Table of Contents

1. IoT Tools 2
2. MQTT Broker 2
2.1. MQTT broker on Raspberry Pi 2
2.1.1. Securing MQTT communication using credentials 4
2.1.2. Mosquitto server autostart 4
2.2. MQTT client 6
2.2.1. Connecting to the MQTT broker 6
2.2.2. Subscribing and Publishing 9
3. ESP8266 01 12
4. Easycam EC-230D camera 13
5. Laboratory ideas 14
5.1. Network enabled power monitor 14
5.2. A sensor network for the III level hall of the Faculty, mesh-based 14
5.3. Web-enabled coffee machine 14
5.4. A WiFi interface to the ToLC equipment 14
5.5. A LoRA based logical gateway 15
5.6. A mobile station for measuring PM10 and PM2.5 levels 15
5.7. A stationery solution for measuring and reporting PM10 and PM 2.5 levels 15

1. IoT Tools

Internet of Things laboratory project infrastructure

2. MQTT Broker

There are various possibilities on how to create an MQTT broker. It may be implemented as PC software (Linux, Windows-based, virtual or physical machine), however naturally we're looking for IoT device, capable of hosting MQTT broker. Here comes a solution based on Raspberry Pi (or clones). Note, we're going to install Mosquitto MQTT broker.

We assume, your RPi installation is installed, updated and the network interface is up, connected to the local or Internet, and there is no firewall.



Security note: once you run mosquitto for the first time, note it is not password protected by default. It is not the serious risk, however, other than massive/flood use of MQTT brokering. We suggest not to keep unsecured MQTT broker in the public network for a long time (other than testing).

2.1. MQTT broker on Raspberry Pi

Installing an MQTT broker on Raspberry Pi contains some tricks regarding current Armbian OS distribution.



Please note, this instruction does not transfer straightforward to the RPi clones (i.e. Orange Pi, Banana Pi, etc.) because they host different processors thus underlying binary libraries are different (i.e. other than armhf).

The instruction below applies to Raspian version Stretch, on Raspberry Pi 3 B. The original Armbian distribution comes with outdated Mosquitto libraries. Depending on the current state of updating the distro, it may be necessary to remove any *mosquitto* repositories before continuing this instruction. Please refer to the // /etc/apt/sources.list.d/ // folder and find any currently installed mosquitto packages:

```
cd /etc/apt/sources.list.d/
ls *mosquitto*
```

In case above code returns any source, remove it:

```
rm <here comes the name>.list
```

Next follow the steps below to install Mosquitto for Stretch:

```
cd ~/Downloads
wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/
sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
sudo apt-get update

cd ~/Downloads
wget
http://security.debian.org/debian-security/pool/updates/main/o/openssl/libss
l1.0.0_1.0.1t-1+deb8u6_armhf.deb
sudo dpkg -i libssl1.0.0_1.0.1t-1+deb8u6_armhf.deb
wget
http://ftp.nz.debian.org/debian/pool/main/libw/libwebsockets/libwebsockets3_
1.2.2-1_armhf.deb
sudo dpkg -i libwebsockets3_1.2.2-1_armhf.deb
sudo apt-get install mosquitto mosquitto-clients
```

Once the installation is finished test, if mosquitto works:

```
mosquitto
```

The server should start, present network configuration and also print messages on client connection (if any):

