Sonoff

Sonoff is a product line of home automation devices which communicate via WiFi. The chinese manufacturer also provides a cloud service and an App to controll all the dvices in your home network.

the most popular sonoff product is their sonoff basic, a single relay on/off switch. It became hugely popular because of two things:

1. it is extreamly cheap! (as in $5 on Aliexpress / Banggood / ebay etc.)
2. it is based on an ESP8266 which by itself is an extremely popular module and for which multiple firmware variants exist including an arduino compatible one. in other words, the sonoff-basic can be re-flashed with an arduino firmware and then used for whatever you want.

Hack it without changing firmware

but even without re-flashing the sonoff switch, it is possible to use it without their cloud service

alternative firmware

- Tasmota
  - works also on some other ESP based devices
  - provides control via serial / web / MQTT

What I intend to do with it

I use digitalSTROM as my main bus system in my House. It has a web-interface served by the “DitialSTROM Server” (DSS) which allows the definition of rooms and scenes. Devices can be assigned to rooms and then different scenes can be programmed for each room or section of a room. Further more, the DSS provides a JSON web-api which enables the use of external tools to control or monitor the stat of the House. DigitalSTORM communicates over a sort of powerline network, so it needs no extra cables for the bus lines but it isn't exactly wireless either. For this to work you need to put a singnal-injector behind every circuit breaker in your house. the injectors are connected to one another and make sure, that all your devices can talk to one another even if they are on different phases of your mains power line.

My house contains two appartments. One of which is equipped with digitalSTROM, the other one isn't. Unfortunately, some lights, like for example the flood light for the garden, which is mounted under the roof, are fed from the upstairs appartments power which is not digitalSTROM enabled. It would be hidiously expensive and I don't have any space left in the upstairs fuse box to mount all the digitalSTORM injectors and stuff just to get one Light onto my bus. That's where sonoffs come in. by connecting a sonoff to my WiFi netowrk and linking it to my digitalSTROM server, i can control the flood light over my digitalSTROM environment even if it is not in the same bus.

since there is the Tasmota Firmware for sonoff that supports MQTT and there is also an MQTT bridge

for digitalSTROM and because MQTT seems to be a nice protocol for IoT applications, i decided to use that to glue the two things together :)

at first i will use the existing MQTT bridge for DS which uses the Web-API. In the future i would like to write a little module that uses the Virtual Device API of digitalSTROM which would allow me to emualte a digitalSTROM device for my sonoff. this would make the integration even smoother and there would be no need for an external rules engine anymore. However, i'll get started with the web-api and rules engine first, as I'd like to try out some MQTT stuff first :)

Falshing Tasmota Firmware

this is straight forward, just follow the guides on the Tasmota github site. there is a wiki with loads of information.

I personally went for flashing over a USB TTL adapter using the Arduino IDE. It is important to notice, that the TTL adapter needs to provide 3.3V, not 5V! Also i recommend using a PL2303 based USB TTL adapter, as others did not work for me (spent hours trying until I finally gave up and tried with a PL2303 based one I had laying around). Another important Point was, how to enter the programming mode: Hold down the pushbutton while connecting only the power pins. Leave at least one of the two data pins disconnected. Then let go of the button and connect the RX and TX (crossed). Flasing did not work for me when i connected all wires at the same time while holding down the button.

I strongly recommend to edit the settings before compiling and adding your wifi credentials, this will save some time and hassle when configuring the device later on. It will simply connect to your wifi and you can access its webpage to do the rest of the configuration.

Here are the steps it took to flash the stock sonoff (current and more detailed instructions can be found in the Tasmota wiki

1. download the latest Arduino IDE and unpack it.
2. create a subdirectory called `portable`
3. start your arduino IDE and go to `File→preferences` and add this to *Additional Boards Manger URLs*: `http://arduino.esp8266.com/stable/package_esp8266com_index.json`
4. select OK and go to `Tools-Boards…-Boards Manager…` and enter ESP into the search. now install the `esp8266` module and close that window.
5. download the sourcecode tar.gz or zip package from the Tasmota releases page
6. extract the contents of the `lib` directory into your arduino ide's `portable/sketchbook/libraries` directory and extract the entire `sonoff` directory into your arduin IDE's `portable/sketchbook/` directory
7. verify your settings under Tools are like these:

```
Board: "Generic ESP8266 Module"  <<<<!!!!
Flash Mode: "DOUT"  <<<<!!!!
Flash Size: "1M (no SPIFFS)"  <<<<!!!!
Debug Port: "Disabled"
Debug Level: "None"
LwIP Variant: "v2 Prebuilt (MSS=536)"
Reset Method: "ck"
Crystal Frequency: "26 MHz"
Flash Frequency: "40MHz"
```

```
Upload Using: "Serial"
CPU Frequency: "80 MHz"
Upload Speed: "115200"
Port: Your COM port connected to sonoff
```

