Hausautomatisierung auf Basis des MQTT-Protokolls

Volks-Schalter

Das MQTT-Protokoll hat sich im Internet of Things als Standard durchgesetzt. Sein leichtgewichtiger Ansatz unterstützt selbst die kostengünstigsten Mikrocontroller. Das eröffnet auch ganz neue Möglichkeiten bei der Heimautomatisierung. Dieser Praxis-Artikel stellt sie vor. *Rainer Poisel*

© weedezign, 123RF

Man stelle sich vor, dass sich der Solltemperaturwert der heimischen Heizung bereits vom Büro aus vorgeben lässt oder dass eine intelligente Nachttischlampe während der Urlaubszeit Anwesenheit vortäuscht. Oder dass sich auch komplexe Abläufe automatisieren lassen, an denen viele Aktoren mitwirken: Zuhause angekommen fahren die Jalousien automatisch herunter, die Stehlampe wird gedimmt, die Hi-Fi-Anlage spielt entspannende Musik, der Kamin zündet … Das alles ist mit Relais realisierbar, die sich via WLAN und IP-Protokoll ins Hausnetz integrieren. Wovon man noch vor Jahren nur träumen konnte, ist heute für relativ wenig Geld realisierbar.

Eine Möglichkeit, das häusliche Umfeld zu automatisieren, verwendet das Protokoll MQTT (Message Queue Telemetry Transport). Während sich ein früherer Magazin-Artikel mit MQTT unter dem Aspekt passender Programmierbibliotheken beschäftigte [1], erörtert dieser nun, wie der findige Bastler mit MQTT beispielhaft ein WLAN-taugliches Relais und einen ebensolchen Taster in eine größere Heimautomatisierungs-Lösung integriert.

Dieses Beispiel lässt sich selbstverständlich nach Gutdünken erweitern und ausbauen. Das sollte nach der Lektüre des vorliegenden Artikels auch nicht mehr so schwierig sein, zumal es eine große und sehr hilfsbereite Community gibt, die viele inspirierende Programmbeispiele parat hat.

Seitdem bekannt wurde, dass Facebook MQTT zum internen Datenaustausch einsetzt [2], entstand ein wahrer Hype um das Ende der 90er Jahre entwickelte Protokoll. Einer seiner großen Vorteile ist, dass viele freie Programmierbibliotheken für die verschiedensten Sprachen frei verfügbar sind. Eine Übersicht über das Protokoll liefert der Kasten "MQTT und Controller". Im vorliegenden Artikel dient MQTT als Basis für die Datenübertragung zum und vom Modul mit den Schaltelementen.

Alles funkt

Dank der Verfügbarkeit freier, unter Linux lauffähiger Entwicklungsumgebungen kann der Heimwerker so genannte Sonoff-Module in einer Linux-basierten Heimautomatisierungs-Lösung verbasteln. Wegen der dabei nutzbaren WLAN-Technologie ist selbst das Umrüsten oder Erweitern bestehender Installationen ohne Neuverkabelung möglich.

Die in diesem Artikel beschriebenen Sonoff-Module gibt es in mehr als zehn verschiedenen Ausführungen. Neben der Standardvariante mit einem Relais und WLAN-Schnittstelle stehen dem versierten Bastler Module mit mehr als einem Ausgang oder anderen Funktechnologien zu Verfügung. Es gibt auch noch andere Bauformen, zum Beispiel Lampensockel (Produktname: Slampher).

Abbildung 1 zeigt den Lieferumfang des Standard-Sonoff-Moduls. An die von außen sichtbaren Anschlüsse schließt der Heimautomatisierer die Zuleitungen an das Stromnetz sowie die Ableitung zum zu schaltenden Verbraucher an.

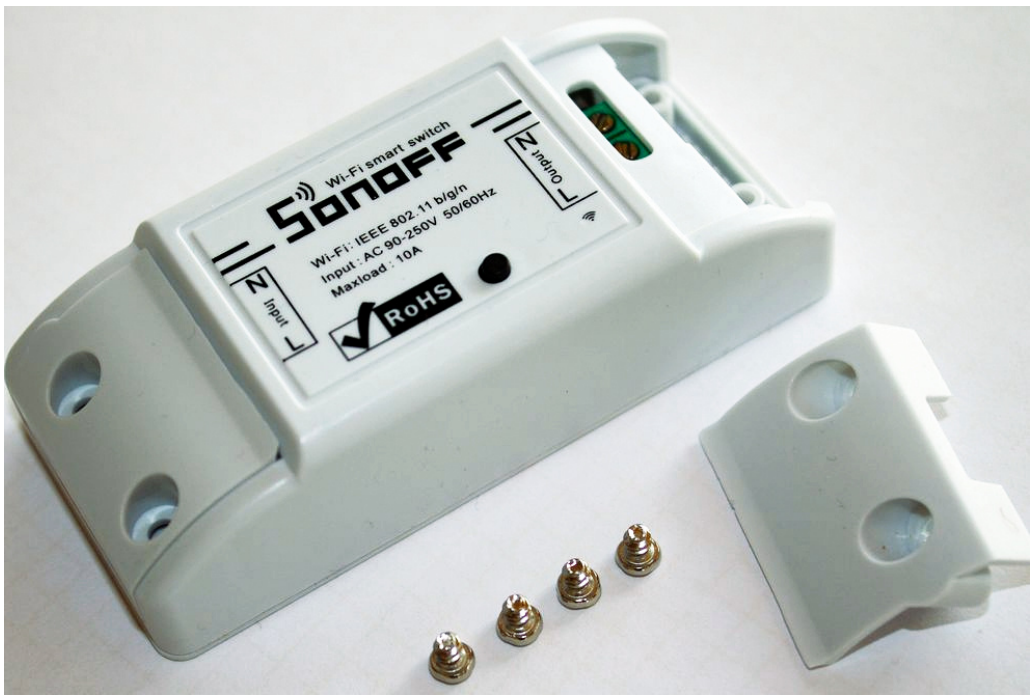

Abbildung 1: Lieferumfang eines Standard-Sonoff-Moduls.

Abbildung 2 lässt erkennen, was sich in dem Kunststoffgehäuse verbirgt. Das Innenleben ist durchaus geeignet, um ein MQTT-taugliches Relais herzustellen. Während der Neutral-Leiter direkt zum Verbraucher hin durchgeschleift ist, wird die Phase in Abhängigkeit von

der Relais-Stellung geschaltet. Die Stromversorgung des Mikrocontrollers erfolgt über ein eingebautes 3,3-Volt-Netzteil.