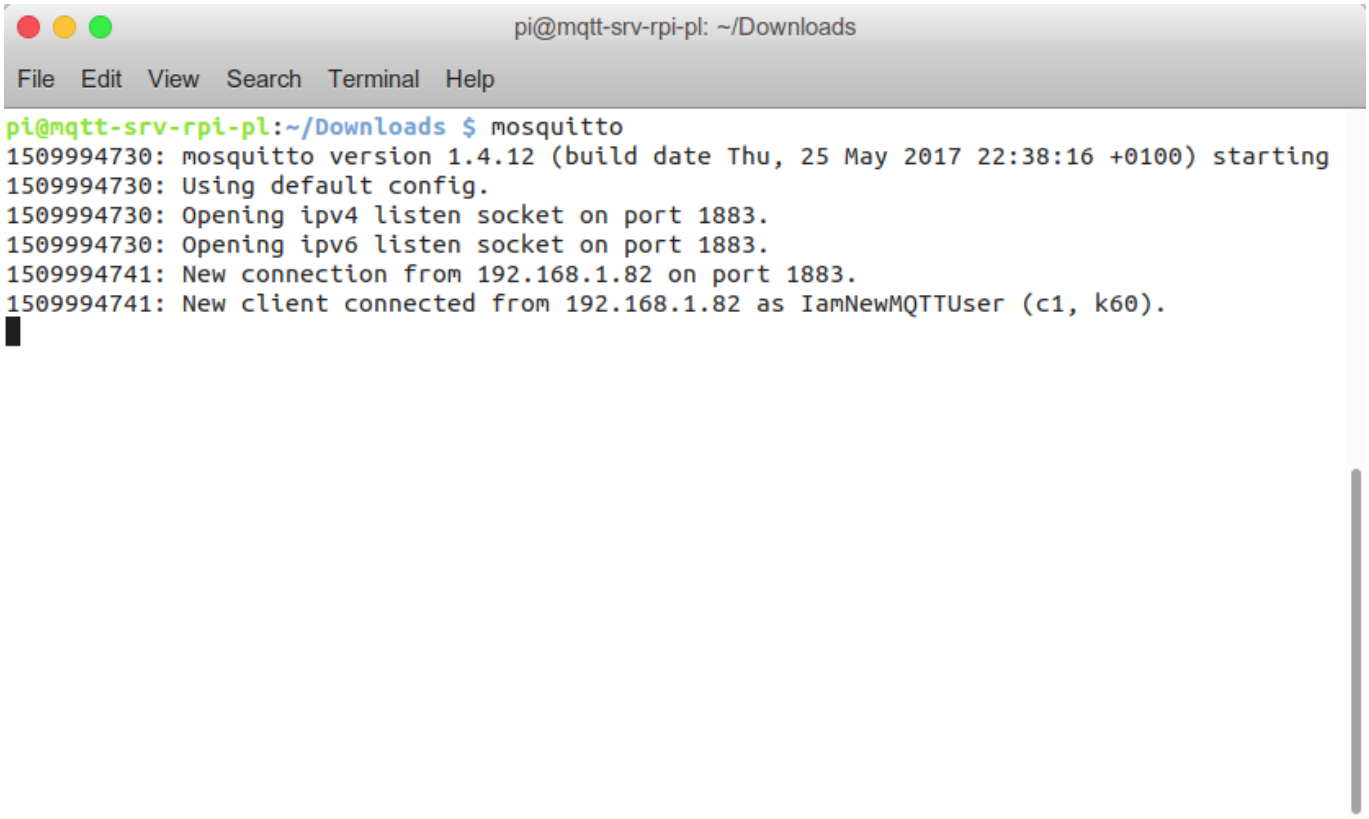


Fig. 1: MQTT - first run

The Mosquitto server runs in the foreground now and its log is redirected to the console. It runs without any configuration loaded. You can connect to it anyway, for testing purposes. Any MQTT client (including installed command-line ones should work). Refer to the section below on **MQTT client**.

To kill server use Ctrl+C.

2.1.1. Securing MQTT communication using credentials

Once Mosquitto server is in the public network, virtually anyone can connect to it as a client. This is not desired state thus it is essential to provide some security.



Note - this is very weak securing yet, it disables public use of your server that is a goal here. Note - user and password are transmitted plain text so you should ensure a secure channel.

To enable credentials there is a need to configure */etc/mosquitto/mosquitto.conf* file, along with providing a link to the file containing users and passwords (`password_file /etc/mosquitto/pwfile`) and to disable anonymous access (`allow_anonymous false`):

```
listener 1883
persistence true
persistence_location /var/lib/mosquitto/
persistence_file mosquitto.db
log_dest syslog
log_dest stdout
log_dest topic
log_type error
log_type warning
log_type notice
log_type information
connection_messages true
log_timestamp true
allow_anonymous false
password_file /etc/mosquitto/pwfile
```

To create a hashed password file use *mosquitto_passwd*. Sample command adding an user named **iot** and providing it a password is presented below:

```
sudo mosquitto_passwd -c /etc/mosquitto/pwfile iot
```

Enter password for user **iot** twice.



Note `-c` command creates a new password file (old one is deleted, if exists).

To append next user to the existing file, omit `-c` in the command above.

