not present in the end node.
99	Wakeup	Message sent when the node starts up.

Table 4. MQTT examples.		
Topic	**Message**	**Description**
home/rfm_gw/sb/node02/dev16	ON	Switches on the LED at node 02.
home/rfm_gw/sb/node02/dev16	READ	Queries the status of the LED at node 02.
home/rfm_gw/sb/node03/dev01	300	Sets the transmit interval of node 03 to 5 minutes.
home/rfm_gw/sb/node03/dev01	0	Disables periodic transmission from node 03.
home/rfm_gw/sb/node18/dev48	READ	Reads the temperature from node 18.
home/rfm_gw/sb/node05/dev02	READ	Reads the radio signal field strength at node 05.
home/rfm_gw/sb/node05/dev03	READ	Reads the software version of node 05.

MQTT messages

The MQTT protocol works on the basis of subscribing to topics (see [3]). After a recipient has subscribed to a topic, the broker ensures that all messages related to that topic are sent to the recipient.

The format of the MQTT topic subscription is

`home/rfm_gw/direction/nodeID/deviceID`

Here `direction` specifies the direction of the data flow: “nb”

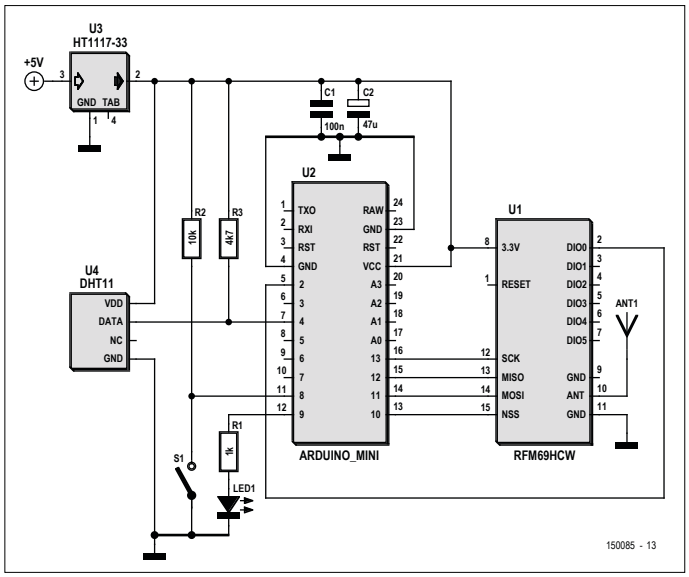

Figure 3. Schematic diagram of an end node.

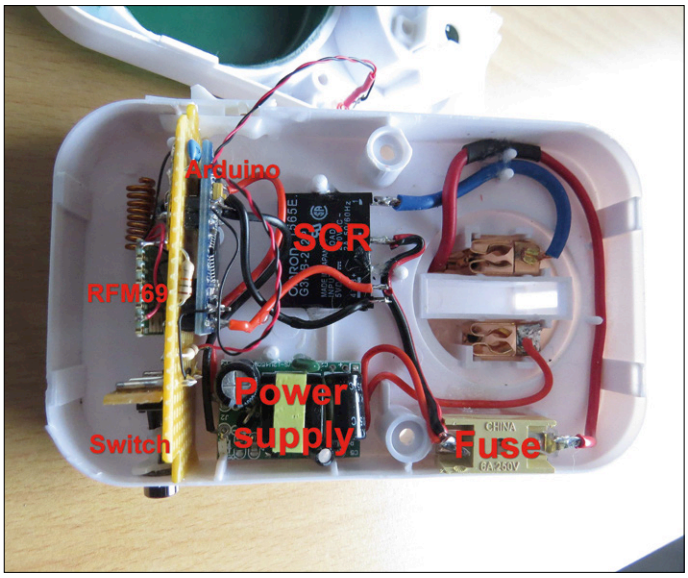

Figure 4. A DIG node in a plug-in timer housing.