8. edit `user_config.h` and enter your wifi name and password. all other settings can be changed later once the device is booted, but you can of course set them here if you want. this will make it much faster if you flash alot of these devices and want the same settings on all of them.
9. **make sure your sonoff is disconnected from the AC power lines***
10. Open sonoff and plug in some Male-Female dupond cables (jumper cables). The 5 pins going from the pushbutton towards the long row of holes are what will be used here. the pin closest to the pushbutton is the +3.3V pin, followed by RX, TX, GND, GPIO14. You won't need GPIO14. On my first unit I soldered a set of pin headers onto the holes, but that's usually not necessary if you have the right cables at hand.
11. connect 3.3V and GND and to the USB TTL, hold down the pushbutton and plug the USB-TTL into your computers USB Port
12. now connect the sonoff's RX to your USB-TTL's TX and vice versa
13. click the upload button on your arduino IDE. it will compile and upload the image.
14. once the upload is complete, disconnect the USB TTL from your computer and plug it back in a few seconds later. your sonoff should now boot (green light flashes shortly) and after a while it should be connected to your wifi and it should serve a web interface on its ip address. in case you can't reach it, try pushing the button once to turn on the LED, this might also activate the webserver on first boot.
15. once you have verified that the firmware boots okay, you can simply disconnect the dupond cables and assemble the case again, before you connect your sonoff to the AC power source.

setting up a MQTT server

I have to say this is the first time i got in contact with MQTT. MQTT is a light weight protocol which is becoming more and more popular for IoT applications. It works on a subscribe/publish basis. There are MQTT clients and there is one MQTT Broaker or Server (same thing). Clients can both publish and subscribe to topics on the broker. a topic “name” is like a linux path. it can have several levels which can be freely chosen. To make things clearer, here is an example with our sonoff device:

- lets assume our sonoff device controls a light in the garden. we therefore call it `garden_light`
- the sonoff device is an MQTT client. it subscribes at the broker mqtt.psuter.ch to the topic `cmnd/garden_light/power` and it publishes amongst other things the topic `stat/garden_light/power`
- on a computer we can subscribe to the topic `stat/garden_light/#` and then publish to the topic `cmnd/garden_light/power` and set a value of `1`.
- since the sonoff is a subscriber of `cmnd/garden_light/power` it will receive our published infomration `1` for that specific topic. the sonoffs firmware will then turn on the light. this causes the status of the relay to change to “ON”, so it will publish to `stat/garden_light/power` and set a value to `ON`.
- the broker will now look for clients holding a subscription for `stat/garden_light/power` and it finds our PC which did subscribe to all sub-topics of `stat/garden_light` and hence gets the updated value for `power`

there is a more detailed overview at HiveMQ and another one at steves-internet-guide.

while one can use a public MQTT server as well, I prefer to have such things at home. Especially since by default Tasmota does not support SSL encryption. Also it is always when you don't need to bother so much about security when first trying out new stuff :)

Mosquitto is an open-source MQTT server or broker, which I am going to install and configure here to handle all sonoff devices. Luckily it is already pre-packed in most distributions, so we don't need to go through a lengthy installation process:

1. set up the server (on `mqtt.psuter.ch` in my case, which runs ubuntu)
    1. `apt install mosquitto`
    2. add some custom configuration options to `/etc/mosquitto/conf.d/custom.conf`:

        ```
        log_type all
        connection_messages true
        log_timestamp true
        allow_anonymous false
        password_file /etc/mosquitto/pwdfile
        ```

        this will mainly log verbosely to the default log-file which is at `/var/log/mosquitto/mosquitto.log` and it will require authentication. albeit in plaintext, but it's better than nothing i guess ;)

    3. restart the server.. now i had to stop and start it as it seems to try to start too quick after stopping and the socket might not be free for the new service to start..

