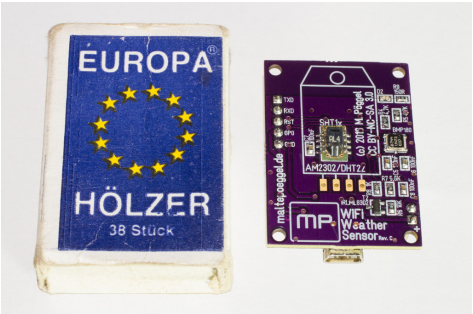
WiFi Weather Sensor

Die eigene Wetterstation im Internet der Dinge

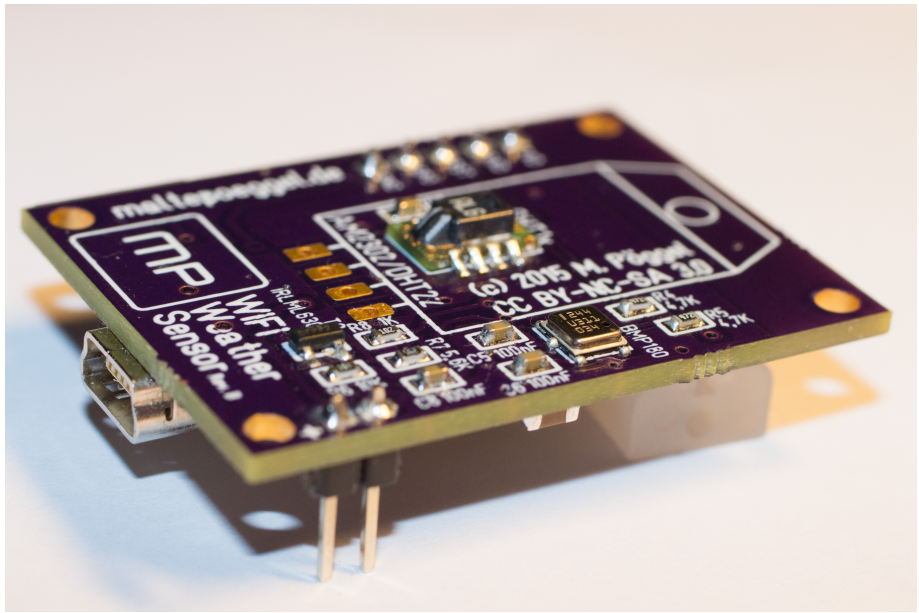
Malte Pöggel

Maker Faire Hannover

28.05.2016



Inhalt



Wetterstation

Eine Wetterstation ist eine Zusammenstellung verschiedener Messgeräte, die zur Messung meteorologischer Größen und damit der Wetterbeobachtung an einem bestimmten Ort und der Klimaforschung dienen.[1]

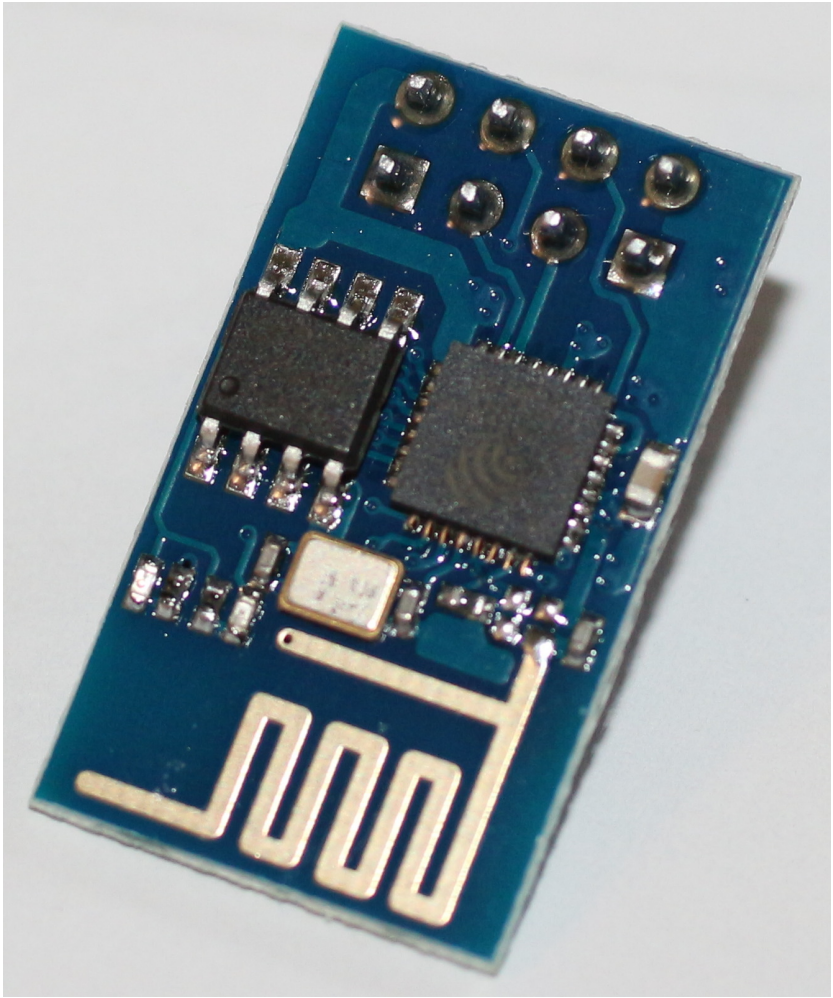
[1] Quelle: https://de.wikipedia.org/wiki/Wetterstation
[2] Bild: Wikipedia / Markus Schweiss

Internet der Dinge

- Intelligente Gegenstände
- Kleine, eingebettete Computer
- Unterstützung im Alltag
- Sensoren und Aktoren
- Übermittlung von Zuständen über das Internet

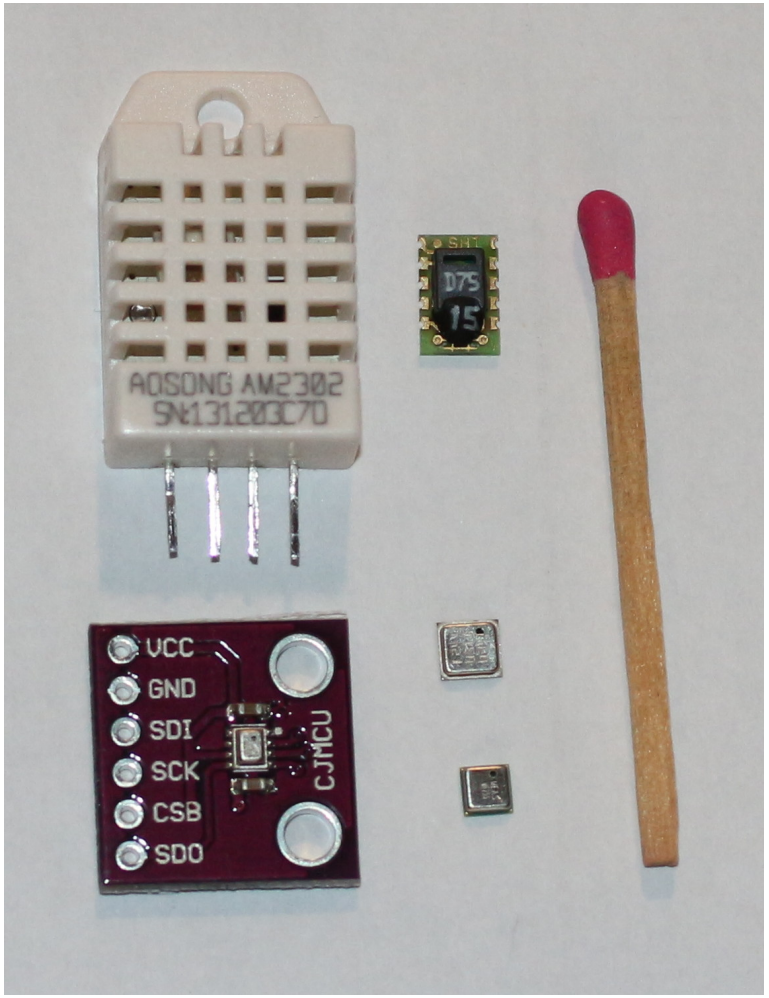
ESP8266 WiFi SoC

- 32-bit Mikrocontroller
- WLAN 802.11b/g/n
- SPI, UART
- GPIO Ports
- ADC
- Sleep Mode $<10\mu$A