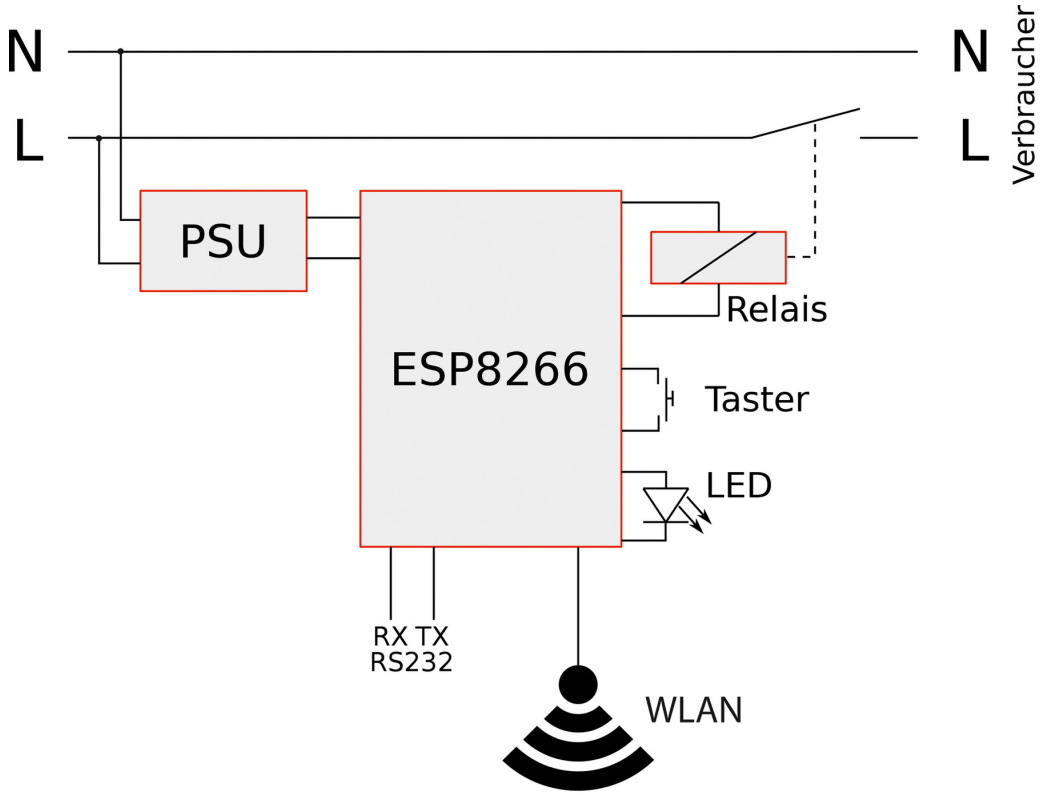

Abbildung 2: Blockschaltbild des Sonoff-Moduls.

Vom Mikrocontroller selbst haben die Entwickler der Sonoff-Module drei seiner General Purpose In-/Outputs (so genannte GPIOs) sowie die WLAN-Antenne und eine RS-232-Schnittstelle nach außen geführt. Die 8-MBit-Programmspeicher liegen außerhalb des Mikrocontrollers auf der Oberseite der Platine in Form eines IC mit SOP8-Formfaktor vor.

Um die RS-232-Schnittstelle zur Programmierung nutzen zu können, muss der Anwender das vorliegende Modul geringfügig adaptieren.

Hardware-Anpassungen

Die freie Programmierung der Sonoff-Module durch Endkunden war seitens des Herstellers – zumindest offiziell – nie vorgesehen. Der ambitionierte Bastler schlägt eine solche fehlende Einladung aber nicht aus, zumal der Aufwand für eine Anpassung gering ist: Er lötet in die für die serielle Schnittstelle vorgesehenen Löcher eine Stiftleiste mit fünf Pins ein. Abbildung 3 zeigt ein bereits modifiziertes Modul, auf dem jemand eine derartige Stiftleiste bereits angebracht hat. Die Bedeutung der Pins ist ebenfalls gekennzeichnet. Weiter unten geht der Artikel dann auf die Programmierung des Moduls ein.

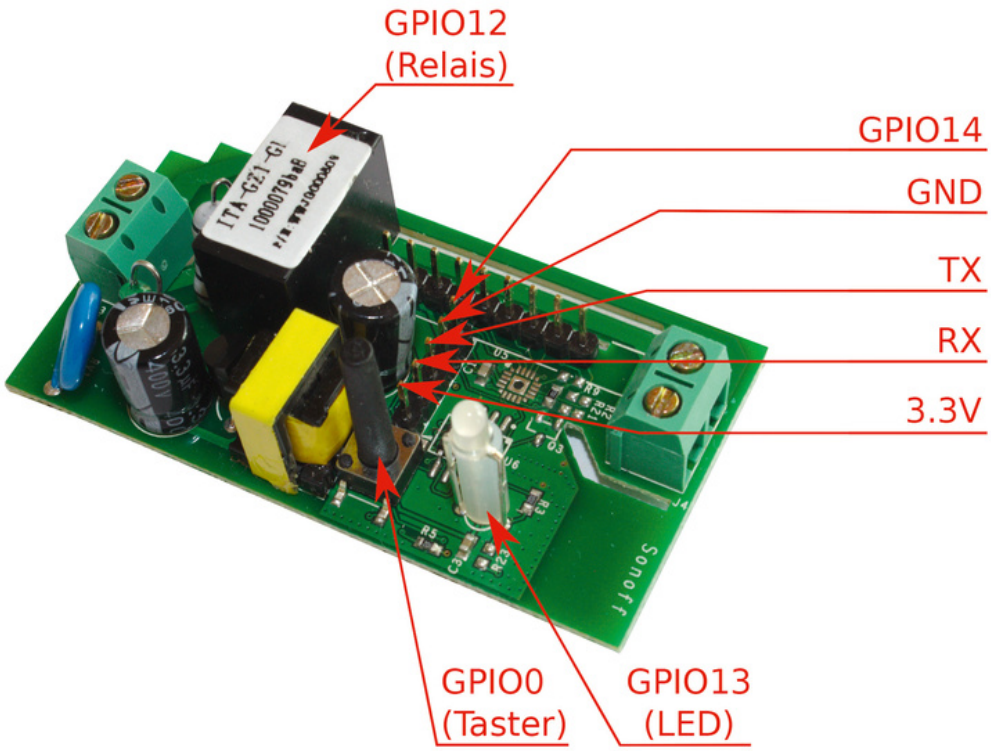

Abbildung 3: Ein um eine Stiftleiste erweitertes Sonoff-Modul.

Sonoff-Module lassen sich bei Bedarf noch weiter aufrüsten. So besteht die Möglichkeit, den Flashchip auszutauschen [5], um die Menge der ablegbaren, nicht-flüchtigen Daten auf 4 MByte zu erhöhen.

Will man sich die Lötarbeit ersparen, so gibt es auch Alternativen zum Sonoff-Modul, die nach demselben Funktionsprinzip arbeiten. Die Firma Olimex bietet beispielsweise ein Entwicklungsboard [6] an, das nach den hier skizzierten Schritten in Betrieb zu nehmen ist. Der Nachteil gegenüber dem Sonoff-Modul liegt darin, dass das Entwicklungsboard über keine eigene Stromversorgung verfügt. Die wäre daher gesondert zu organisieren und anzuschließen.