(northbound) means from the node to OpenHAB (sensor data), while "sb" (southbound) means from OpenHAB to the node (commands).

- `nodeID` is the node identifier. Each node has a unique ID (address), which is determined when the code is compiled. The gateway always has node ID 01.
- `deviceID` is the function identifier, as previously described.

The MQTT message (payload) depends on the device and the direction.

- *Southbound*: The message contains commands for the node (ON, OFF, READ) or settings for the node, such as the number of seconds for the transmit interval.
- *Northbound*: The message contains the value of the parameter belonging to the DevID.

The gateway subscribes to all southbound messages for all nodes in the network by means of the following wildcard topic:

```
home/rfm_gw/sb/#
```

Of course, OpenHAB needs to receive all messages from the nodes, so it subscribes to the appropriate topics:

```
home/rfm_gw/nb/#
```

Table 4 shows a number of examples of MQTT messages.

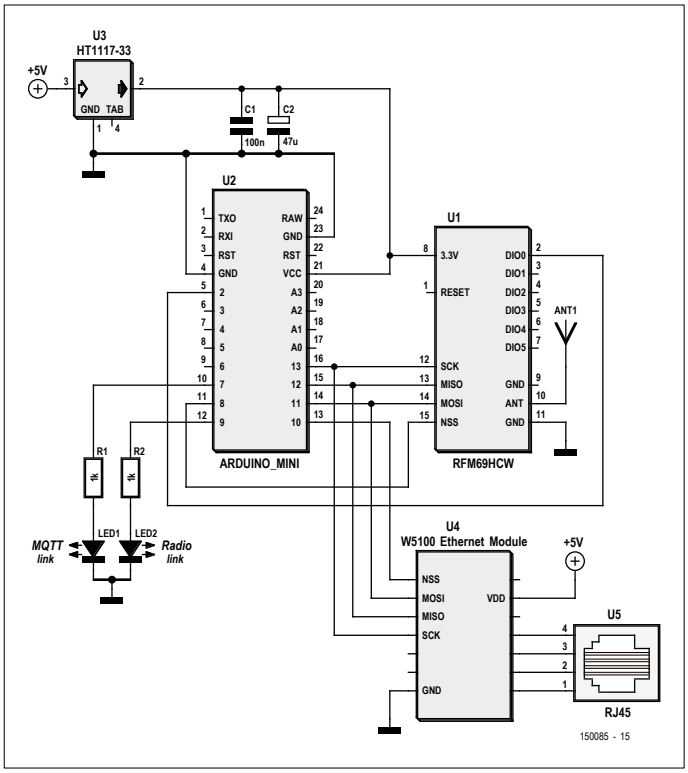

Figure 5. Schematic diagram of the gateway.

End nodes

The end nodes are based on Arduino boards. The following conditions and constraints are taken into account in the design:

- The RFM69 module has an operating voltage of 3.3 V, and the maximum permissible voltage on its inputs is 3.9 V. To keep things simple, we use an Arduino that also operates from 3.3 V. We opted for the Arduino Pro Mini, which operates from 3.3 V and can be connected to the RFM69 without level conversion. You can also use the Arduino Buono R3, which can be switched to 3.3 V.
- The RFM69 module briefly draws a significant amount current (130 mA) when transmitting, so the supply needs to have sufficient capacity.
- The Arduino and the RFM69 communicate with each other over the SPI bus. The standard Arduino pins are used for that purpose.
- The RFM69 may not be operated without an antenna. A piece of wire 8.6 cm long gives excellent results in most cases.

The schematic diagram of one of the end nodes is shown in **Figure 3**. This node is equipped with a DHT11 temperature and humidity sensor connected to pin 4 of the Arduino. A pushbutton is connected to digital I/O pin 8, and an LED is connected to pin 9.