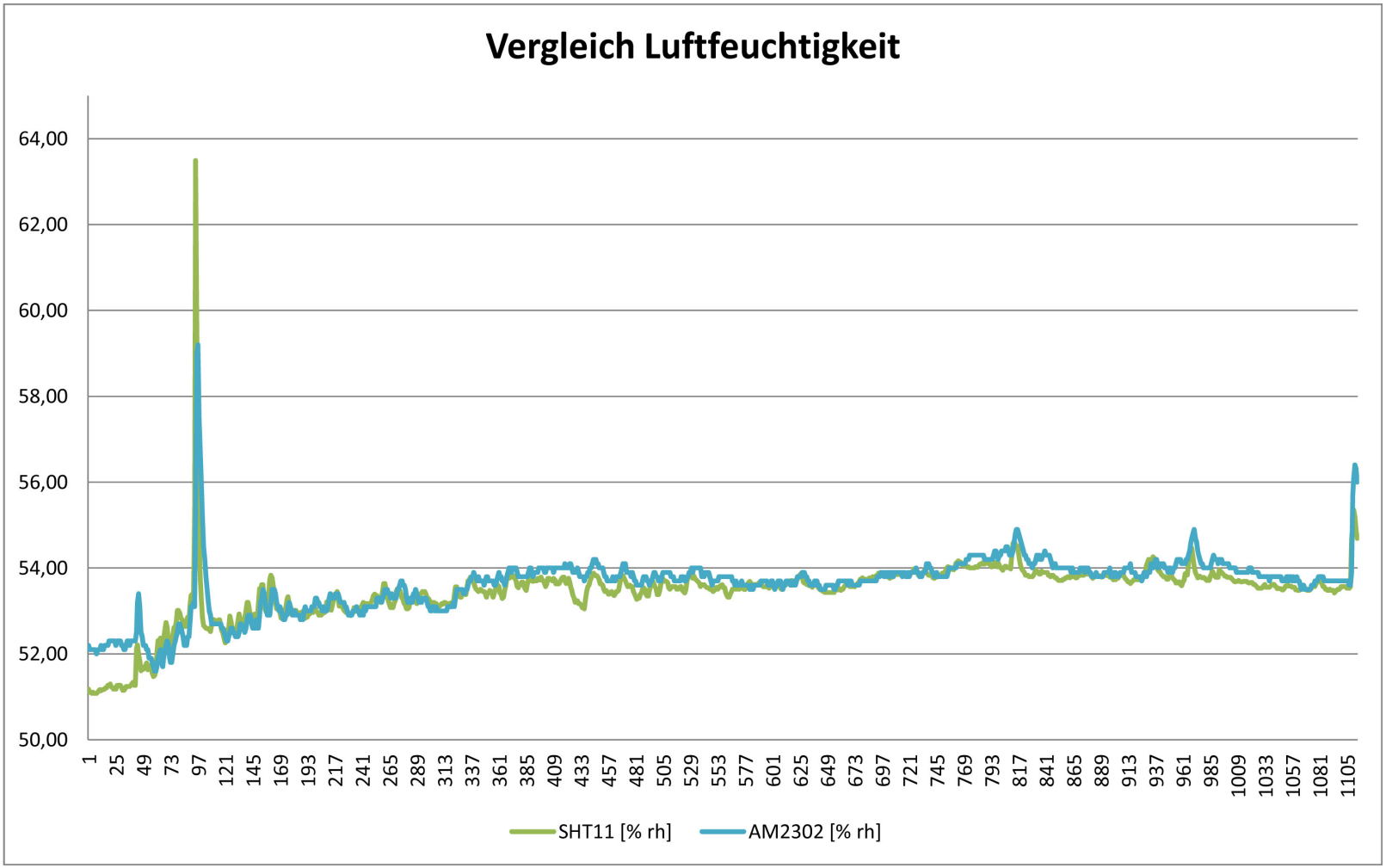


Überblick der Sensoren

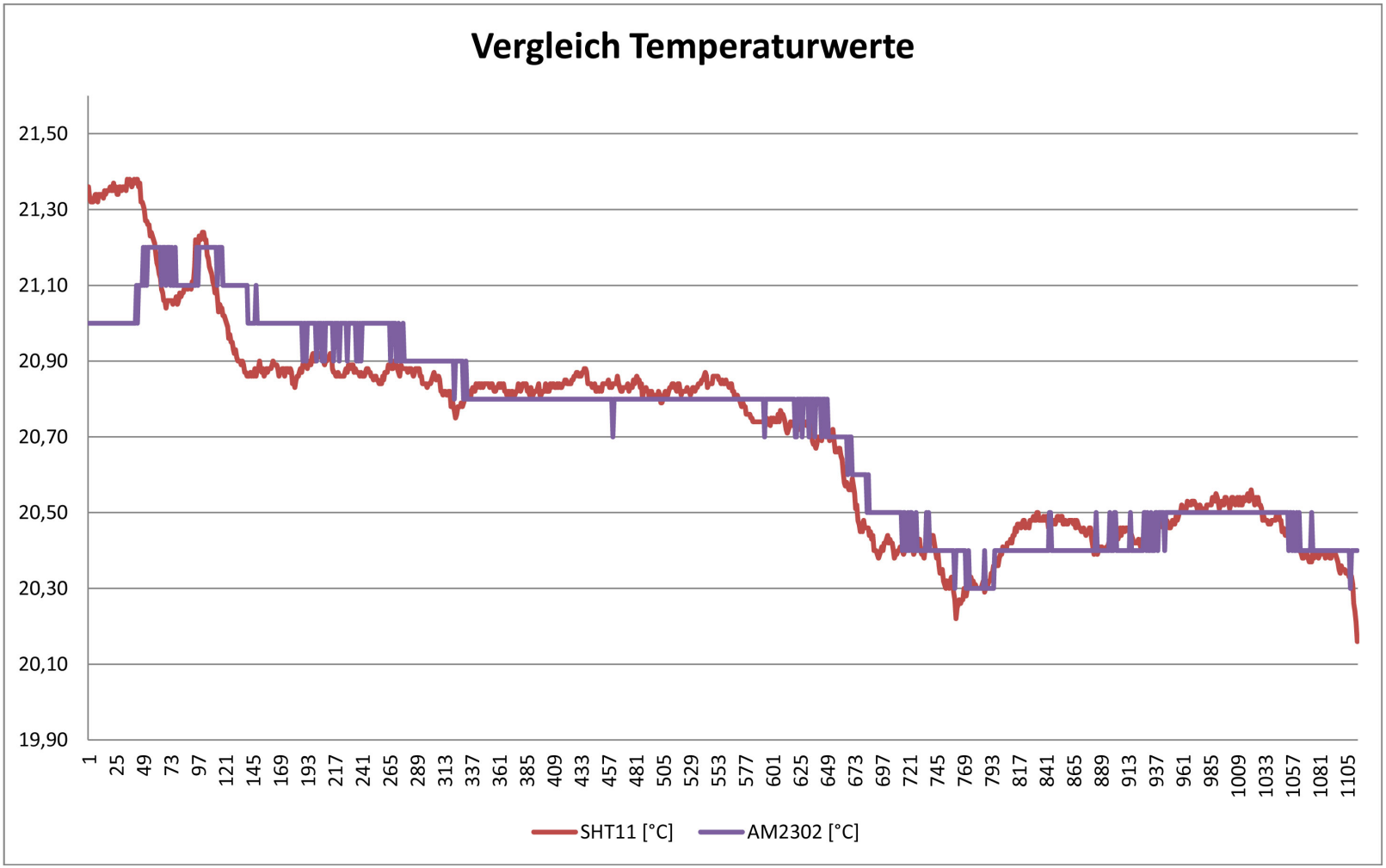
- Temperatur & Luftfeuchte
  - SHT11 / 15
    - 0,05%RH (±2%RH)
    - 0,01°C (±0,3°C)
  - DHT22 / AM2302
    - 0,1%RH (±5%RH)
    - 0,1°C (±0,5°C)
- Luftdruck
  - BMP180
    - 0,01hPa (±0,12hPa)
  - MS5637
    - 0,016hPa (±0,1hPa)