2.1.2. Mosquitto server autostart

Now it is time to make mosquitto server start automatically on Raspbian boot. Unfortunately, modern

Mosquittos' installer (here version 1.4.12) provides incompatible settings for autostart in case of Raspbian Stretch. You need to configure service yourself. While it may sound bad, it is not so difficult indeed.
Time to create a new service, through configuration file for Mosquitto. File is: /etc/systemd/system/mosquitto.service . Note - if this file exists, that perhaps means your installation package correctly handled new autostart mechanisms, thus Mosquitto may be already enabled to autostart. To check if your mosquitto is already starting automatically, reboot the system, then check if your Mosquitto is listening:

```
sudo reboot
```

and once your RPi restarts, re-connect terminal and issue:

```
netstat -l -p |grep 1883
```

Your RPi should answer something like this:



Fig. 2: Mosquitto - check server is running

If you see the server ready and listening, do not follow this manual - it is in case your server does not start automatically on boot-up.

 Follow below only if your server does not boot automatically

First remove old configuration (if any):

```
sudo update-rc.d mosquitto remove
```

next, create new /etc/systemd/system/mosquitto.service file:

```
sudo nano /etc/systemd/system/mosquitto.service
```

and fill it with following content:

```
[Unit]
Description=Mosquitto MQTT Broker daemon
ConditionPathExists=/etc/mosquitto/mosquitto.conf
Requires=network.target

[Service]
Type=simple
ExecStart=/usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf -d
ExecReload=/bin/kill -HUP $MAINPID
PIDFile=/run/mosquitto.pid
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Note - the configuration file for Mosquitto server is located in */etc/mosquitto/mosquitto.conf* (default, as configured in */etc/systemd/system/mosquitto.service* file) and the service, while working, inserts a process identifier (PID) into the file: */run/mosquitto.pid* (defaults, reconfigurable in */etc/mosquitto/mosquitto.conf*).

Now enable the service and reboot to test it working:

```
sudo systemctl enable mosquitto.service
sudo reboot
```

2.2. MQTT client

There are various MQTT clients. One of the most convenient yet offering full features and great GUI and also the one that does not require any installation is Java-based MQTT-SPY. Simply download it, and run.

2.2.1. Connecting to the MQTT broker

Once the main screen appears, please add new connection:

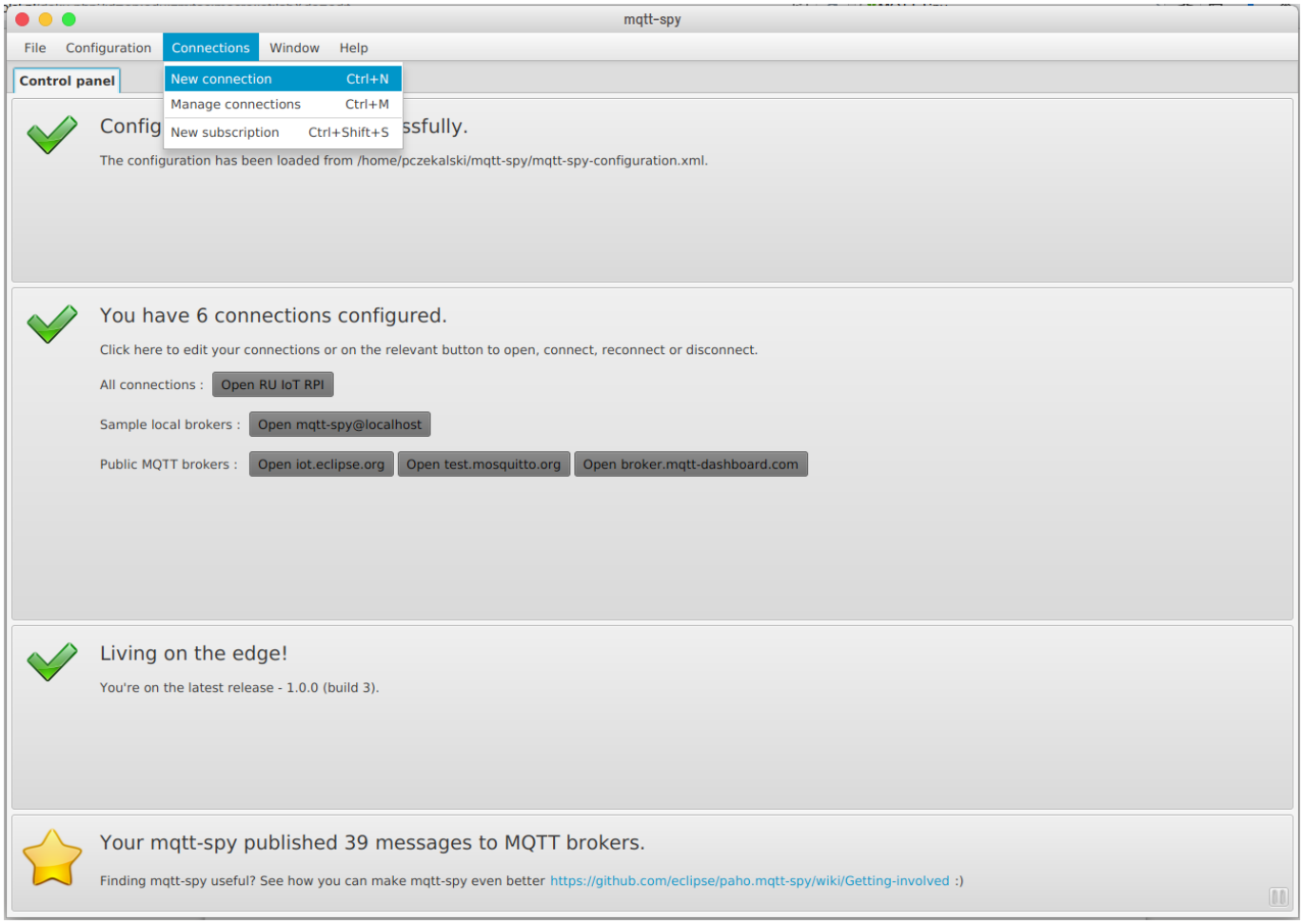


Fig. 3: MQTT-Spy

Next provide connection details (currently your MQTT server does not need any credentials, it simply accepts anonymous connections from anyone):