Prototyping für Mikrocontroller

Die Programmierung erfolgt im Artikel mit Hilfe der Arduino-IDE [7], die auch für Linux erhältlich ist. Die mit dieser IDE geschriebenen Programme verarbeitet üblicherweise ein GNU-C++-Compiler und Linker für die jeweilige Zielhardware. Mit wenigen Ausnahmen lassen sich daher fast alle C++-Sprachkonstrukte (Klassen, Vererbung, Templates und so weiter) einsetzen [8].

Im Auslieferungszustand kann die Arduino-IDE nur Mikrocontroller der ATMega-Reihe der Firma Atmel programmieren. Mittlerweile gibt es aber die notwendigen Erweiterungen, um auch ESP8266-Mikrocontroller ansprechen zu können, wie sie im Sonoff-Modul zu finden sind. Um eine ausreichend aktuelle Version der Arduino-IDE zu erhalten, bietet sich die Installation über die Pakete an, die auf der Homepage des Arduino-Projekts [7] zu finden sind.

Die Entwickler der ESP8266-Erweiterung für das Arduino-Framework haben die Installation [9] gut in die Arduino-IDE integriert. Über das Menü »File | Preferences« kann der Anwender zusätzliche URLs für so genannte Board Manager eintragen. Hinter diesem Begriff verbergen sich jene Funktionen, die das grundsätzliche Ansprechen der anvisierten Zielhardware ermöglichen.

Bei ESP8266-Mikrocontollern trägt man die URL »http://arduino.esp8266.com/stable/package_esp8266com_index.json« ein. Der eigentliche Installationsvorgang erfolgt nun über den Board Manager. Zu ihm führt das Menü »Tools | Board: | Boards Manager«. Nachdem der Anwender den Index der verfügbaren Basispakete ermittelt hat, filtert er die Liste der verfügbaren Board Support Packages nach der Zeichenkette »ESP8266« und installiert das gleichnamige Paket. Zum Zeitpunkt der Erstellung dieses Artikels war die Version 2.3.0 aktuell.

Mit Hilfe des Bibliotheksmanagers installiert der Anwender nun noch die Pub-Sub-Client-Bibliothek [10]. Diese Bibliothek erlaubt die Kommunikation über das MQTT-Protokoll. Die Installation gelingt mit dem Library Manager. Er ist über die Menüfolge »Sketch | Include Library | Manage Libraries« erreichbar. Die Eingabe von »PubSubClient« im Filter-Eingabefeld schränkt die Auswahl an Bibliotheken ein und mit einem Klick auf den »Install«-Button startet die Installation. Aktuell liegt die Bibliothek in der Version 2.6.0 vor.

Einspielen lassen sich die mit der Arduino-IDE entwickelten Programme über eingebaute RS-232-Bibliotheken oder, im Falle von Over-the-Air-Updates, über Hilfsskripte in Python. Daher ist für die weiteren Schritte das Vorhandenseins eines Python-Interpreters Voraussetzung. Nutzer Debian-basierter Distributionen erledigen die Installation mit dem Kommando: »apt-get install python«.

Die Bibliothek für Over-the-Air-Updates des Programmcodes, Arduino OTA [11], wird bereits mit dem Basispaket für ESP8266-Mikrocontroller mitgeliefert, man muss sie daher nicht gesondert installieren. Wer die Bibliothek einsetzt, kann Updates ohne Anschluss einer seriellen Schnittstelle direkt über WLANs vornehmen.

Blinken Lights

Das erste Programmierbeispiel (Listing 1) demonstriert den prinzipiellen Umgang mit digitalen Ein- und Ausgängen. Die Zustände der Ausgänge (hier sind das das Relais und die LED), schalten bei steigender Flanke des Eingangssignal (hier ist das der Taster) um. Die Setup-Funktion ist stets der Einsprungspunkt in einem Arduino-Programm. Üblicherweise initialisiert diese Funktion, die das System nur einmal beim Start durchläuft, die Variablen sowie die Peripherie der Zielhardware.

Listing 1: Blink-Programm

```
01 static int const buttonPin = 0;
02 static int const relayPin  = 12;  // active high
03 static int const ledPin    = 13;  // active low
04
05 static uint8_t prevButtonVal;
06 static uint8_t curOutVal;
07
08 static inline bool isRaisingEdge(uint8_t prevVal, uint8_t cur\
09   return prevVal == LOW && curVal == HIGH;
10 }
11
12 void setup() {
13   pinMode(buttonPin, INPUT);
14   pinMode(ledPin, OUTPUT);
15   pinMode(relayPin, OUTPUT);
16
17   prevButtonVal = digitalRead(buttonPin);
```

```
17   prevButtonVal = digitalRead(buttonPin);
18   digitalWrite(relayPin, LOW);
19   digitalWrite(ledPin, HIGH);
20   curOutVal = LOW;
21 }
22
23 void loop() {
24   uint8_t curButtonVal = digitalRead(buttonPin);
25   if (isRaisingEdge(prevButtonVal, curButtonVal)) {
26     curOutVal = curOutVal == HIGH ? LOW : HIGH;
27     digitalWrite(relayPin, curOutVal);
28     digitalWrite(ledPin, curOutVal == HIGH ? LOW : HIGH);
29   }
30   prevButtonVal = curButtonVal;
31 }
```

Daran schließt sich die Loop-Funktion an, die das System immer wieder durchläuft. Im Gegensatz zu Desktop-Programmen gibt es bei Mikrocontroller-Software häufig kein Ausstiegsszenario aus der Hauptschleife. Denn eine »return«-Anweisung in der »loop()«-Funktion würde den Mikrocontroller nur in einen Zustand versetzen, in dem er tatenlos verharrt, bis er von der Stromversorgung getrennt wird oder bis der Watchdog zuschlägt und ihn neu startet.

Die »PinMode()«-Funktion (Zeilen 13 bis 15) konfiguriert hier innerhalb der Setup-Funktion die relevanten Pins als Ein- beziehungsweise Ausgang. Wichtig ist in diesem Zusammenhang, das Relais aufgrund seiner Leistungsbeschaltung im Gegensatz zur LED über positive Logik anzusprechen. Daher befinden sich diese beiden Ausgänge nur scheinbar in gegensätzlichen Grundzuständen (Zeilen 17 und 18).

Um eine steigende Flanke des Tasters zu erkennen, muss das Programm die Änderung im Zeitverlauf betrachten. Dies bewerkstelligt es mit der »prevButtonVal«- und der »curButtonVal«-Variablen in der Loop-Funktion. Der Wert »false« der Variablen »prevButtonVal« bei gleichzeitigem Wert »true« der Variablen »curButtonVal« deutet auf das Vorhandensein einer steigenden Flanke des Eingangssignals hin. Um die Lesbarkeit des Codes zu erhöhen, setzt das Programm eine Hilfsfunktion »isRaisingEdge()« ein (Zeilen 8 bis 10 und 25).