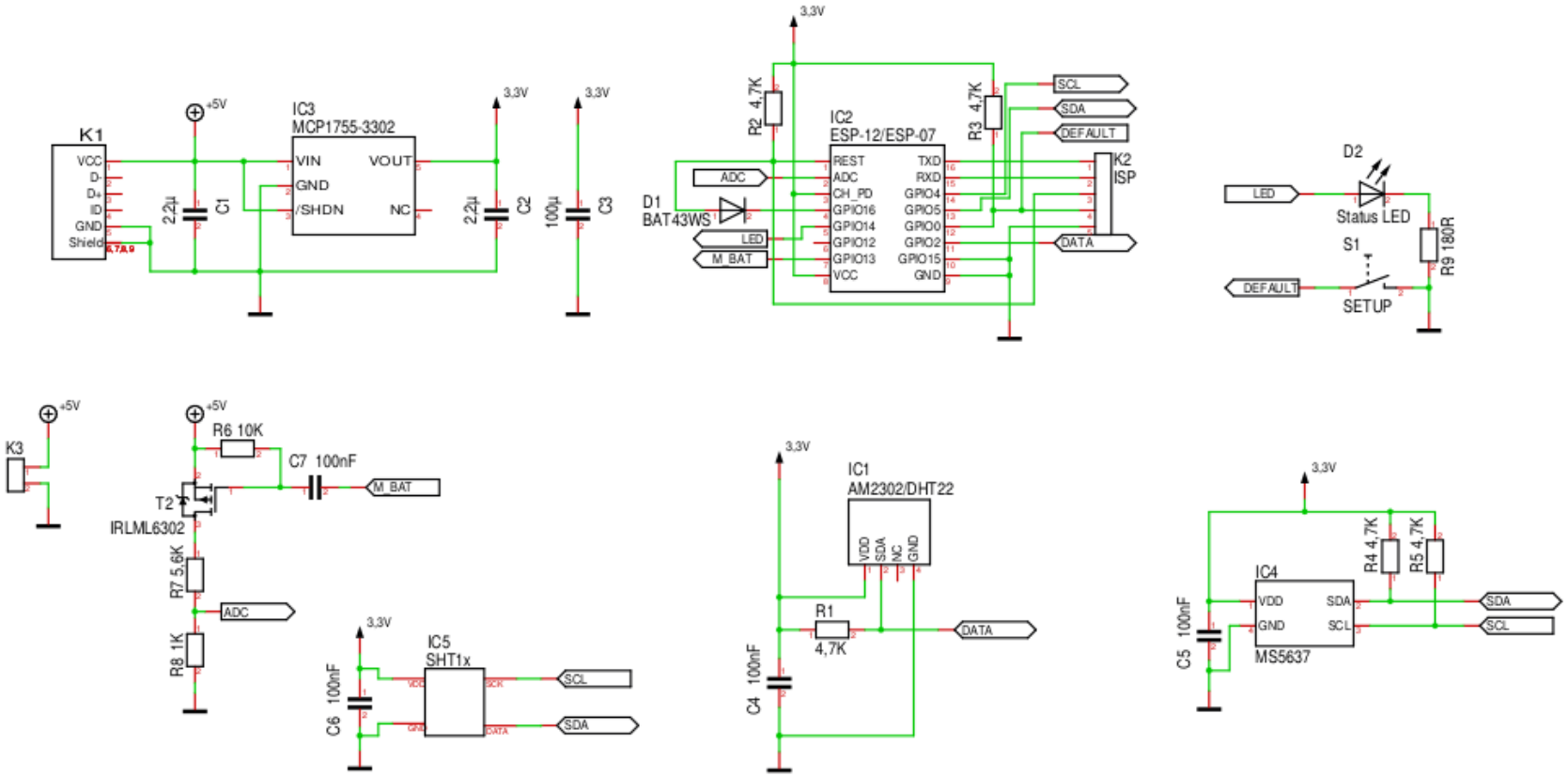
SHT11 vs AM2302



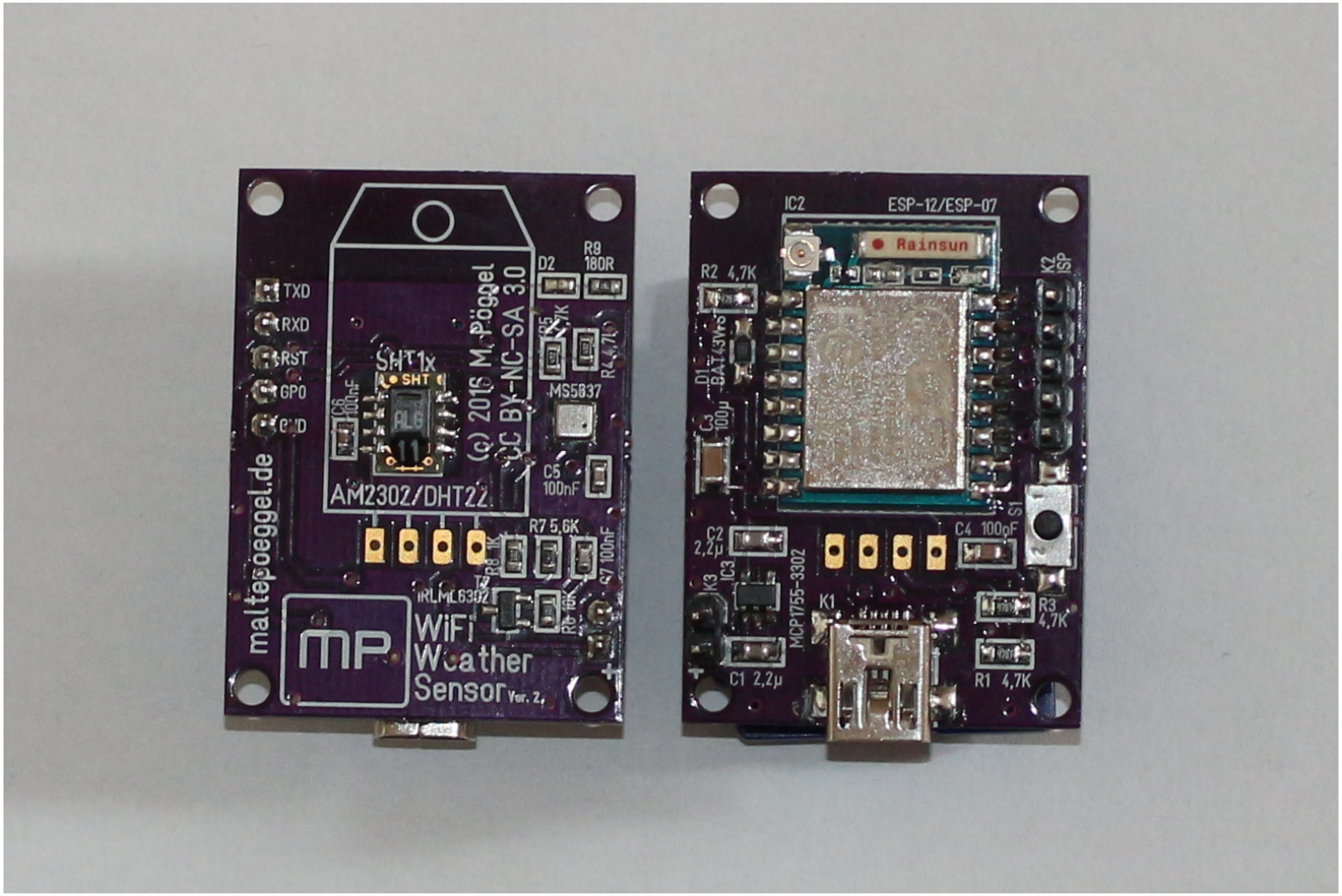
SHT11 vs AM2302



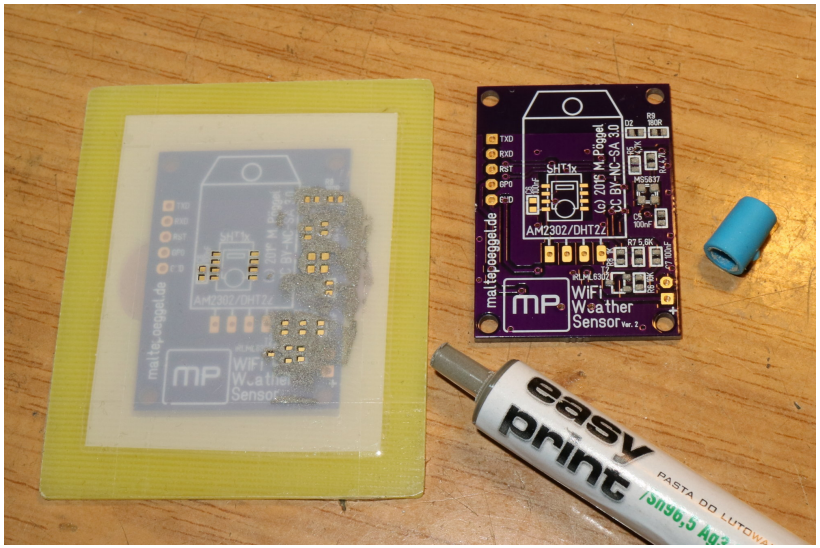
Schaltplan

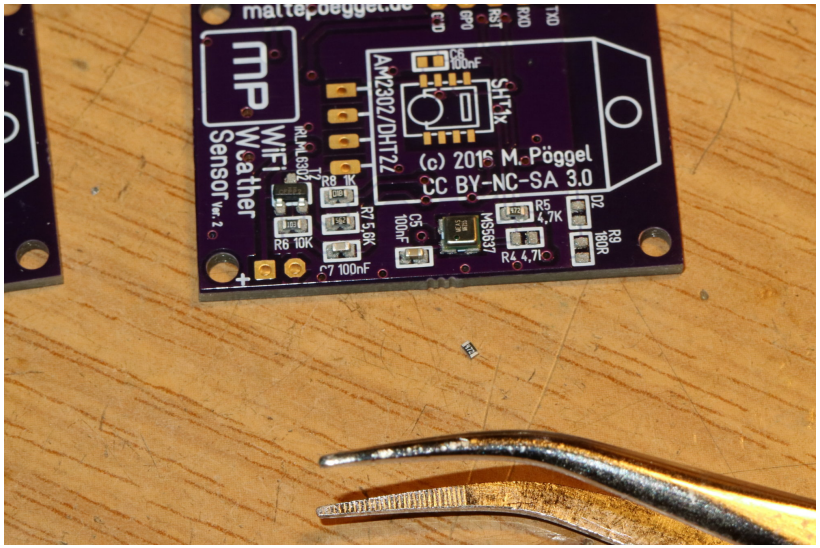

K1
VCC
D-
D+
ID
GND
Shield
+5V
IC3
MCP1755-3302
VIN
GND
/SHDN
VOUT
NC
C1
2,2µ
C2
2,2µ
C3
100µ
3,3V
IC2
ESP-12/ESP-07
REST
ADC
CH_PD
GPIO16
GPIO14
GPIO12
GPIO13
VCC
TXD
RXD
GPIO4
GPIO5
GPIO0
GPIO2
GPIO15
GND
R2 4,7K
R3 4,7K
D1
BAT43WS
ADC
LED
M_BAT
SCL
SDA
DEFAULT
DATA
K2
ISP
D2
LED
Status LED
S1
SETUP
R9 180R
K3
R6 10K
C7 100nF
M_BAT
T2
IRLML6302
R7 5,6K
R8 1K
ADC
IC5
SHT1x
C6 100nF
SCL
SDA
IC1
AM2302/DHT22
VDD
SDA
NC
GND
R1
4,7K
C4 100nF
DATA
IC4
MS5637
R4 4,7K
R5 4,7K
C5 100nF

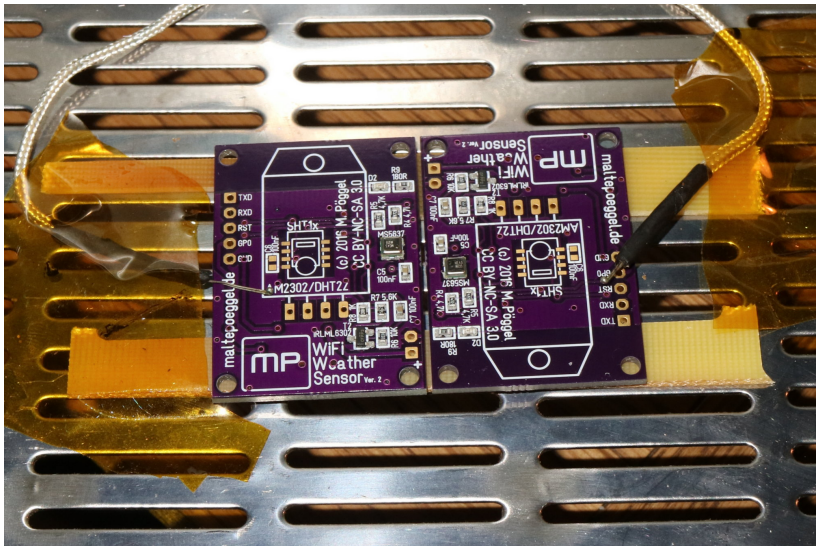
Leiterplatte

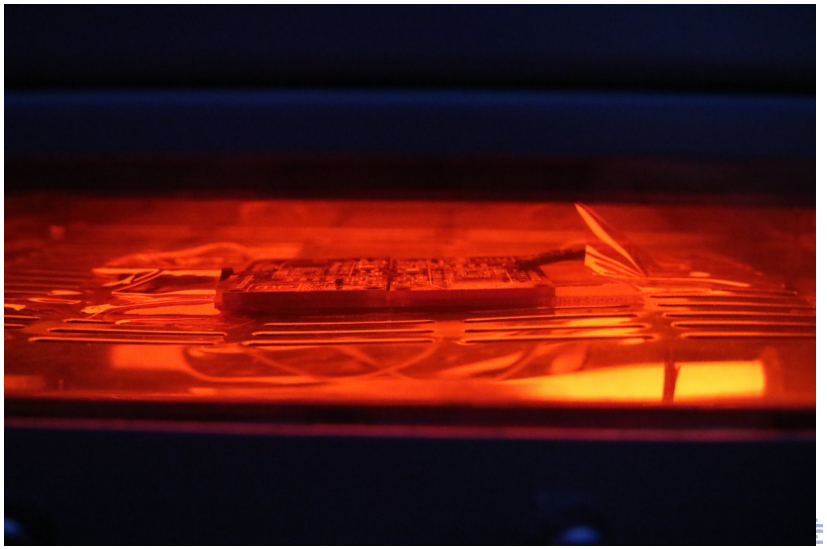


Löten im Reflowofen

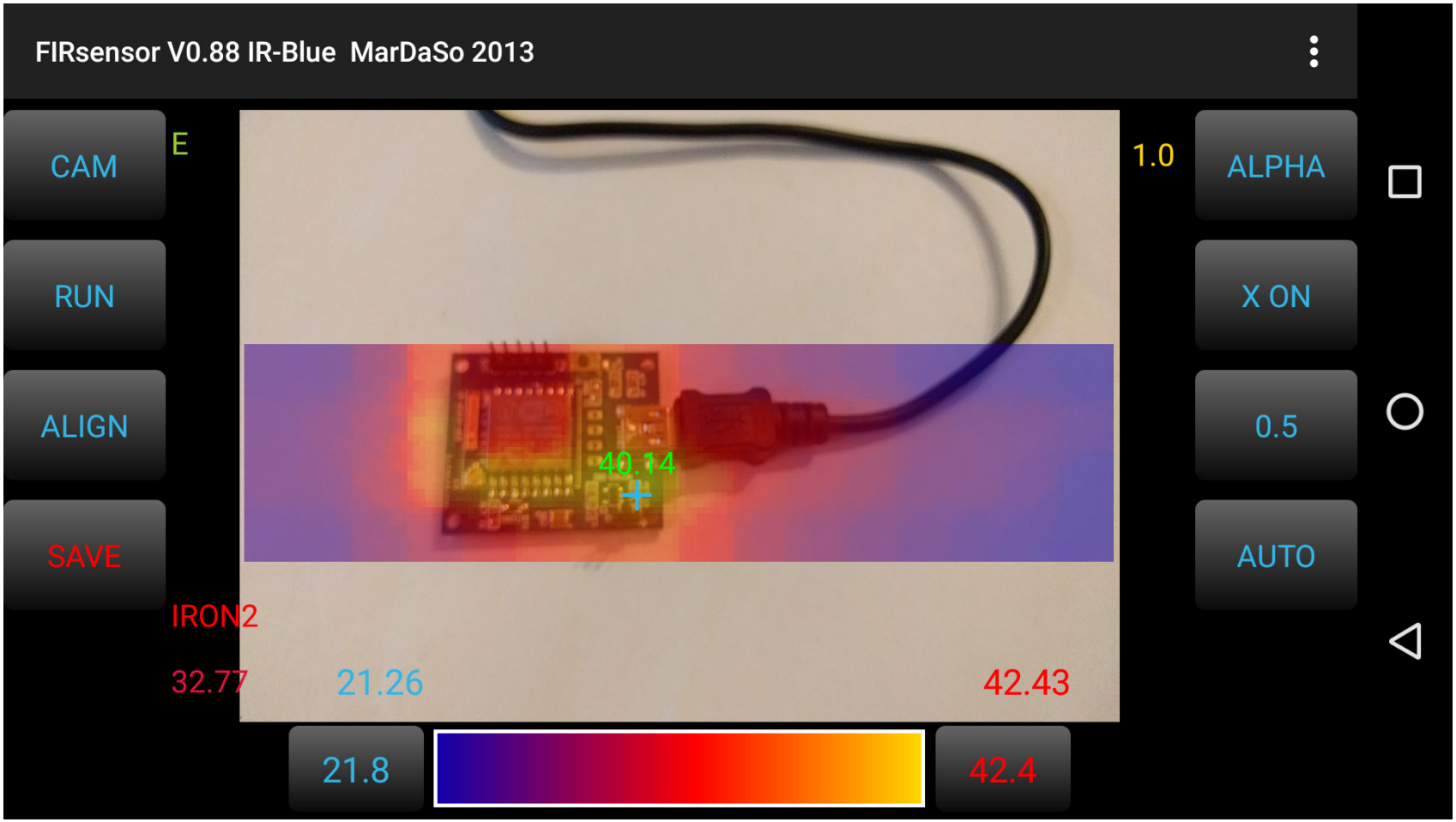








Eigenerwärmung

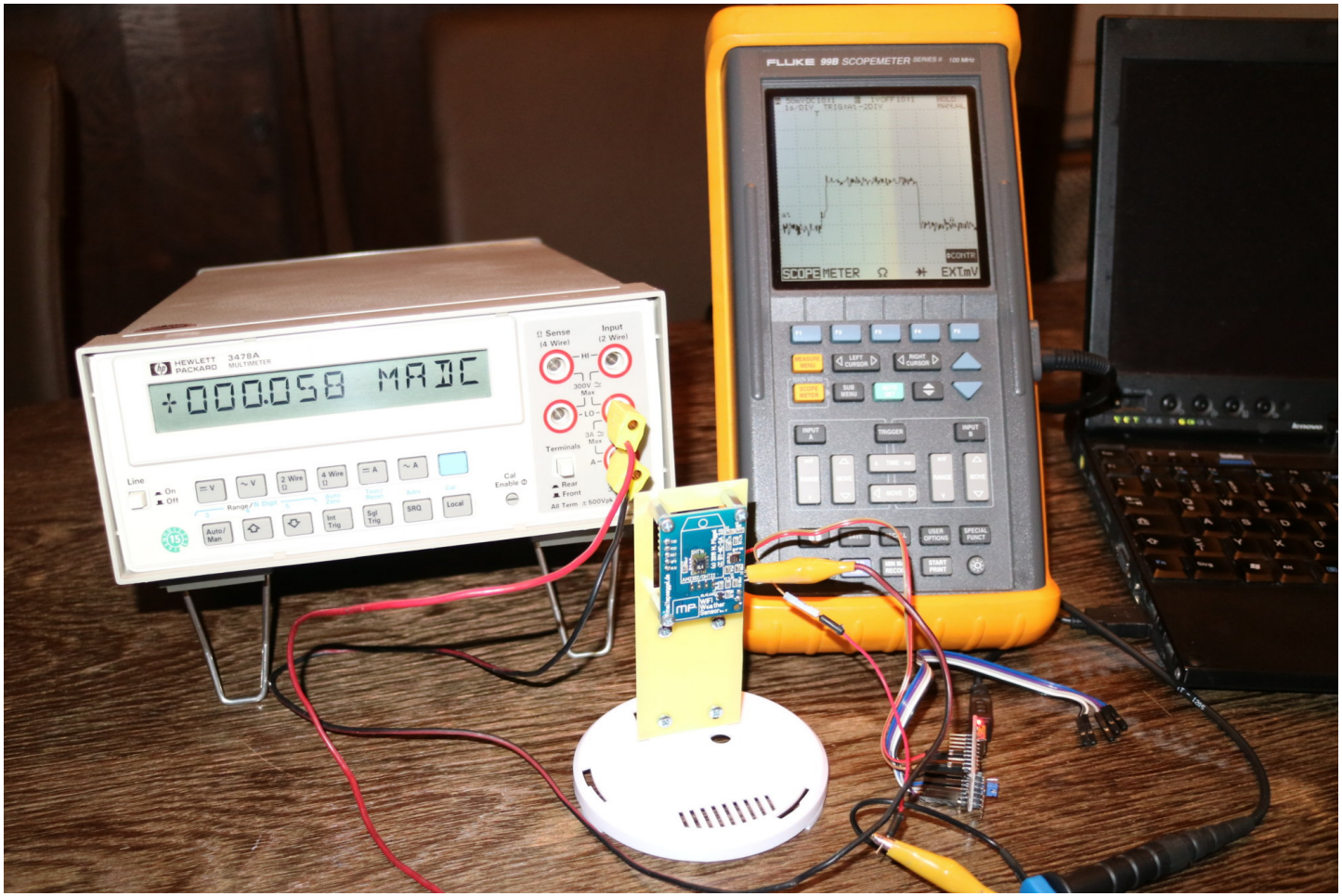


Eigenerwärmung

- Platine hat sich nach 10 Minuten auf 40°C aufgeheizt...
- Lösung: Stromsparmechanismen
  - Messung direkt nach dem Einschalten
  - Daten möglichst schnell übermitteln (∼10sec)
  - Low Power Standby bis zur nächsten Messung