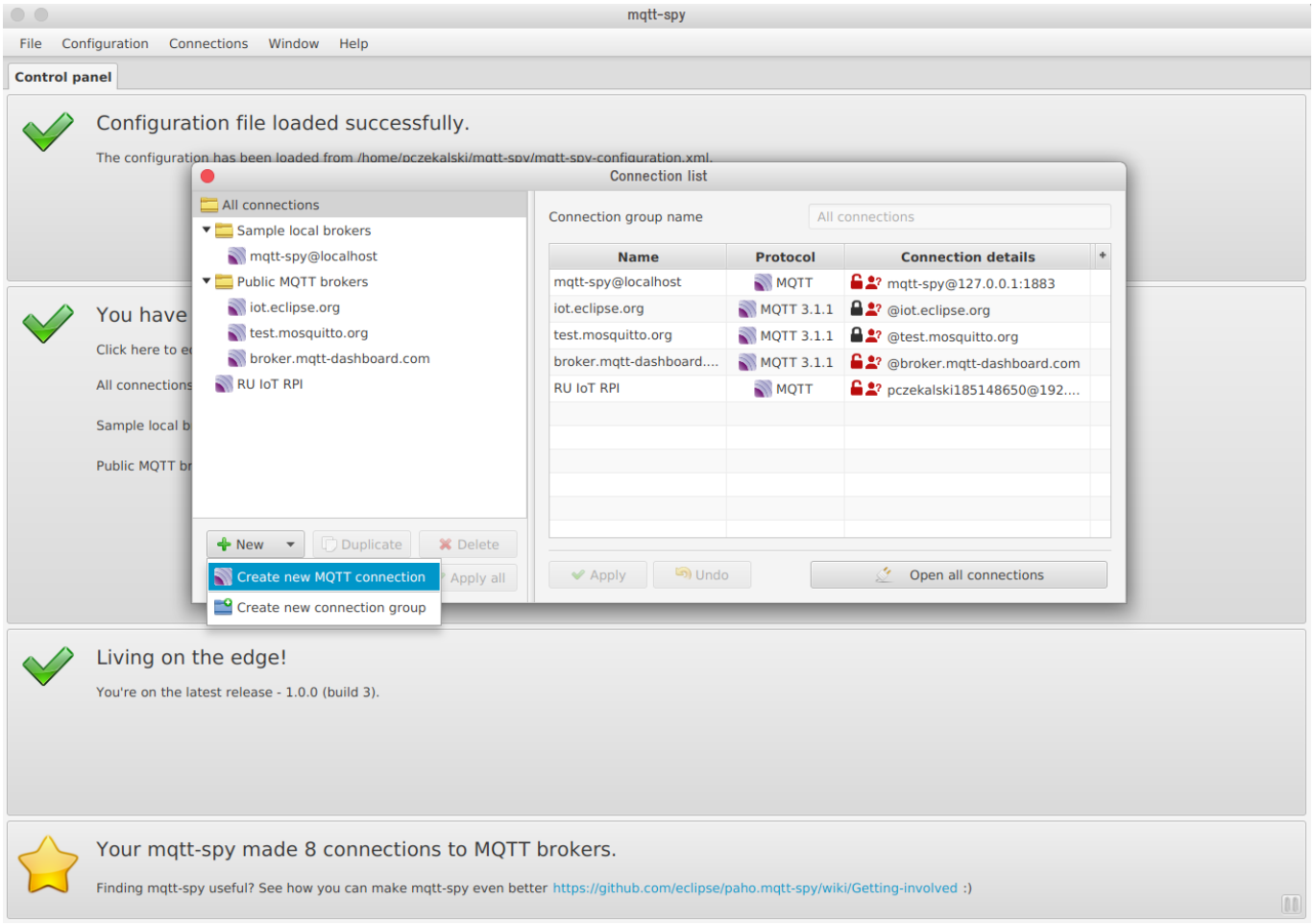


Fig. 4: MQTT-Spy - connection manager

Give it some name, client ID and IP address of your RPi (if you're running on the same machine, use 127.0.0.1 - IP loopback).

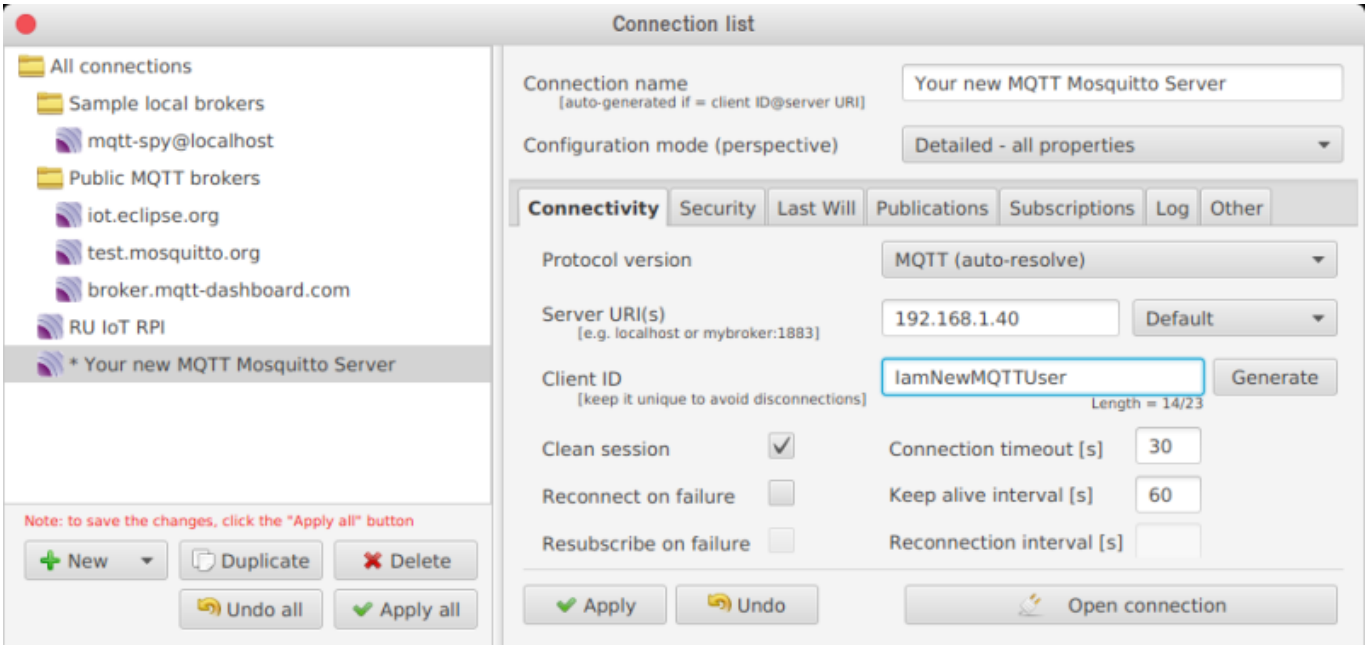


Fig. 5: MQTT-Spy - new connection to your broker

Now click [Open connection]. MQTT-Spy may ask you to save the connection. This may fail if you run MQTT-Spy client for the first time - you need to create default configuration file first. Nevertheless you should see the following window with green coloured tab (if succesfully connected):