Während der Programmierung des Sonoff-Moduls ist es wichtig, dass der Anwender den Netzanschluss auf jeden Fall trennt. Sonst besteht Lebensgefahr, falls er die Platine berührt! Die Stromversorung erfolgt während des Programmiervorgangs über das Programmierwerkzeug.

Beim ESP8266 übernimmt ein geeigneter USB-Seriell-Wandler den Flashvorgang. Er muss auf 3,3-Volt-Spannungsniveau arbeiten. Wer die Programmierung mit einem der ebenfalls gängigen 5-Volt-Wandler durchführt, riskiert Schäden am Mikrocontroller oder an der Peripherie des Sonoff-Moduls. Beim Verbinden des Wandlers mit dem Mikrocontroller ist darauf zu achten, dass der RX-Pin des Wandlers mit dem TX-Pin des Mikrocontrollers verbunden wird. Analog ist der TX-Pin des Wandlers mit dem RX-Pin des Mikrocontrollers zu verbinden.

ESP8266-Mikrocontroller sind vor der Übertragung des Programms in den so genannten Download-Mode zu versetzen. Um dies zu erreichen, schaltet der Programmierer den GPIO0-Pin beim Anlegen der Versorgungsspannung gegen Masse. Bei Sonoff-Modulen erreicht er dies einfach dadurch, dass er den Taster betätigt, während er die

Versorgungsspannung anschließt. Daraus ergibt sich folgende Anschlussreihenfolge: Zuerst schließt man die Masse, dann die Übertragungsleitungen (RX, TX) und zuletzt die Versorgungsspannung an.

Für eine erfolgreiche Inbetriebnahme ist es noch erforderlich, die Eckdaten des Sonoff-Moduls einzustellen (Tabelle 1). Wurden die richtigen Einstellungen vorgenommen und der Mikrocontroller in den Download-Mode gebracht, erfolgt der Flashvorgang über die Menüfolge »Sketch | Upload«. Üblicherweise dauert dies zwischen 30 Sekunden und einer Minute.

Tabelle 1: Eckdaten des Sonoff-Moduls

Komponente	Wert
Board:	Generic ESP8266 Module
Flash Mode:	QIO
Flash Frequency:	40 MHz
CPU Frequency:	80 MHz
Flash Size:	1M (64K SPIFFS)
Debug Port:	Disabled
Debug Level:	None
Reset Method:	ck
Upload Speed:	115200

Die Doppelbedeutung (Moduswahl beim Booten und Eingangssignal im Betrieb) des GPIO0-Pin ist aus Sicht der IT-Security nicht ganz unproblematisch. Wenn der Taster herausgeführt ist, könnten Angreifer das Gerät nämlich durch Trennen und Wiederverbinden der Stromversorgung bei gehaltenem Taster in den Download-Modus bringen. Danach könnten sie problemlos ihre Schadsoftware einspielen und das System damit kompromittieren. Alternativ zum Taster an GPIO0 bietet es sich daher an, den an der Stiftleiste herausgeführten GPIO14 als Eingangs-Pin zu verwenden. Er hat im Gegensatz zum GPIO0-Pin während des Bootvorgangs keine besondere Bedeutung.

Die folgenden Abschnitte stellen die drei Ein-/Ausgabegeräte schrittweise per MQTT zur Verfügung. Zuvor zeigen die nächsten Schritte aber, wie sich das Einspielen von Updates noch wesentlich vereinfachen lässt. Der Lieferumfang der Arduino-IDE und der meisten Erweiterungsbibliotheken umfasst gut dokumentierten Beispielcode. So gibt es auch ein minimales Arduino-Projekt, das das Einspielen von Software-Updates über das WLAN erlaubt.

Luft-Aktualisierung

Zum Beispielcode führt die Menüfolge »File | Examples | ArduinoOTA | BasicOTA«. Hat der Programmierer das Skript in den Zeilen 6 (WLAN-Kennung; Variable »ssid«) und 7 (WLAN-Passwort; Variable »password«) sowie in Zeile 24 (OTA-Kennung; Parameter der Methode »ArduinoOTA.setHostname«) angepasst, dann lädt er anschließend das Programm per USB-Seriell-Wandler auf den Mikrocontroller.

Nach einem Neustart der Arduino-IDE ist fortan unter dem Menüpunkt »Tools | Port« im Bereich »Network ports« das Sonoff-Modul als »Generic ESP8266 Module« zu finden. Das Einspielen neuer Software erfolgt von nun an über die WLAN-Schnittstelle. Das umständliche Anstecken des USB-Seriell-Wandlers ist dann nicht mehr erforderlich. Over-the-Air (OTA) aktualisierbare Geräte lassen sich über Multicast DNS [12] bekanntmachen.

Ein Echo-Server-Dienst ist im Kern des ESP8266-Arduino-Framework leider nicht vorgesehen, was jegliche Ping-Versuche ins Leere gehen lässt und die Fehlersuche nicht gerade vereinfacht.

Die Zahl der verfügbaren MQTT-Broker ist mittlerweile beachtlich [13]. Im Folgenden wird einer der prominentesten Vertreter – Mosquitto [14] – installiert und konfiguriert, um als zentraler Angelpunkt zu dienen.

MQTT für alle

Aufgrund seiner Popularität ist Mosquitto in den Paketarchiven der allermeisten Linux-Distributionen in einer aktuellen Version enthalten. Hier ist mindestens die Version 1.4 Voraussetzung, da in ihr der Transport von MQTT-Nachrichten über das Websockets-Protokoll bereits implementiert ist.

Nutzer Debian-basierter Distributionen führen zur Installation das Kommando »apt-get install mosquitto« aus. Die ganze Konfiguration passiert dann im Konfigurationsverzeichnis »/etc/mosquitto«. Glücklicherweise ist die Konfiguration auf Debian-basierten Distributionen modularisiert, sodass Root-User zusätzliche Konfigurationsanweisungen in separaten Dateien im »/etc/mosquitto/conf.d«-Verzeichnis ablegen können.

Die Datei »default.conf« enthält danach den Eintrag:

```
listener 1883
```

Die Datei »websockets.conf« soll aus diesen beiden Zeilen bestehen:

```
listener 1884
protocol websockets
```

Während »default.conf« die Konfigurationsanweisung für das MQTT-Protokoll über TCP beinhaltet, sind in »websockets.conf« die Direktiven für das MQTT-Protokoll über Websockets angeführt.

Nach erfolgter Konfiguration startet der Systemverantwortliche den Broker neu, um die Einstellungen zu übernehmen: »systemctl restart mosquitto«. Das »netstat«-Kommando hilft bei der Überprüfung der Konfiguration.

In der Ausgabe von »netstat -tanp« sollte erkennbar sein, dass auf den Ports 1883 und 1884 der Mosquitto-Prozess auf Verbindungen lauscht.