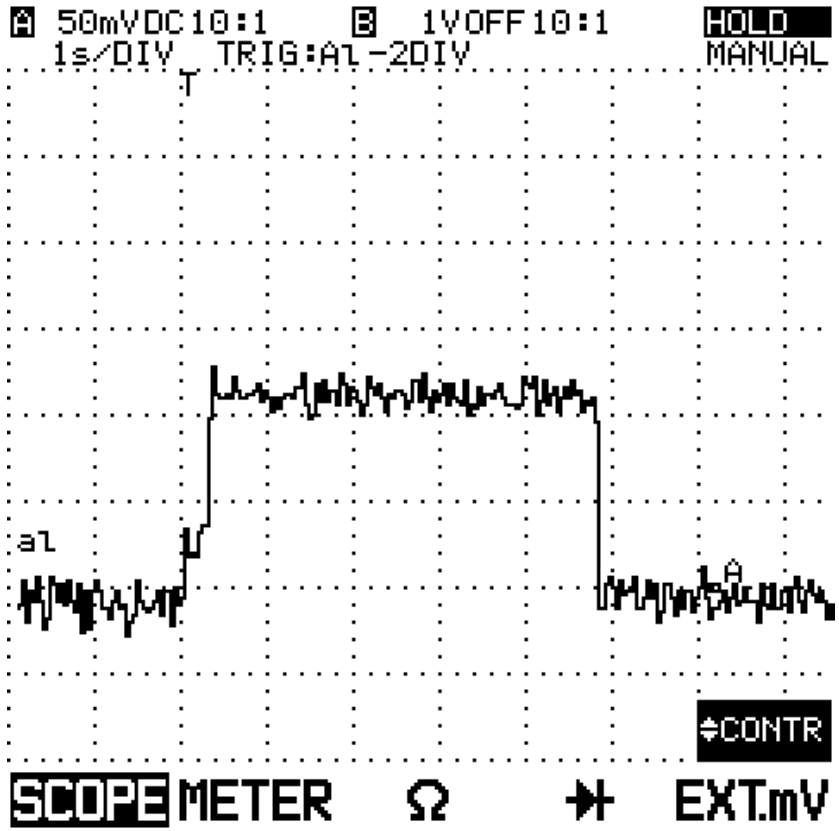
$\longrightarrow$ *Keine Beeinflussung der Messwerte mehr!*

Batterielebensdauer



Batterielebensdauer

- Betrieb: 120mA
- Standby: 60µA
- Betriebsdauer: etwa 5sec



Batterielebensdauer

Ein Intervall von 5min ergibt $\frac{60\text{min}\cdot 24\text{h}}{5\text{min Intervall}} = 288$ Messungen/Tag.

5 Sekunden Betrieb: $\frac{5\text{sec}\cdot 288\text{Messungen}}{60\text{sec}\cdot 60\text{min}} = 0{,}4\text{h}$ pro Tag
$0{,}4\text{h} \cdot 120\text{mA} = 48\text{mAh}$

4:55 Minuten Standby: $\frac{295\text{sec}\cdot 288\text{Messungen}}{60\text{sec}\cdot 60\text{min}} = 23{,}6\text{h}$ pro Tag
$23{,}6\text{h} \cdot 0{,}06\text{mA} = 1{,}42\text{mAh}$

Der tägliche Verbrauch beträgt $48\text{mAh} + 1{,}42\text{mAh} = 49{,}42\text{mAh}$.

Handelsübliche Alkalibatterien mit 2500mAh gewährleisten den Betrieb für 50,6 Tage, also mehr als 1,5 Monate.

Montage

- Schutz vor Sonne
- Schutz vor Regen
- Platz für Modul & Akku
- TFA 98.1114.02
- Preis etwa 15 Euro

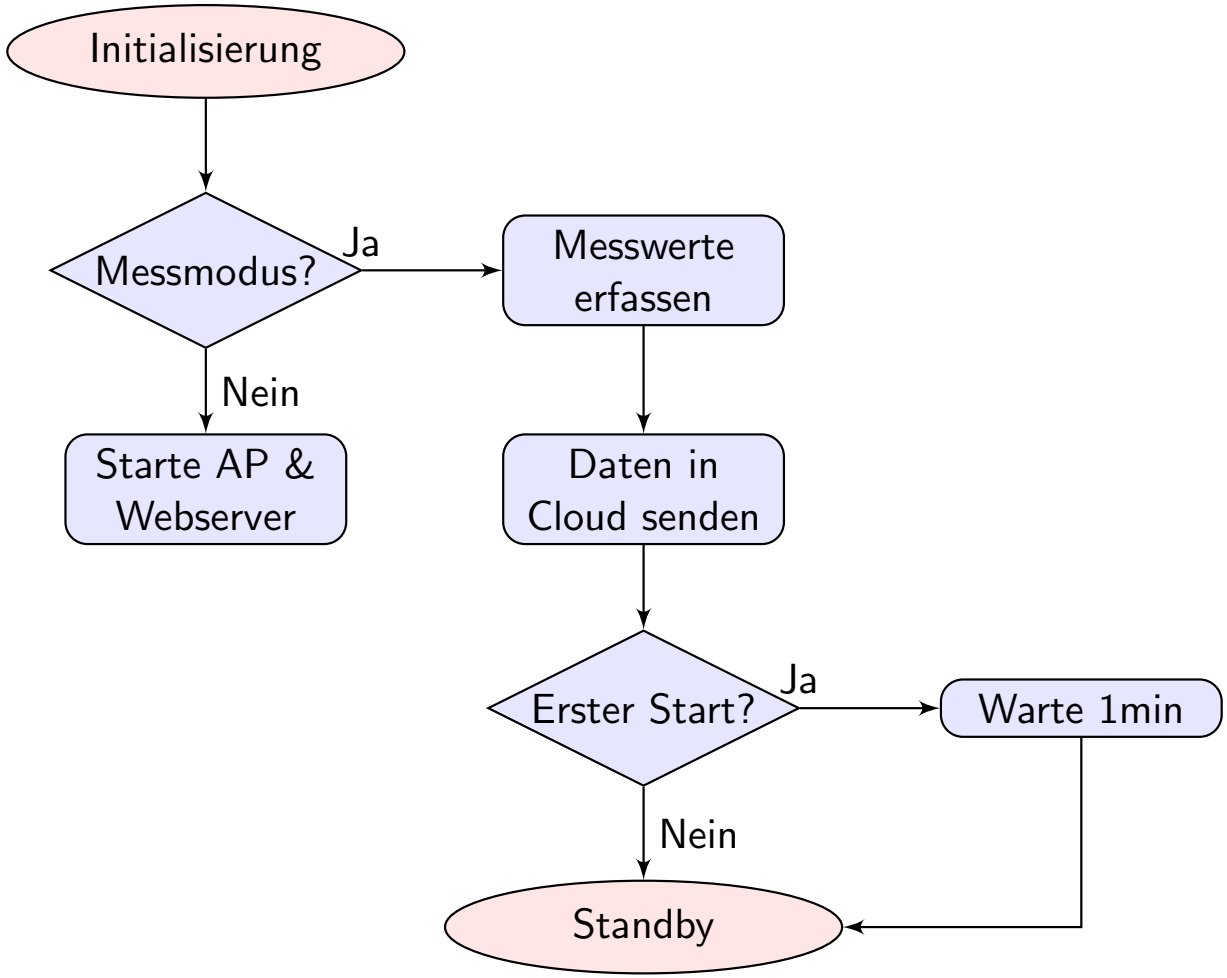
Kosten

- Modul mit SHT11: ca. 39 Euro
- Modul mit AM2302: ca. 22 Euro
- Außengehäuse: ca. 15 Euro
- Halteplatte, Abstandshalter: ca. 2 Euro

Generelles zur Firmware

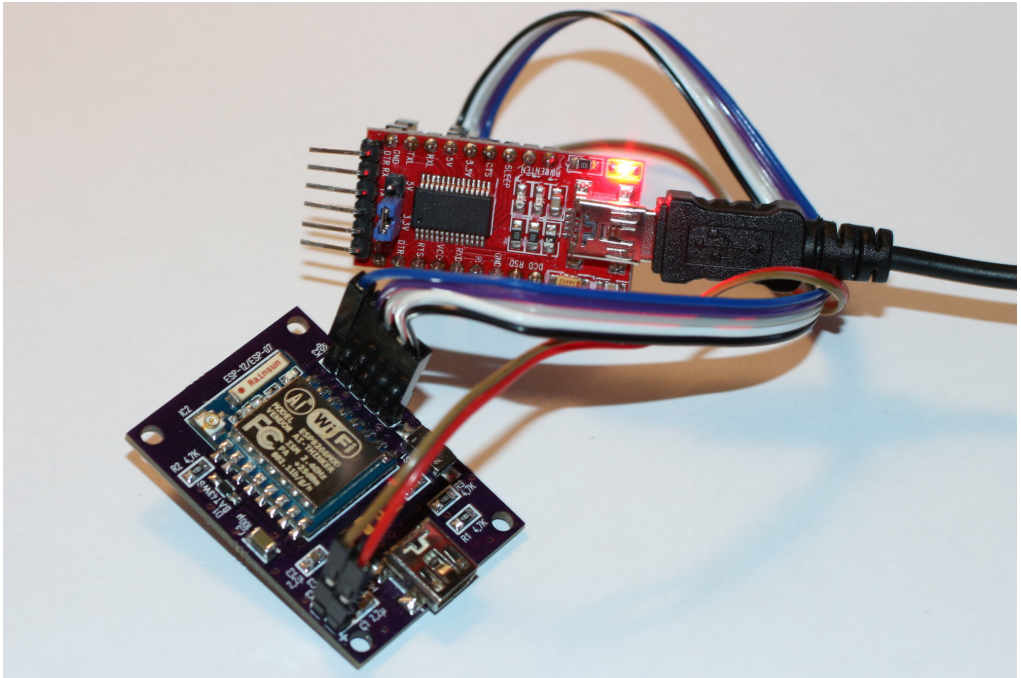
- Compiler: ESP Open SDK
  - GCC (Xtensa LX106)
  - Espressif ESP8266 IoT SDK
  - https://github.com/pfalcon/esp-open-sdk
- Bibiliotheken
  - esphttpd, espfs
    - https://github.com/Spritetm/esphttpd
  - esphttpclient
    - https://github.com/Caerbannog/esphttpclient
  - esp_mqtt
    - https://github.com/vowstar/esp_mqtt
  - I2C Treiber
    - https://github.com/zarya/esp8266_i2c_driver
- Lizenz: GNU GPL

Ablauf


Initialisierung
Messmodus?
Ja
Nein
Messwerte erfassen
Starte AP & Webserver
Daten in Cloud senden
Erster Start?
Ja
Nein
Warte 1min
Standby

Compilieren und Flashen

- Selber compilieren / flashen mittels Toolchain
- Alternativ fertige Binärdatei
- Upload über serielle Verbindung