        ```
            - now we need to create our password file <code>
        mosquitto_passwd -c /etc/mosquitto/pwdfile sonoffs
        chown mosquitto.mosquitto /etc/mosquitto/pwdfile
        chmod 600 /etc/mosquitto/pwdfile
        ```

        and enter the desired password for our sonoff devices

        ```
        systemctl stop mosquitto.service
        systemctl start mosquitto.service
        ```

    4. in a window open the log file to track what's going on :)

        ```
        tail -f /var/log/mosquitto/mosquitto.log
        ```

2. configure the sonoff device:
    1. in the sonoffs web interface, go to `Configuration - Configure Other` and make usre MQTT is enabled
    2. then go to `Configuration - Configure MQTT` and enter the host name of your server (in my case `mqtt.psuter.ch`), set the client to `garden` and the user to `sonoffs` and enter your recently set password. also set the topic to `garden_light` and click save.
    3. after a few seconds you should see some movement in your mosquitto log.. your sonoff should register and enter its subscriptions and publish its state.
3. install a client to test the server
    1. this is of course optional, but it certainly helps in understanding MQTT to play around with the mosquitto_client.
    2. on a client machine (can be the server as well or any other linux machine in your network) install the `msoquitto-client` package:

```
sudo apt install mosquitto-clients
```

3. now start a client and subscribe to the sonoff topics

   ```
   mosquitto_sub -h mqtt.psuter.ch -u sonoffs -P sonoff -t
   stat/garden_light/#
   ```

   note that the # is a *multi*-wildcard and a + is a *single*-wildcard. this means that a + in a topic can only replace one level, while # can replace several levels at once.

4. if you now press the button on your sonoff, you will receive a status message on your client which was published to the broker and then handed to your client since it had a subscription for that topic.
5. vice-versa, you can now, preferrably in a second terminal on the same machine publish another topic to turn the light on (or off), whatever makes sense:

   ```
   mosquitto_pub -h mqtt.psuter.ch -u sonoffs -P sonoff -t
   cmnd/garden_light/power -m 0
   ```

   now you can see the information flow described at the beginin of this section.

MQTT interface for digitalSTROM

Chriss Gross wrote a MQTT Bridge for digitalSTROM in node.js and kinly made it available to all of us on GitHub.

Unfortunately there isn't much documentation around there, so at least this time it makes sense for me to write everything down in here that i found out about it :)

in order to set it up, simply install it through npm:

```
npm install --save mqtt-dss-bridge
```

this will create a sub-directory `node_modules/mqtt-dss-bridge`. In there you can find the `config.js` file where you can configure the bridge. You can leave most of it at the default but some things need to be adjusted:

- `refreshInterval`: 5000 milliseconds might be too long.. i set it down to 1000 for now which still is quite laggy when you want to turn other lights on or off in a room based on room actions. but i guess since it needs to actively poll the dss server, I'll try to slowly make my way to the right setting.
- `url` of the MQTT client: i left it at localhost, as this is running on the same server
- `url` of the DigitalSTROM server: this is where you need to enter your DSS' IP Address. Leave the Port at 8080, as the API uses a different port than your web-interface
- `appToken`: this was the tricky part: first open the url https://<yourDSShere>:8080/json/system/requestApplicationToken?applicationName=dssBridget in a browser where you are not logged-in to the digitalSTROM Server's web-interface. you will receive an appToken in the response. Copy/Paste that token into your config.js. Following that, log-in to your DSS Web-Interface and go to the **advanced view → system → Access**

Authorization. in there you should now see an entry for “dssBridget” with the ending of the access token shown behind it. check the checkbox and click apply. only now, your mqtt bridge will actually have access to the DSS.
- since this mqtt bridge does not seem to be set up to support any authentication or other security for now, I simply enabled anonymous access to my MQTT server. certainly not something i want to keep this way for ever, but that has to wait until i got it all sorted out :)

now simply run `node index.js` to start the bridge.

you will notice alot of activity in your mosquitto log :)

in order to see all the topics your dss publishes, you may run a `mosquitto_sub` client that subscribes to `dss/#` like so:

```
mosquitto_sub -h mqtt.psuter.ch -u sonoffs -P sonoff -v -t dss/#
```

this helps in finding your way around the massive amount of inormation that is published by the dss bridge.

Since my goal for now is to turn on my sonoff when a certain scene is selected in a room i am most interested in the topic `dss/apartment/zones/Garden/groups/1/lastCalledScene`. with the above command I found out, that when i select scene 1 the value of that topic acutally becomes a json value of `{ “$value”:5 }`. When the light is turned off, the value changes to `0` instead.

In order to turn the lights on in a room, you can publish to the topic `set/dss/apartment/zones/Garden/groups/1/lastCalledScene` and set the value to `5` or `0`. here is an example using `mosquitto_pub`:

```
mosquitto_pub -h mqtt.psuter.ch -u sonoffs -P sonoff -t
set/dss/apartment/zones/Garden/groups/1/lastCalledScene -m 5
```

Future projects

eventually I'd like to write a script that uses the plan44 vdcd external device API to integrate a sonoff device into a digitalSTROM setup as a Joker fixture (can be used for all available sort of fixtures like light, shade, ventilation etc.).