Analysearbeit

Während ein vorangehender Artikel über das MQTT-Protokoll [1] die Nutzung der Kommandozeilen-Werkzeuge der C/C++-Variante des Eclipse-Paho-Projekts [15] erläuterte, geht dieser Artikel auf die Verwendung des grafischen Analysewerkzeugs »mqtt-spy« [16] ein. »mqtt-spy« steht mittlerweile, wie auch das Paho-Framework, unter der Schirmherrschaft des Eclipse-Projekts.

Die Installation erfolgt im einfachsten Fall über die Jar-Version der in Java implementierten Software. Diese ist beziehbar über die Downloadseite des Projekts [17]. Nach erfolgtem Download startet der Bastler die grafische Oberfläche aus dem Downloadverzeichnis mit dem Kommando:

```
java -jar mqtt-spy-*-jar-with-dependencies.jar
```

Die Menüfolge »Connections | New connection« erlaubt das Einrichten einer Verbindung zum zuvor konfigurierten MQTT-Broker.

Die wichtigste Einstellung der sich öffnenden Konfigurationsmaske ist der Server-URI. Die Eingabe erfolgt hier im Format *Server-Name:* oder *IP-Adresse:Port*. Passend zur oben vorgenommenen Konfiguration trägt man hier die IP-Adresse oder den DNS-Namen samt »1883« für die Portnummer, getrennt durch einen Doppelpunkt, ein. Nach einem Klick auf den »Open Connection«-Knopf öffnet sich die Verbindung (Abbildung 4).

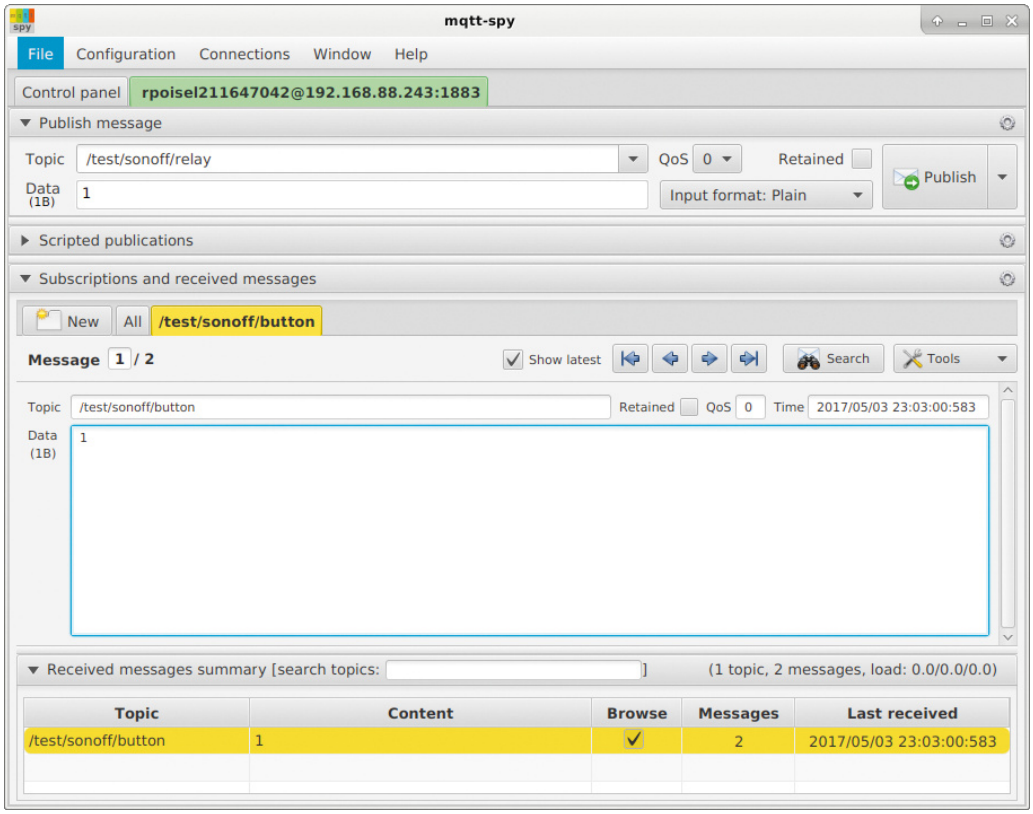

Abbildung 4: Die grafische Oberfläche von »mqtt-spy«.

MQTT-Nachrichten lassen sich flexibler erstellen, wenn bei der Dateneingabe das Eingabeformat auswählbar ist. Dies erreicht ganz einfach, wer über die Menüfolge »Window | Change Perspective | Detailed« die entsprechende Ansicht öffnet. Die restliche Oberfläche ist dann selbsterklärend gestaltet.

Im oberen Bereich besteht die Möglichkeit, MQTT-Nachrichten zu publizieren. Der mittlere Teil verwaltet die Subscriptions, und der untere Bereich zeigt schließlich eine Übersicht über bislang erhaltene Nachrichten an. Dabei sollte der Anwender im Hinterkopf behalten, dass über den Broker publizierte Nachrichten für ihn erst dann zu empfangen sind, wenn er auch das entsprechende Thema abonniert hat.

Dass der Bastler über dieses komfortable Werkzeug beliebige MQTT-Nachrichten sowohl publizieren als auch empfangen kann, ist sehr hilfreich, wenn er überprüfen will, ob die umprogrammierten Sonoff-Module auch tatsächlich so funktionieren, wie sie sollen.

Sonoff goes MQTT

Der Weg zum MQTT-fähigen Sonoff-Modul ist bereits fast geglückt. Nun kommt nur noch die bislang noch nicht verwendete Pub-Sub-Client-Bibliothek hinzu. Während das Drücken und Loslassen des Tasters am Sonoff-Modul MQTT-Nachrichten publiziert, werden sowohl das Relais als auch die LED beim Nachrichtenempfang ein- beziehungsweise ausgeschaltet. Listing 2 zeigt den Quelltext der MQTT-Anbindung für Sonoff-Module. Auch das Update der ESP8266-Firmware über die WLAN-Schnittstelle ist implementiert.