Funktionsmodi und LED

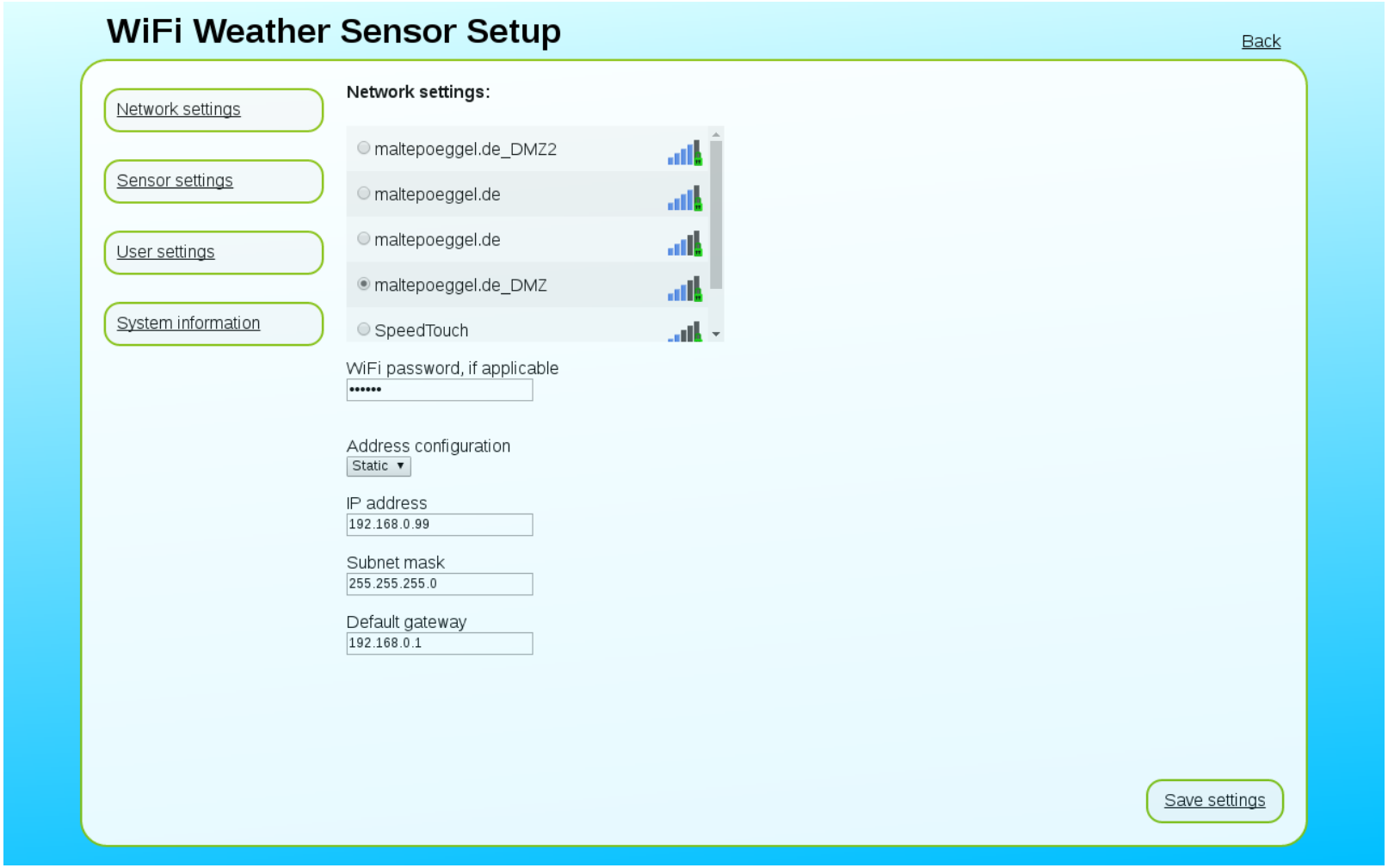
- Taster für Konfigurationsmodus
  - 3sec halten: Umschalten Config / Normal
  - 10sec halten: Werkseinstellungen
- Status LED
  - 1x blitzen: Warten auf WLAN / DHCP
  - 2x blitzen: Datenübertragung
  - 3x blitzen: Fehler beim Senden
  - 4x blitzen: Warten nach erstem Start

Webinterface

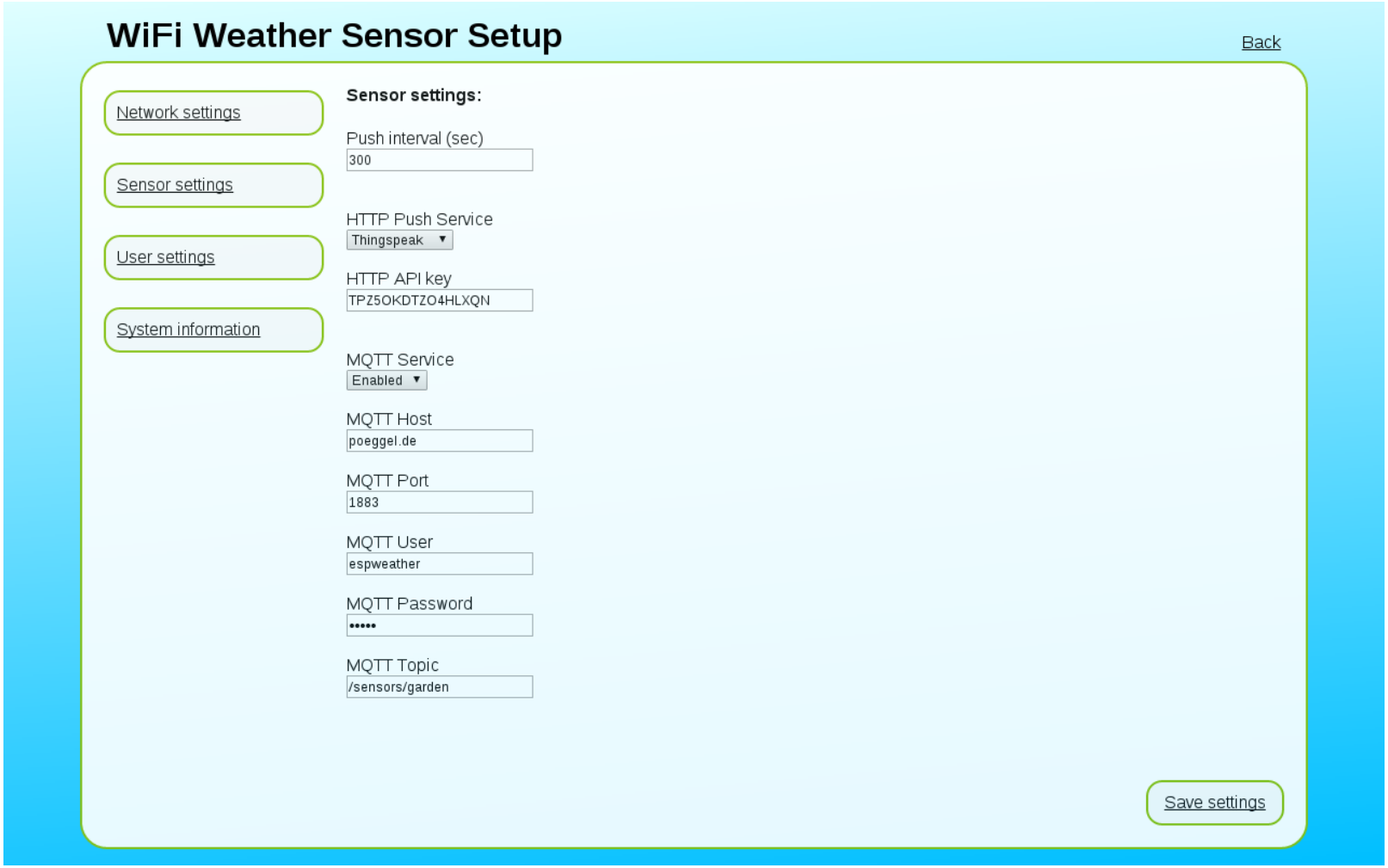
- Im Konfigurationsmodus aufrufbar
- Offenes WLAN
- http://192.168.4.1/
- Benutzer: admin, Passwort: insecure