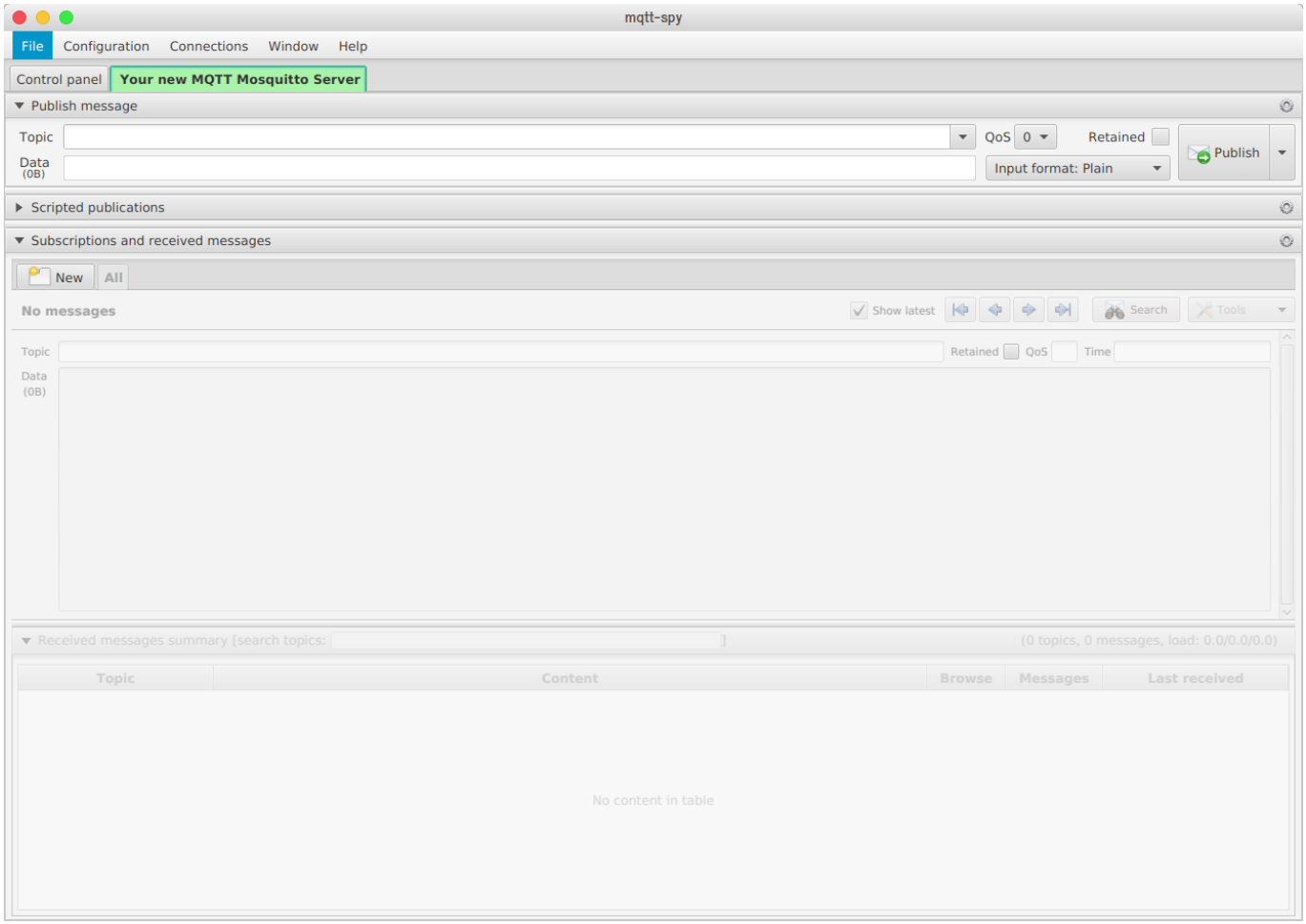


Fig. 6: MQTT-Spy - client connected

In case connection fails, first check if both (client and server) can connect on port 1883 and validate IP addresses.

2.2.2. Subscribing and Publishing

Now it is time to test MQTT messages forwarding.
We will simulate it using one client both for publishing and subscribe.
First, it is time to subscribe. Click New tab in the “Subscriptions and received messages” section then enter topis