Listing 2: Ein-/Ausgänge per MQTT schalten (Fortsetzung)

```
001 #include <ESP8266WiFiMulti.h>
002 #include <ArduinoOTA.h>
003 #include <PubSubClient.h>
004
005 namespace {
006
007 int const buttonPin              = 0;
008 int const relayPin               = 12;
009 int const ledPin                 = 13;
010
011 char const* mqttTopicSubscribe   = "/test/sonoff/relay";
012 char const* mqttTopicPublish     = "/test/sonoff/button";
013 char const* mqttClientId         = "client-linuxmagazin";
014 char const* mqttBroker           = "rasppi3"
015 int const mqttPort               = 1883;
016
017 uint16_t const otaPort           = 8266;
018 char const* otaHostname          = "linuxmagazin";
019
020 uint8_t prevButtonVal;
021 ESP8266WiFiMulti wifiMulti;
022 WiFiClient wificlient;
023 PubSubClient client(wificlient);
024
025 static inline void blinkNum(size_t num, uint8_t pin) {
026   for (size_t cnt = 0; cnt < num; cnt++) {
027     digitalWrite(pin, LOW);
028     delay(100);
029     digitalWrite(pin, HIGH);
030     delay(100);
031   }
032 }
033
034 static void mqttCallback(char* topic,  byte* payload, unsigne
035   Serial.print("MQTT message: '");
036   for (unsigned int cnt = 0; cnt < length; cnt++) {
037     Serial.print((char)payload[cnt]);
038   }
039   Serial.print("' on topic '");
040   Serial.print(topic);
041   Serial.println("'");
042
043   if (length == 0) {
```

```
044     return;
045   }
046
047   switch (payload[0]) {
048     case '1':
049       digitalWrite(ledPin, LOW);      // LED is active-low, s
050       digitalWrite(relayPin, HIGH);
051       break;
052     case '0':
053       digitalWrite(ledPin, HIGH);     // LED is active-low, s
054       digitalWrite(relayPin, LOW);
055       break;
056     default:
057       Serial.print("MQTT: Unknown value");
058       break;
059   }
060 }
061
062 static void mqttReconnect() {
063   while (!client.connected()) {
064     Serial.print("Attempting MQTT connection...");
065     if (client.connect(mqttClientId)) {
066       Serial.println("connected");
067       blinkNum(2, ledPin);
068       client.subscribe(mqttTopicSubscribe);
069     } else {
070       Serial.print("failed, rc=");
071       Serial.print(client.state());
072       Serial.println(" try again in 5 seconds");
073       delay(5000);
074     }
075   }
076 }
077
078 }
079
080 void setup() {
081   Serial.begin(115200);
082   Serial.println("Booting");
083
084   Serial.println("Initializing inputs and outputs.");
085   pinMode(ledPin, OUTPUT);
086   pinMode(relayPin, OUTPUT);
087   pinMode(buttonPin, INPUT);
088
089   digitalWrite(ledPin, HIGH);
090   digitalWrite(relayPin, LOW);
091   prevButtonVal = digitalRead(buttonPin);
092
093   Serial.println("Initializing WiFi credentials.");
094   wifiMulti.addAP("SSID", "Password"); // bitte anpassen
095
096   Serial.print("Proceeding. Initializing ArduinoOTA. ");
097   ArduinoOTA.setPort(otaPort);
098   ArduinoOTA.setHostname(otaHostname);
099   ArduinoOTA.onStart([]() {
100     Serial.println("OTA Update Start");
101   });
```

```
102   ArduinoOTA.onEnd([]() {
103     Serial.println("\n OTA UPdate End");
104   });
105   ArduinoOTA.onProgress([](unsigned int progress, unsigned ir
106     Serial.printf("Progress: %u%%\r", (progress / (total / 1(
107   });
108   ArduinoOTA.onError([](ota_error_t error) {
109     Serial.printf("Error[%u]: ", error);
110     if      (error == OTA_AUTH_ERROR   ) Serial.println("Auth
111     else if (error == OTA_BEGIN_ERROR  ) Serial.println("Begi
112     else if (error == OTA_CONNECT_ERROR) Serial.println("Conr
113     else if (error == OTA_RECEIVE_ERROR) Serial.println("Rece
114     else if (error == OTA_END_ERROR    ) Serial.println("End
115   });
116
117   client.setServer(mqttBroker, mqttPort);
118   client.setCallback(mqttCallback);
119 }
120
121 void loop() {
122   static auto connectionConfigured = false;
123   auto curButtonVal = digitalRead(buttonPin);
124
125   if (wifiMulti.run() == WL_CONNECTED) {
126     if (!connectionConfigured) {
127       ArduinoOTA.begin();
128       Serial.println("WiFi Ready");
129       Serial.print("IP address: ");
130       Serial.println(WiFi.localIP());
131       connectionConfigured = true;
132     }
133     if (client.connected()) {
134       if (prevButtonVal != curButtonVal) {
135         client.publish(mqttTopicPublish, curButtonVal == HIGH
136       }
137       client.loop();
138     } else {
139       mqttReconnect();
140     }
141   }
142   else {
143     connectionConfigured = false;
144   }
145
146   ArduinoOTA.handle();
147   prevButtonVal = curButtonVal;
148 }
```

Zu Beginn definiert der Programmierer die Programmkonstanten: In den Zeilen 5 bis 7 legt er fest, an welchen GPIOs die Peripherie des Sonoff-Moduls angeschlossen ist. Die Zeilen 9 bis 15 definieren MQTT-spezifische Parameter. Programmrelevante Variablen sammeln die Zeilen 17 bis 20 im Daten- beziehungsweise BSS-Segment. Die Zugangsdaten für die WLAN-Verbindung enthalten die Zeilen 89 bis 92.

Die verbleibende Logik der Setup-Funktion initialisiert – entsprechend den zuvor definierten Konstanten – die serielle Schnittstelle (Zeile 79), Ein- und Ausgänge (Zeilen 80

bis 86), die Aktualisierung über die WLAN-Schnittstelle (Zeilen 95 bis 113) sowie die MQTT-Anbindung (Zeile 118).

Eingehende MQTT-Nachrichten arbeitet das Modul ereignisbasiert ab. Aus diesem Grund registriert der Programmierer beim stellvertretenden Client-Objekt einen Callback (Zeilen 119 und 34 bis 60), der bei Empfang von MQTT-Nachrichten aufgerufen wird. Die Logik des Callbacks ist überschaubar gehalten: Die wesentlichen Bestandteile einer MQTT-Nachricht (Topic, Payload und Länge) erhält der Callback durch die Pub-Sub-Client-Bibliothek als Parameter übergeben. Der Rumpf der Funktion muss sie auswerten. In diesem Beispiel hat sich der Programmierer entschlossen, das Relay und die LED beim Empfang einer »1« ein- und beim Empfang einer »0« auszuschalten. Alle anderen Werten führen zur Ausgabe einer Fehlermeldung über die serielle Schnittstelle.

Die zyklisch abzuarbeitende Loop-Funktion ruft im Wesentlichen die Event-Loops der Subsysteme auf (WLAN-Verbindung in Zeile 126, MQTT-Verbindung in Zeile 138 und OTA-Updates in Zeile 147). Im Falle der WLAN-Verbindung wird anhand des Rückgabewerts der Event-Loop-Funktion zwischen verbundenem Zustand (Zeilen 127 bis 141) und nicht-verbundenem Zustand (Zeile 144) unterschieden.