Webinterface

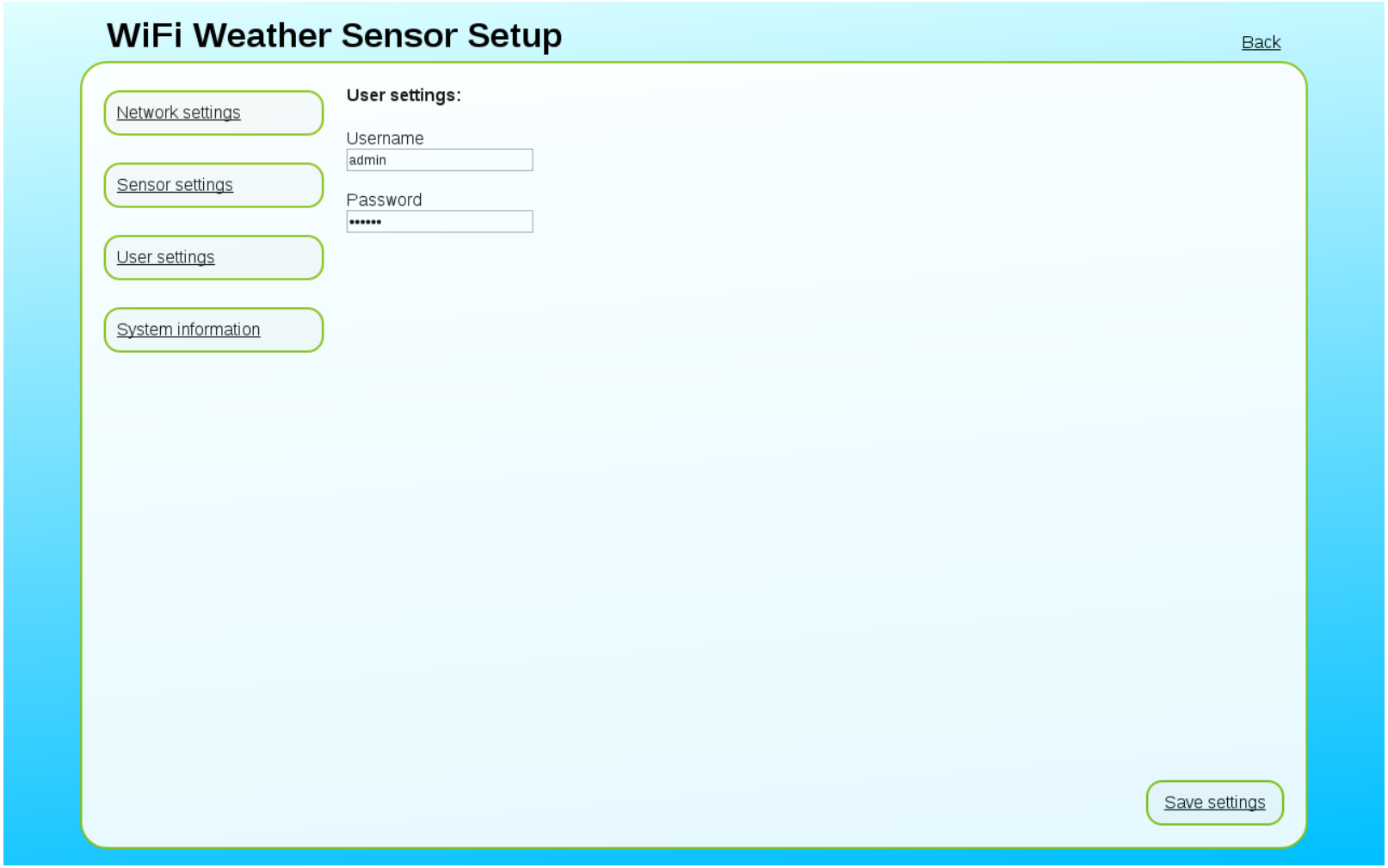
Webinterface



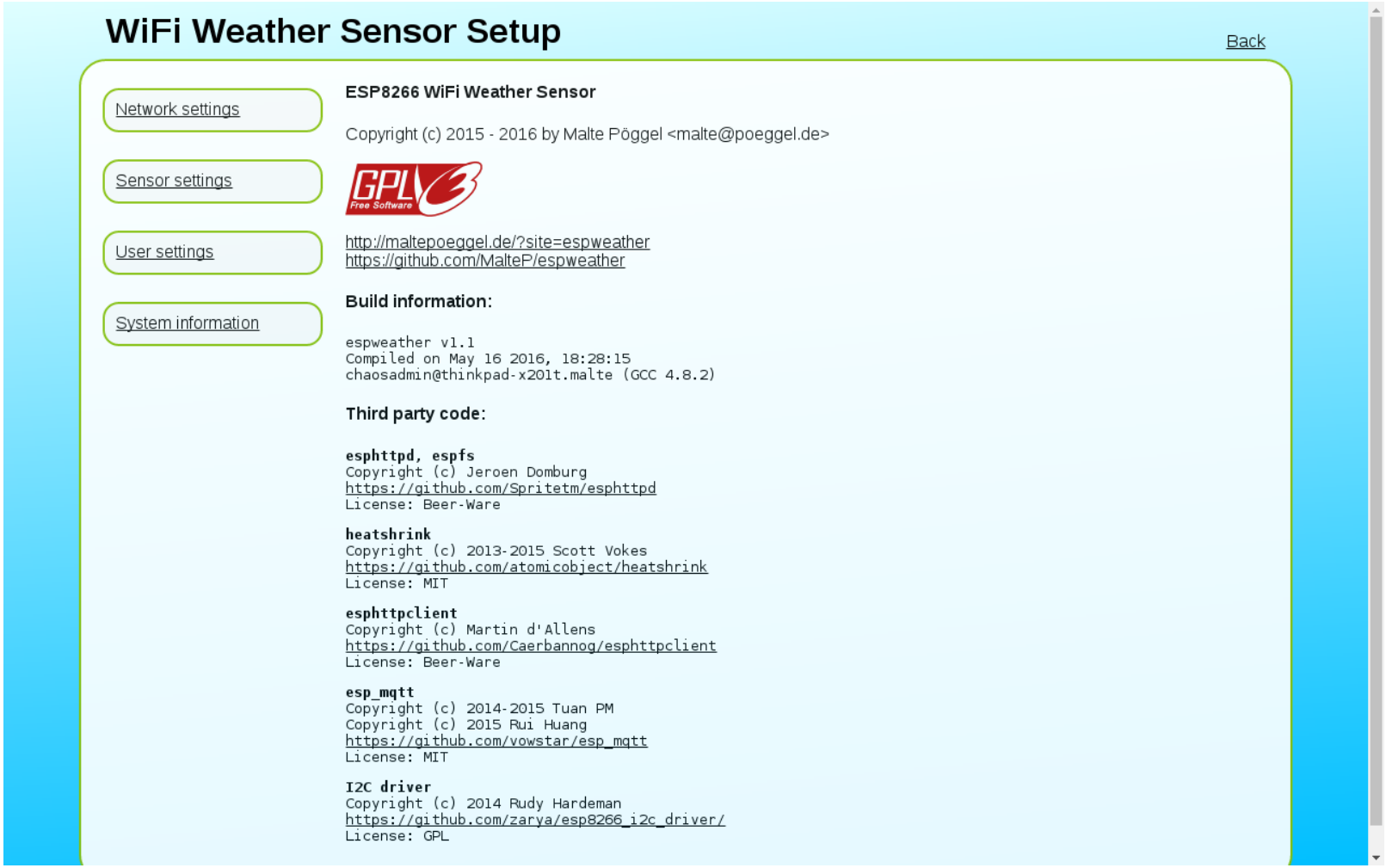
Webinterface



Webinterface



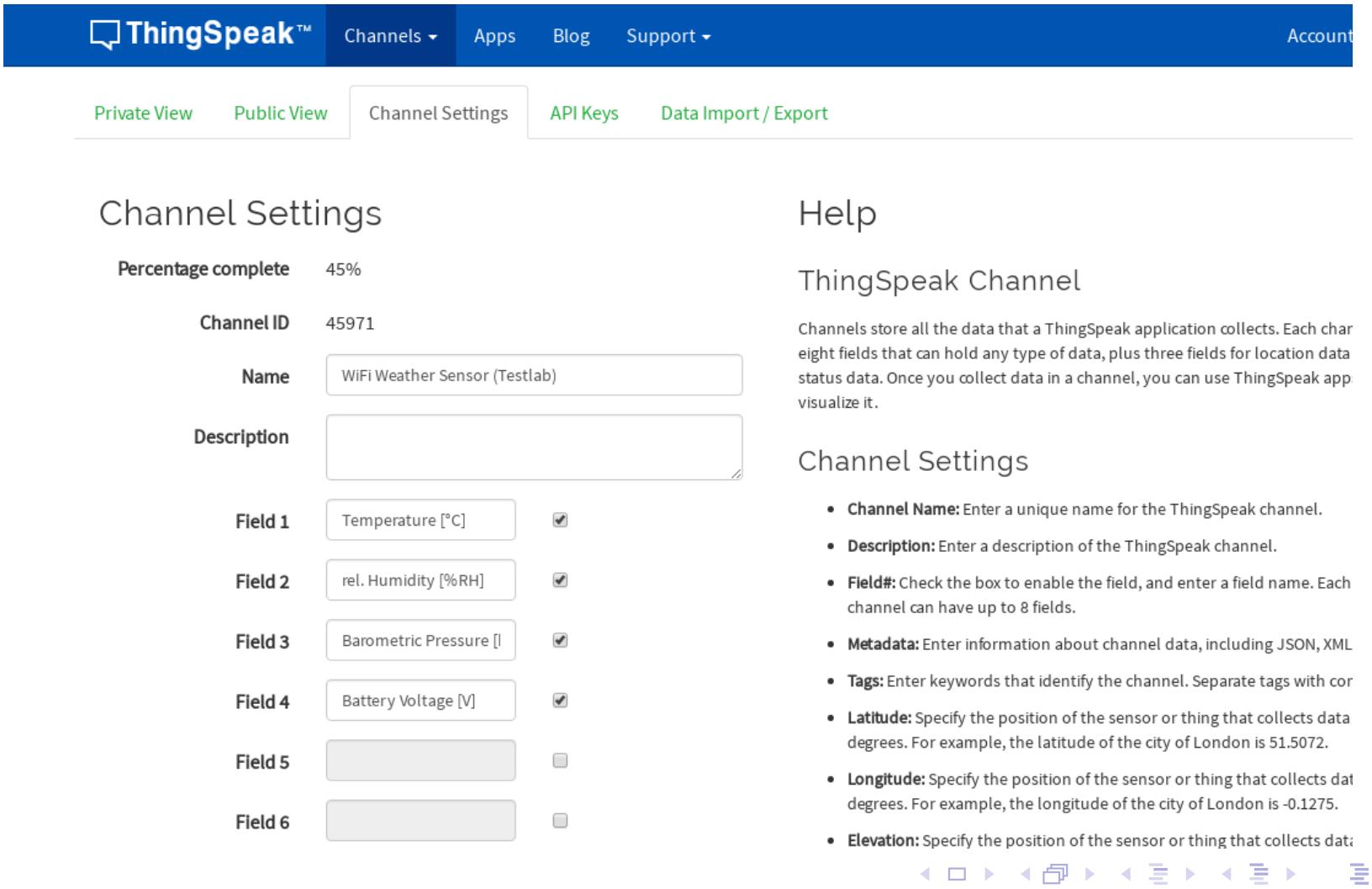
Webinterface



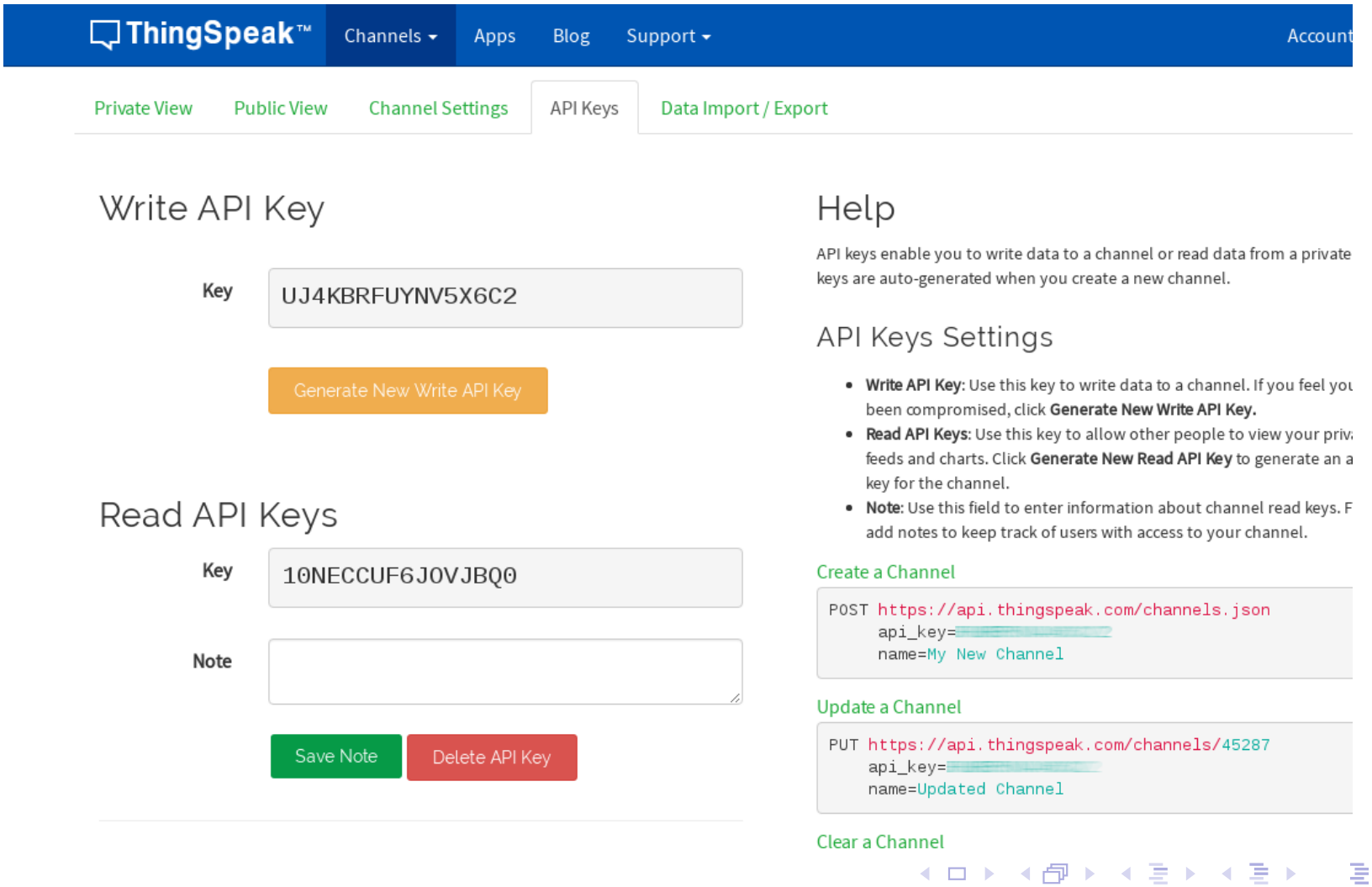
Die Cloud

- Cloud Computing: Bereitstellung von IT-Infrastruktur über das Internet
- In unserem Fall: Datenbank und Visualisierung der Messwerte
- Bekannte Dienste:
  - thingspeak.com
  - adafruit.io
  - data.sparkfun.com

Thingspeak

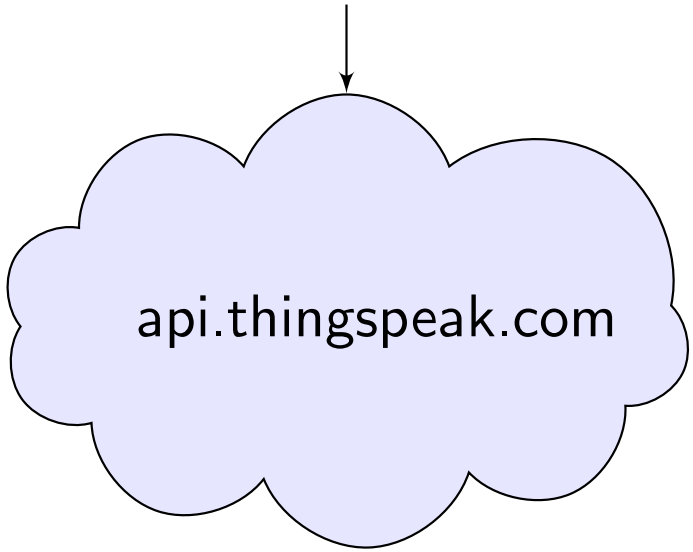

ThingSpeak™
Channels
Apps
Blog
Support
Account
Private View
Public View
Channel Settings
API Keys
Data Import / Export
Channel Settings
Percentage complete
45%
Channel ID
45971
Name
WiFi Weather Sensor (Testlab)
Description
Field 1
Temperature [°C]
Field 2
rel. Humidity [%RH]
Field 3
Barometric Pressure [
Field 4
Battery Voltage [V]
Field 5
Field 6
Help
ThingSpeak Channel
Channels store all the data that a ThingSpeak application collects. Each char
eight fields that can hold any type of data, plus three fields for location data
status data. Once you collect data in a channel, you can use ThingSpeak app
visualize it.
Channel Settings
Channel Name: Enter a unique name for the ThingSpeak channel.
Description: Enter a description of the ThingSpeak channel.
Field#: Check the box to enable the field, and enter a field name. Each
channel can have up to 8 fields.
Metadata: Enter information about channel data, including JSON, XML
Tags: Enter keywords that identify the channel. Separate tags with cor
Latitude: Specify the position of the sensor or thing that collects data
degrees. For example, the latitude of the city of London is 51.5072.
Longitude: Specify the position of the sensor or thing that collects dat
degrees. For example, the longitude of the city of London is -0.1275.
Elevation: Specify the position of the sensor or thing that collects dat