The RFM69 module is connected to the Arduino over the SPI bus (SCK, MOSI, MISO and NSS). The 3.3 V supply voltage is provided by an AMS1117 voltage regulator, with enough decoupling capacitors to suppress current spikes.
The software loaded into the Arduino determines the function of the end node. The following end node devices have been developed so far:

- DHT is a node with sensors for temperature and humidity. This node has a digital output and a pushbutton.
- DIG is a simplified version of the DHT node. It only has a pushbutton and a digital output.
- RFID is a node with an RFID reader. When an RFID device is detected, the node sends the RFID identifier to OpenHAB.
- LCD is a node with a liquid crystal display. Text strings can be sent from OpenHAB to the display.
- RC is a node equipped with a 433 MHz transmitter module. It can be used to operate switches in the Dutch *KlikAanKlikUit* home automation system [4].
-

The software can easily be adapted to add or modify functions. The construction of a node is strongly dependent on its function. For example, the case of an inexpensive mechanical timer switch can be used to house a DIG node. An example of this is shown in **Figure 4**.

The gateway

The gateway consists of an Arduino, an RFM69 module and an W5100 Ethernet module. **Figure 5** shows the schematic diagram. The gateway also uses an Arduino operating at 3.3 V. On the gateway the SPI bus is shared by the RFM69 and W5100 modules. The SCK, MOSI and MISO signals are connected to both devices in parallel. Different Slave Select (SS) signals are used to select the individual devices on the bus: pin 8 for the RFM69 and pin 10 for the Ethernet module.
An AMS1117 also provides the 3.3 V supply voltage for the gateway. Two LEDs are provided to indicate the status of the gateway.
A separate module is used for the Ethernet connection. If you use an Arduino Buono board, you can use a Wiznet Ethernet

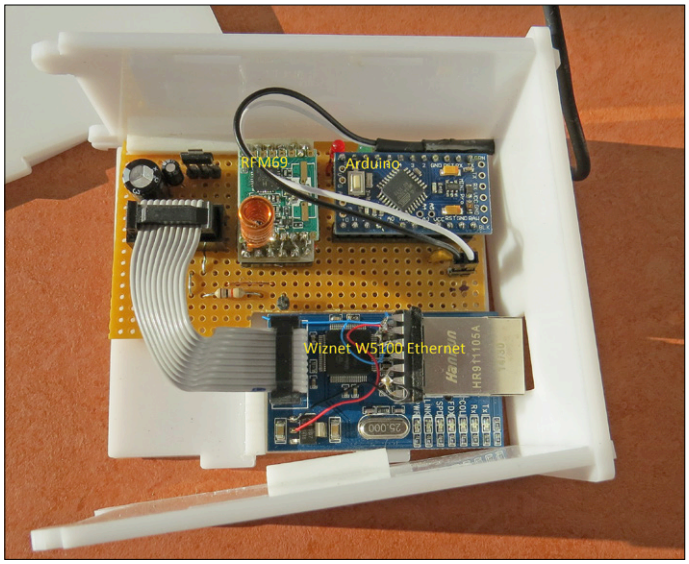

Figure 6. The author's fully assembled gateway.

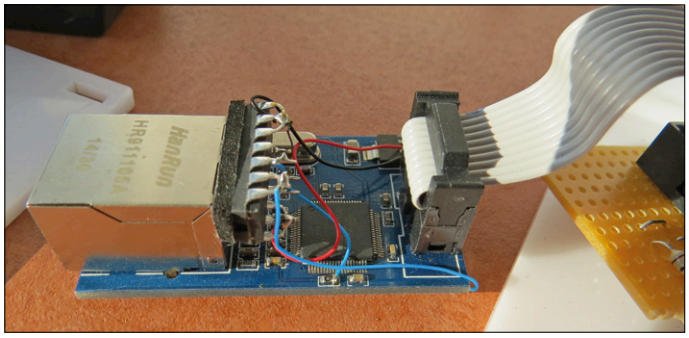

Figure 7. The 4011 is attached to the housing of the Ethernet connector and generates the SEN signal for the W5100 from the CS signal.