Der erste Durchlauf im Zustand "Verbinden" initialisiert die OTA-Datenstrukturen. Unabhängig davon wird gegebenenfalls die MQTT-Verbindung neu aufgebaut (Zeilen 134 und 140) oder der aktuelle Zustand des Tasters als MQTT-Nachricht publiziert, sollte sich dieser seit dem letzten Durchlauf verändert haben (Zeilen 135 bis 137).

Den Verbindungsaufbau hat der Programmierer in die Funktion »mqttReconnect()« gekapselt. Das Innere einer Schleife (Zeilen 63 bis 75) versucht die Verbindung mit dem MQTT-Broker herzustellen. Gelingt dies nicht auf Anhieb, wartet das Programm zwischen den Versuchen, die Verbindung aufzubauen, 5 Sekunden (Zeilen 70 bis 73). Im Erfolgsfall blinkt die eingebaute LED mit Hilfe einer Hilfsfunktion (»blinkNum()«, Zeilen 25 bis 32) zweimal (Zeile 67).

Einmal auf das Modul übertragen, steht der Integration des nun auch Firmware-seitig adaptierten Sonoff-Moduls nichts mehr im Wege.

Die Nachrichtenquelle

Erste Tests lassen sich nun mit »mqtt-spy« durchführen. Die Publikation von »0« beziehungsweise »1« mit dem Topic »/test/sonoff/relay« sollte die LED und das Relais zum Ein- beziehungsweise Ausschalten bewegen. Ebenso sollte das Drücken des Tasters am Sonoff-Modul dazu führen, dass das Modul die entsprechenden Nachrichten mit dem Topic »/test/sonoff/button« publiziert.

Eine Weboberfläche

Tatsächlich will man als Nutzer des Systems jedoch nicht unbedingt ein Analysewerkzeug zum Betrieb einsetzen. Daher baut der Programmierer im nächsten Schritt mit Hilfe der Javascript-Variante der Eclipse-Paho-Bibliothek in Listing 3 eine Weboberfläche für diesen Zweck zusammen. Auf den Einsatz eines Web Application Framework verzichtet das Beispiel bewusst, damit der Fokus auf die Nutzung der MQTT-Anbindung gerichtet bleiben kann.

Listing 3: MQTT-Nachrichten per Javascript

```
01 <html>
02     <head>
03         <link rel="stylesheet" href="sonoff.css">
04         <script
05             src="https://cdnjs.cloudflare.com/ajax/libs/paho-m
06             type="text/javascript">
07         </script>
08         <script>
09             client = new Paho.MQTT.Client(window.location.host
10             client.onConnectionLost = function (responseObject
11                 if (responseObject.errorCode !== 0) {
12                     alert("onConnectionLost:" + responseObject
13                 }
14             };
15
16             client.onMessageArrived = function (message) {
17                 document.getElementById("button").innerHTML =
18                     message.payloadString == "1" ? "pressed" :
19             };
20
21             client.connect({onSuccess: function() {
22                 client.subscribe("/test/sonoff/button");
23             }});
24
25             function changeState(newState) {
26                 message = new Paho.MQTT.Message(newState ? "1"
27                 message.destinationName = "/test/sonoff/relay"
28                 client.send(message);
29             }
30         </script>
31     </head>
32     <body>
33         <div style="display: flex; align-items: center;">
34             <label class="switch" style="float: left;">
35                 <input type="checkbox" onclick="changeState(th
36                 <div class="slider round" style="float: left;"
37             </label>
38             <div id="button"></div>
39         </div>
40     </body>
41 </html>
```

Das Relais des Sonoff-Moduls wird als Checkbox implementiert, die das Programm als Slider darstellt. Die erforderlichen Stylesheets lädt die Zeile 3. Beim Betätigen des Tasters am Modul zeigt die Oberfläche einen selbsterklärenden Text (»pressed«) neben dem Slider an.

Die Seite lässt sich zu Testzwecken direkt vom integrierten Webserver in Python ausliefern. Dazu ruft der Bediener eine Shell auf und wechselt dann in das Verzeichnis, das die CSS- und HTML-Dateien enthält. Dort gibt er – je nach verwendeter Python-Version – eines der folgenden beiden Kommandos ein:

```
python -m SimpleHTTPServer 80
```

oder

```
python3 -m http.server 80
```

Das Skript in Listing 3 ist so gestaltet, dass die in den Zeilen 4 bis 7 geladene MQTT-Bibliothek eine Verbindung zu jenem MQTT-Broker aufbaut, der auf dem selben Host läuft wie der diese Webseite liefernde Webserver (Zeile 9).

Die Interaktion mit der Javascript MQTT-Bibliothek erfolgt asynchron über Callbacks. Ein Verbindungsverlust wird über eine Alert-Box dargestellt (Zeilen 10 bis 14). Beim Empfang einer »1« wird der Text »pressed« in das zugehörige »div«-Element gesetzt (Zeilen 16 bis 19 und 39). Dazu abonniert das MQTT-Client-Objekt bei erfolgreichem Verbindungsaufbau das Theme »/test/sonoff/button« (Zeilen 21 bis 23).

Den Slider haben die Zeilen 35 bis 38 in das HTML-Dokument eingebettet. Dabei löst ein Klick auf den Slider den Aufruf der Funktion »changeStatus()« aus. Das Funktionsargument ist in Abhängigkeit von der Aktivierung der dem Slider zugrunde liegenden Checkbox »true« oder »false«. Bei Aufruf der »changeState()«-Funktion erfolgt die Publikation einer »1« beziehungsweise einer »0« an das »/test/sonoff/relay« (Zeilen 25 bis 29).

Fazit

Anstelle des selbst eingerichteten MQTT-Brokers kann auch ein öffentlicher MQTT-Broker, etwa von Cloud MQTT [18], zum Einsatz kommen. Diese Broker sind über das Internet erreichbar und haben den Vorteil, dass sie die in diesem Artikel vorgestellten MQTT-Relais von jedem Internetanschluss aus kontaktieren können, ohne dabei eine direkte Verbindung zu haben.

Dieser Artikel demonstriert, wie einfach mittlerweile auch die Funk-basierten Heimautomatisierungs-Lösungen implementierbar sind, die auf standardisierten Protokollen aufbauen. Ziel des Artikels war die Anbindung von Tastern und Relais per MQTT über WLANs.

Die Einbindung weiterer Sensoren, etwa die von Temperaturfühlern oder anderen Ausgangsmodulen, zum Beispiel von LED-Controllern [19], ist einfach zu bewerkstelligen. Auch die Integration in das Open-HAB-Framework hat schon jemand aus der Community erfolgreich durchgeführt [20].

Auch fertig implementierte alternative Firmware ist für Sonoff-Module in Form von Open Source verfügbar [21]. Diese braucht der Anwender in der Regel nur mehr an das vorliegende Einsatzszenario anzupassen.

Wer an dem Thema interessiert ist und sich näher mit der Programmierung der ESP-Mikrocontroller beschäftigen möchte, ist mit dieser Mikrocontroller-Serie als Basis ganz gut bedient. Neben digitalen Ein- und Ausgängen gibt es auch Programmbibliotheken zur Unterstützung diverser Bus-Protokolle sowie zur detaillierten Kontrolle über das WLAN-Interface.