Thingspeak

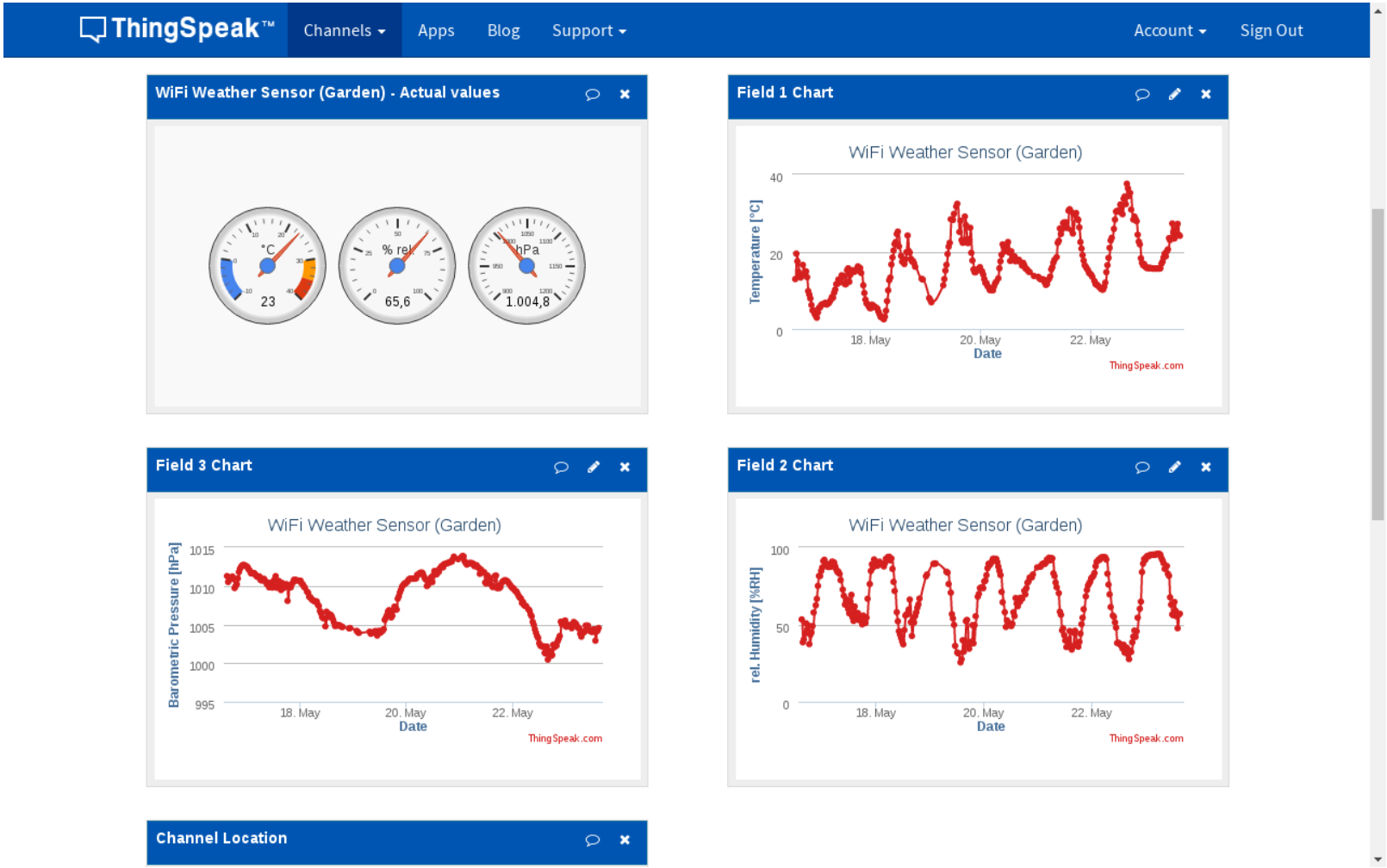

ThingSpeak™
Channels
Apps
Blog
Support
Account
Private View
Public View
Channel Settings
API Keys
Data Import / Export
Write API Key
Key
UJ4KBRFUYNV5X6C2
Generate New Write API Key
Read API Keys
Key
10NECCUF6JOVJBQ0
Note
Save Note
Delete API Key
Help
API keys enable you to write data to a channel or read data from a private
keys are auto-generated when you create a new channel.
API Keys Settings
Write API Key: Use this key to write data to a channel. If you feel you
been compromised, click Generate New Write API Key.
Read API Keys: Use this key to allow other people to view your priv
feeds and charts. Click Generate New Read API Key to generate an a
key for the channel.
Note: Use this field to enter information about channel read keys. F
add notes to keep track of users with access to your channel.
Create a Channel
POST https://api.thingspeak.com/channels.json
api_key=
name=My New Channel
Update a Channel
PUT https://api.thingspeak.com/channels/45287
api_key=
name=Updated Channel
Clear a Channel

Thingspeak

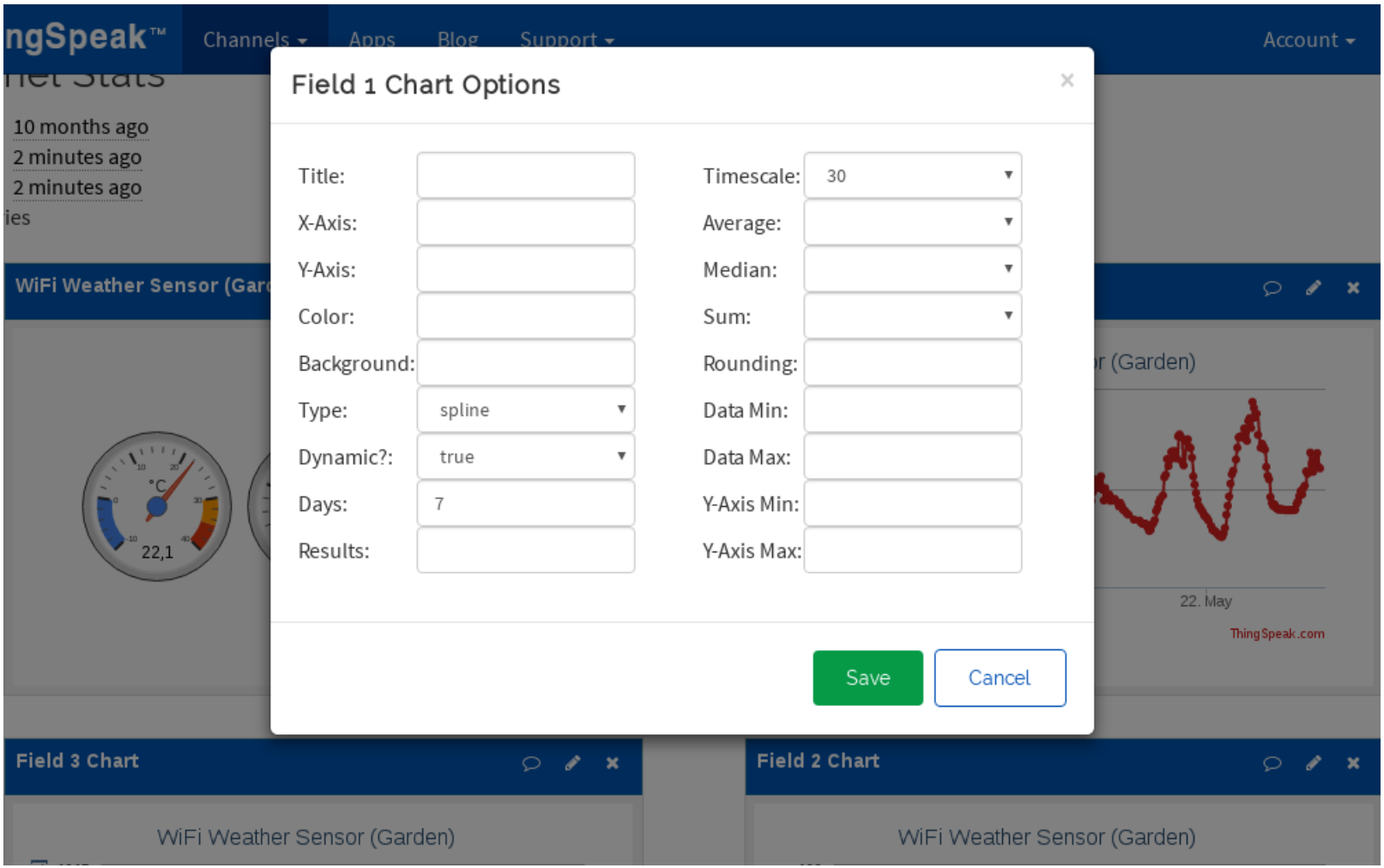
GET http://api.thingspeak.com/update?
api_key=UJ4KBRFUYNV5X6C2&
field1=25.1&
field2=52.8&
field3=1020.4&
field4=5.04



Thingspeak


ThingSpeak™
Channels
Apps
Blog
Support
Account
Sign Out
WiFi Weather Sensor (Garden) - Actual values
°C
23
% rel.
65,6
hPa
1.004,8
Field 1 Chart
WiFi Weather Sensor (Garden)
Temperature [°C]
40
20
0
18. May
20. May
22. May
Date
ThingSpeak.com
Field 3 Chart
WiFi Weather Sensor (Garden)
Barometric Pressure [hPa]
1015
1010
1005
1000
995
18. May
20. May
22. May
Date
ThingSpeak.com
Field 2 Chart
WiFi Weather Sensor (Garden)
rel. Humidity [%RH]
100
50
0
18. May
20. May
22. May
Date
ThingSpeak.com
Channel Location

Thingspeak


ngSpeak™
Channels
Apps
Blog
Support
Account
10 months ago
2 minutes ago
2 minutes ago
WiFi Weather Sensor (Gar
Field 1 Chart Options
Title:
X-Axis:
Y-Axis:
Color:
Background:
Type:
spline
Dynamic?:
true
Days:
7
Results:
Timescale:
30
Average:
Median:
Sum:
Rounding:
Data Min:
Data Max:
Y-Axis Min:
Y-Axis Max:
Save
Cancel
22,1
22. May
ThingSpeak.com
Field 3 Chart
WiFi Weather Sensor (Garden)
Field 2 Chart
WiFi Weather Sensor (Garden)

Risiken

- Unverschlüsselte Übertragung
  - Zu wenig Speicherplatz für SSL
- Abhören durch Dritte
  - Innentemperatur - jemand zuhause?
- Manipulation durch Angreifer
  - Wenn mit Daten etwas gesteuert wird
- Abgreifen von Massendaten
  - Angriff auf Clouddienst lohnt

Risiken

- Lösung für kritische Daten: eigener Server
- Lokal ohne aufwändige Sicherheitsmaßnahmen
  - Raspberry Pi
- Im Rechenzentrum per VPN Tunnel
  - Preiswerter V-Server
- Was aus dem Internet erreichbar ist - immer absichern!

MQTT

- Message Queue Telemetry Transport
- Nachrichtenprotokoll für Machine-to-Machine-Kommunikation
- Seit 2013 standardisiert für das Internet der Dinge
- TCP Port 1883
- TLS Verschlüsselung möglich

MQTT

Experiment mit mosquitto Server

Subscribe:

mosquitto_sub -h weather.poeggel.de -p 1883 -v -t /sensors/#

Publish:

mosquitto_pub -h weather.poeggel.de -p 1883 -t /sensors/development/temp -m "13.37"

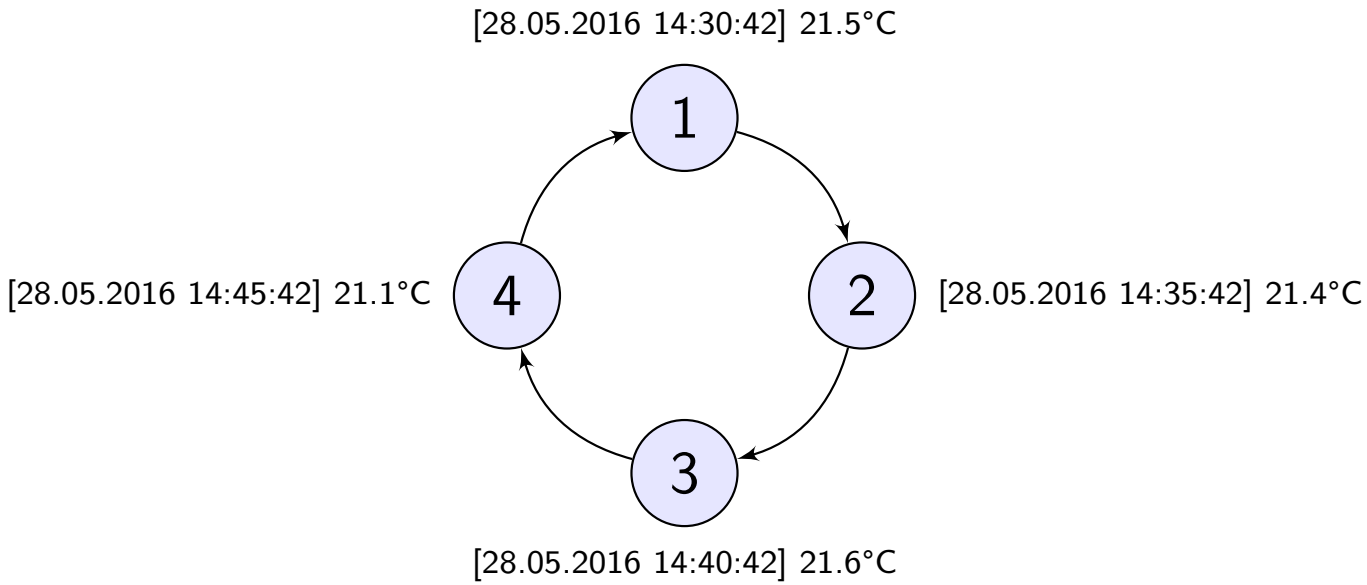
Ausgabe mosquitto_sub:

/sensors/development/temp 13.37

Die Daten werden von beliebig vielen Subscribern empfangen!