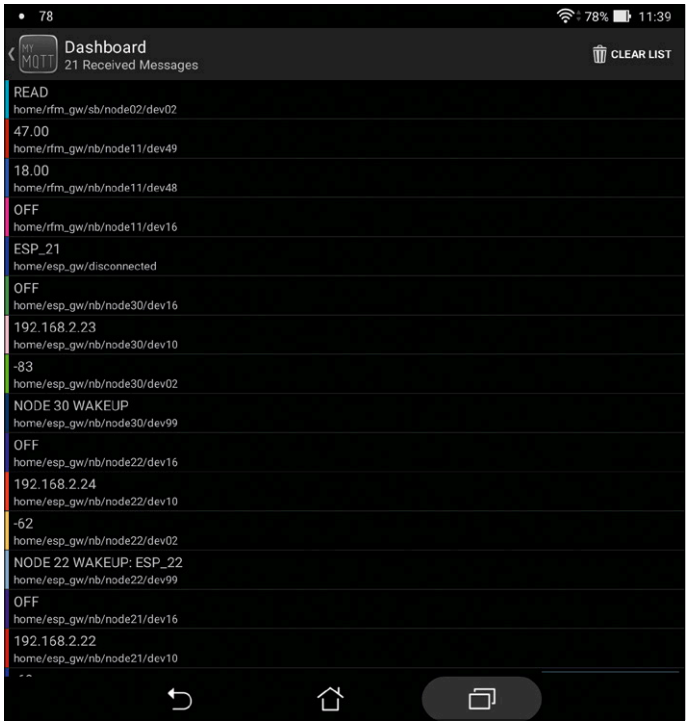

Figure 8. This screenshot shows some examples of MQTT messages.

shield instead. Only one gateway is needed for the entire system. For this reason, the author built the gateway on a prototyping board (**Figure 6**).

There are two things you have to pay attention to when building the gateway, as described below.

W5100 bug

The hardware of the W5100 is not directly suitable for sharing an SPI bus with other devices. When an SPI bus device is not using the bus, it is supposed to release the signal lines so that they can be used by other devices. The original W5100 is not designed to do this. The author solved this problem by using an inverter to generate the SEN control signal from the CS signal (see [5] for more details). The necessary inverter is integrated in newer versions of the W5100 module. The module which the author bought on eBay did not include this feature, so the inverter had to be added externally. As can be seen in **Figure 7**, a type 4011 IC was used for this. There you can also see that it is easy to connect the SEN signal line via a pull-up resistor on the circuit board. Remember to connect the unused inputs of the 4011 to ground.

Short-circuit via ICSP header

If you use a standard-format Arduino with the W5100 shield, the Arduino board is connected to the 5 V supply voltage of the Ethernet shield through the ICSP header. That pulls up the 3.3 V supply voltage line on the Arduino, which can result in damage to the RFM69. This can be avoided by cutting off the VCC pin of the ICSP header or bending it aside so that it does not make contact.

To be continued

In the following article we will describe the software for the gateway and the nodes, as well as the configuration of Mosquitto and OpenHAB. If you can't wait, you can already download the software at [6]. You must have Mosquitto available in order to test the gateway. It's also handy to have an MQTT client for testing. The MyMQTT app is a good choice for Android, and on a Windows system you can use Chrome Lens or MQTT. FX. **Figure 8** gives an impression of the messages you can expect to see.

This project has now been built by several other people. Extensions, issues and experience are discussed and shared on the forum site [7]. ⏮

(150085)

Web Links

[1] https://mosquitto.org
[2] www.openhab.org
[3] www.elektormagazine.com/tags/journey-into-the-cloud
[4] www.klikaanklikuit.nl (Dutch!)
[5] http://john.crouchley.com/blog/archives/662
[6] https://github.com/computourist/RFM69-MQTT-client
[7] http://homeautomation.proboards.com/board/2/openhab-rfm69-based-arduino