So ist es etwa denkbar, ein Anemometer (Wind-Sensor) zu implementieren, das beim Überschreiten der zulässigen Windgeschwindigkeit das WLAN-Interface hochfährt und eine entsprechende MQTT-Nachricht publiziert. Die auf das zugehörige Nachrichtenthema abonnierten Jalousien fahren dann hoch, um Beschädigung zu vermeiden.

Zusätzliche Programmbibliotheken erlauben es, den Mikrocontroller in einen Schlafmodus zu versetzen, um auf diese Weise Energie zu sparen. Eine entsprechender Infrastruktur

vorausgesetzt, lässt sich dann sogar ein autarkes System installieren, das nur durch eine Solarzelle samt zugehörigem Pufferakku mit Strom versorgt wird.

An diesen Beispielen sieht man ganz gut: vieles wird dank günstiger IoT-Hardware samt vieler verfügbarer Programmbibliotheken und offener Standards selbst in Bestandsbauten ohne Neuverkabelung möglich. (jcb)

MQTT und Controller

MQTT ist ein Protokoll, das nach dem Publish/Subscribe-Mechanismus arbeitet. Die Kommunikationsteilnehmer tauschen dabei Nachrichten nicht direkt untereinander aus, sondern über einen so genannten MQTT-Broker. Der Broker ist dafür verantwortlich, die Nachrichten anhand ihres Themas (Topics) an interessierte Clients weiterzuleiten [3]. Beim Abonnieren eines Topic sind neben der exakten Angabe auch Wildcards (# und +) nutzbar. So lassen sich ganze Themengruppen auf einmal abonnieren.

Als Transportmedium für MQTT dient üblicherweise das TCP-Protokoll. Grundsätzlich ist der Transport aber über jedes Stream-basierte Medium möglich, beispielsweise auch über Websockets oder RS232. Die TCP-basierte Variante eignet sich insbesondere für den Einsatz über das Internet. Wer auf Sicherheit achten möchte, kann die Verbindungen auch per SSL verschlüsseln.

Heutzutage sind leistungsfähige Mikrocontroller-basierte Systeme, die den Einsatz von MQTT über WLANs ermöglichen, selbst in geringen Stückzahlen besonders günstig zu bekommen. Prominente Vertreter in diesem Bereich sind die so genannten Sonoff-Module [4], die man auf gängigen Internet-Marktplätzen für umgerechnet knapp 5 Euro pro Stück bereits inklusive Porto direkt aus China bestellen kann. Die Module kommunizieren über eine vom Hersteller bereitgestellte Cloud-Infrastruktur. Im vorliegenden Beispiel soll stattdessen aber ein selbst eingerichteter MQTT-Broker als zentraler Angelpunkt fungieren.

Der Hersteller der Sonoff-Module – I-Tead – verbaut in den Modulen ESP8266-Mikrocontroller. Die erfreuen sich besonders unter Bastlern großer Beliebtheit. Nicht alle Komponenten der Systemsoftware dieser Mikrocontroller stehen im Quelltext zur Verfügung. In der Praxis hat das aber kaum Auswirkungen auf die Lösbarkeit gängiger Programmieraufgaben. Die Programmierung erfolgt mit einer Variante des GCC-Compilers, den es für Linux, Windows und Mac OS gibt. (Mehr dazu weiter unten.) Für ESP8266-Mikrocontroller existieren auch Python- und LUA-Interpreter.

Mittlerweile ist bereits der Nachfolger des ESP8266 – der ESP32 – erhältlich. Der ESP32 verfügt neben der stärkeren Rechenleistung über zusätzliche Schnittstellen, etwa eine Bluetooth-Schnittstelle. Dieser Artikel geht nur auf die Nutzung des ESP8266 ein, da er für die vorgestellten Szenarien völlig ausreicht. .

Infos

1. Jan-Piet Mens, "Stille Post": Linux-Magazin 11/2015, S. 68, [http://www.linux-magazin.de/Ausgaben/2015/11/MQTT]

2. Pressemeldung über den Einsatz von MQTT bei Facebook: [https://www.ibm.com/developerworks/community/blogs/mobileblog/entry/why_facebook lang=en]
3. MQTT Essentials, Topics best practices: [http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices]
4. Sonoff-Modul: [https://www.itead.cc/sonoff-wifi-wireless-switch.html]
5. Scargill's Tech Blog: Sonoff Module mit 32 MBit Flash ausstatten: [http://tech.scargill.net/32mb-esp01/]
6. Olimex ESP8266-EVB: [https://www.olimex.com/Products/IoT/ESP8266-EVB/open-source-hardware]
7. Arduino Quick Start Guide für Linux: [https://www.arduino.cc/en/Guide/HomePage]
8. "Using modern C++ techniques with Arduino": [http://hackaday.com/2017/05/05/using-modern-c-techniques-with-arduino/]
9. Installationsanleitung für die ESP8266 Arduino-Erweiterung: [http://esp8266.github.io/Arduino/versions/2.0.0/doc/installing.html]
10. Pub-Sub-Client-Bibliothek für das Arduino-Framework: [https://github.com/knolleary/pubsubclient]
11. Arduino-OTA-Bibliothek für ESP8266-Mikrocontroller: [https://github.com/esp8266/Arduino/tree/master/libraries/ArduinoOTA]
12. Multicast DNS: [https://en.wikipedia.org/wiki/Multicast_DNS]
13. MQTT Broker Capabilities: [https://github.com/mqtt/mqtt.github.io/wiki/server-support]
14. Mosquitto: [https://mosquitto.org]
15. Eclipse Paho: [http://www.eclipse.org/paho/]
16. Projekthomepage Mqtt-spy: [https://kamilfb.github.io/mqtt-spy/]
17. Downloadseite Mqtt-spy:[https://github.com/eclipse/paho.mqtt-spy/wiki/Downloads]
18. Cloud MQTT: [https://www.cloudmqtt.com]
19. Magic Home LED Controller: [http://tinkerman.cat/magic-home-led-controller-espurnad/#more-1614]
20. Open HAB: iTEAD Sonoff Switches and Sockets: [https://community.openhab.org/t/itead-sonoff-switches-and-sockets-cheap-esp8266-wifi-mqtt-hardware/15024/1]
21. Sonoff Tasmota: [https://github.com/arendst/Sonoff-Tasmota]

Der Autor

Rainer Poisel arbeitet als Embedded Software Engineer bei der Logi.cals GmbH in Niederösterreich. Dort kümmert er sich um die Portierung des Laufzeitsystems

Logi.RTS auf verschiedenste Plattformen: vom 8-Bit-Mikrocontroller zum Multi-Core-Industrie-PC. Daneben entwickelt er eine Linux-basierte Eigenbaulösung zur Heimautomatisierung.