RRDTool

- RRDTool - Round Robin Database
- Feste Anzahl von Datensätzen
- Mehr Messwerte - letzter wird überschrieben



RRDTool

- Sind alte Daten verloren?
- Mehrere Archive (RRA) erstellen!
  - Unterschiedliche Zeitspanne
  - Daten werden über längere Zeit zusammengefasst
  - Einzelne Messwerte über längere Zeit nicht relevant
  - Geringer Speicherplatzbedarf

Datenbankanbindung

- collectd
  - Umfangreich, MQTT in Entwicklerversion
  - https://collectd.org/
- MQTT2RRD
  - Python, eine DB pro Wert
  - https://github.com/irvined1982/MQTT2RRD
- MQTT2RRD
  - C, Cache und eine DB für alles
  - https://github.com/MalteP/mqtt2rrd

Beispielkonfiguration

Datenbank erstellen:

```
rrdtool create espweather.rrd –start now –step 300 \
DS:temp0:GAUGE:600:-20:85 \
DS:hum0:GAUGE:600:0:100 \
DS:pres0:GAUGE:600:800:1200 \
DS:bat0:GAUGE:600:0:9 \
DS:temp1:GAUGE:600:-20:85 \
DS:hum1:GAUGE:600:0:100 \
DS:pres1:GAUGE:600:800:1200 \
DS:bat1:GAUGE:600:0:9 \
RRA:AVERAGE:0.5:1:8640
```

Mit Step der Intervall der Datenpunkte angegeben, hier 300 Sekunden. DS definiert einen Datenpunkt mit entsprechendem Namen vom Typ GAUGE (einfacher Zahlenwert). Es folgen Minimal- und Maximalwert.

Beispielkonfiguration

Mit dem Argument

RRA:AVERAGE:0.5:1:8640

wird ein Archiv für 8640 Messwerte im festgelegten Interval von 5 Minuten angelegt.

Dies entspricht einem Monat Daten, denn $\frac{60\text{min}\cdot 24\text{h}\cdot 30\text{d}}{5\text{min}} = 8640$

Soll darüber hinaus gespeichert werden, werden einfach weitere RRA angelegt.

Beispielkonfiguration

RRDtool aktzeptiert nur Aktualisierungen im Sekundentakt, Messwerte werden daher gesammelt und verzögert geschrieben. Die Konfiguration des MQTT2RRD Clients sieht wie folgt aus:

```
### Basic settings
mqtt_host = "localhost";
mqtt_port = 1883;
mqtt_user = "";
mqtt_pass = "";
client_id = "mqtt2rrd";
topic = "/sensors/#";

### Database settings
database = "espweather.rrd";
update_delay = 5;
update_interval = 10;
```

Beispielkonfiguration

```
### Topic to RRD DS assignment
topics = (
{ topic = "/sensors/0/temperature"; ds = "temp0"; timevalid = 600; },
{ topic = "/sensors/0/humidity"; ds = "hum0"; timevalid = 600; },
{ topic = "/sensors/0/pressure"; ds = "pres0"; timevalid = 600; },
{ topic = "/sensors/0/battery"; ds = "bat0"; timevalid = 600; },
{ topic = "/sensors/1/temperature"; ds = "temp1"; timevalid = 600; },
{ topic = "/sensors/1/humidity"; ds = "hum1"; timevalid = 600; },
{ topic = "/sensors/1/pressure"; ds = "pres1"; timevalid = 600; },
{ topic = "/sensors/1/battery"; ds = "bat1"; timevalid = 600; }
);
```

Damit sind die MQTT Topics ihren jeweiligen Spalten in der Datenbank zugeordnet.

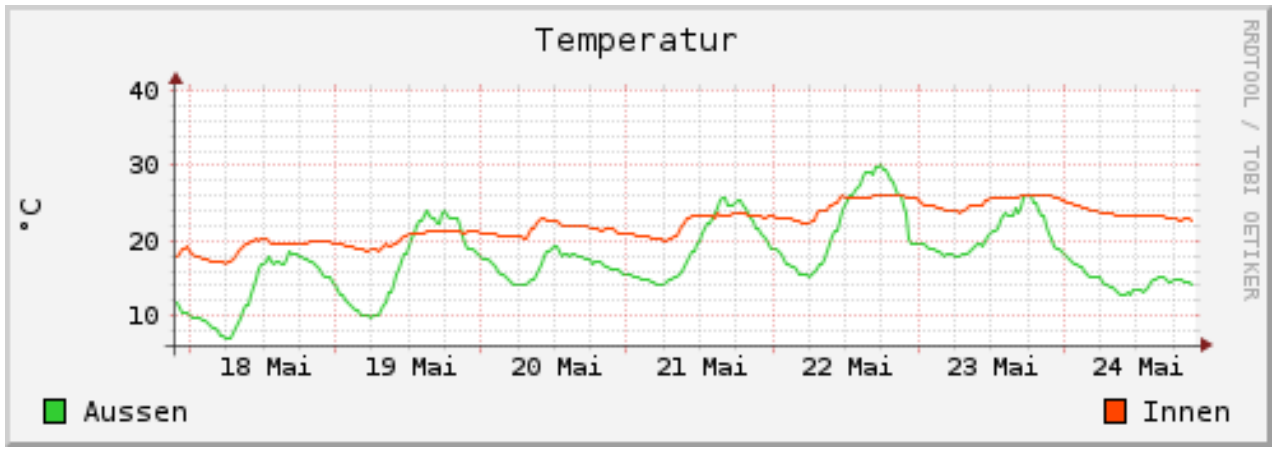
Beispielkonfiguration

Sind Sensoren und MQTT Server sind konfiguriert, der MQTT2RRD Client gestartet werden die nacheinander empfangenen Werte zwischengespeichert und nach 5 Sekunden geschrieben. Dies geschieht maximal alle 10 Sekunden. Der Client handhabt die Timeouts alleine und aktualisiert immer alle gültigen Werte zusammen.

Beispielkonfiguration

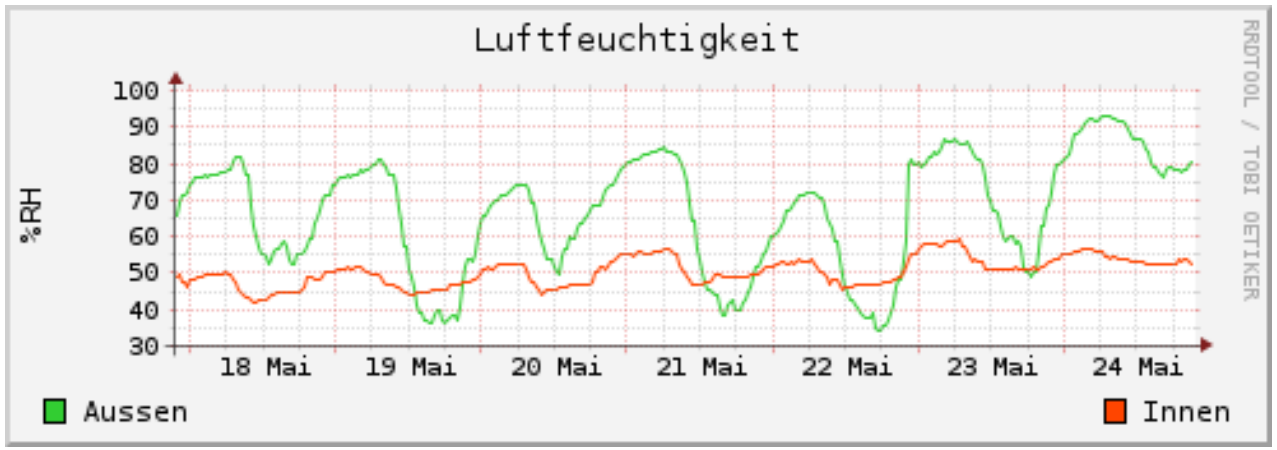
Mit RRDGraph können sehr einfach Grafiken aus der RRD Datenbank erstellt werden. Dies kann per Hand oder automatisch in einem Cronjob geschehen.

```
rrdtool graph temp.png -e now -s end-7d -i -E -v "°C" -t "Temperatur" \
DEF:var0=espweather.rrd:temp0:AVERAGE \
LINE1:var0#32cd32:"Aussen" \
DEF:var1=espweather.rrd:temp1:AVERAGE \
LINE1:var1#ff4500:"Innen"
```



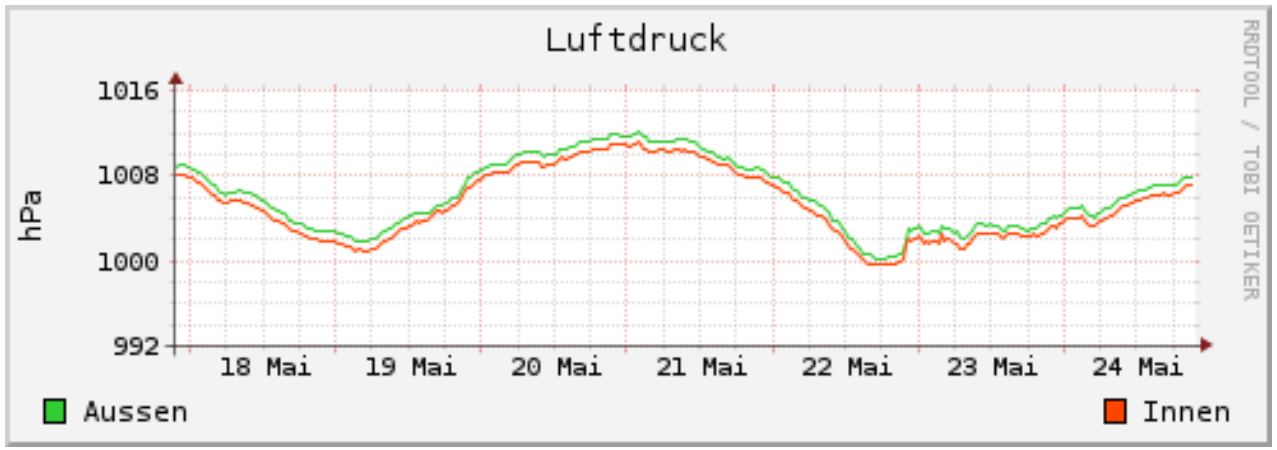
Beispielkonfiguration

```
rrdtool graph hum.png -e now -s end-7d -i -E -v "%RH" -t "Luftfeuchtigkeit" \
DEF:var0=espweather.rrd:hum0:AVERAGE LINE1:var0#32cd32:"Aussen" \
DEF:var1=espweather.rrd:hum1:AVERAGE LINE1:var1#ff4500:"Innen"
```



Beispielkonfiguration

```
rrdtool graph pres.png -e now -s end-7d -i -E -A -y 2:4 -X 0 -v "hPa" -t "Luftdruck" \
DEF:var0=espweather.rrd:pres0:AVERAGE LINE1:var0#32cd32:"Aussen" \
DEF:var1=espweather.rrd:pres1:AVERAGE LINE1:var1#ff4500:"Innen"
```



Fragen?

Weitere Infos unter:

http://www.maltepoeggel.de/

https://github.com/MalteP/espweather