you want to subscribe to then click [Subscribe]. Assuming you're the only one using the server, you may subscribe to all communicates using wildcard # (hash), as below:

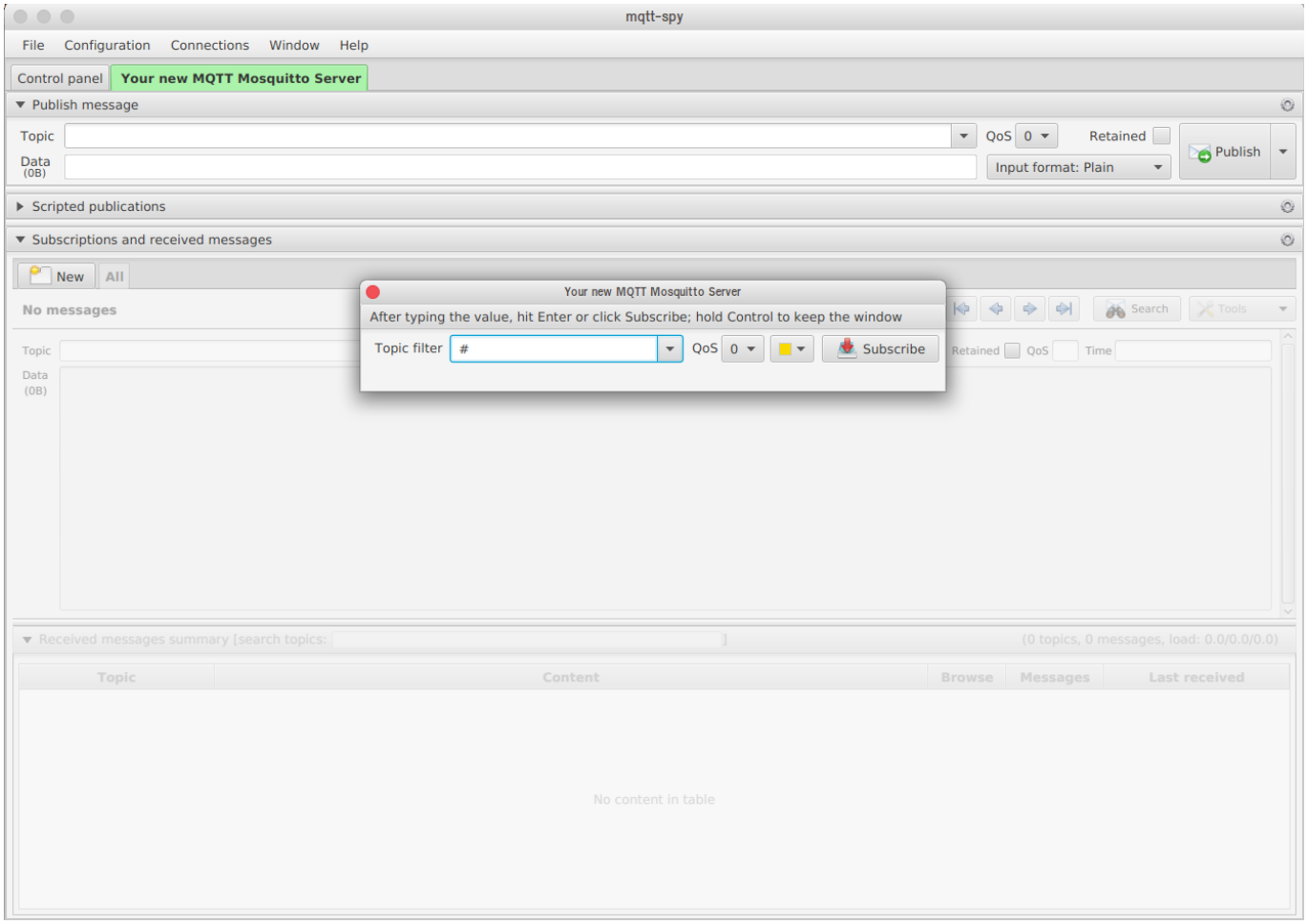


Fig. 7: MQTT-Spy - subscribe to any (all) messages

Once done, click yellow # tab to observe incoming communicates.
Next step is to publish any topic to ... yourself (actually that sounds odd, anyway remember it is for testing purposes - in a real environment it is rare Published and Subscriber of the particular topic are the same system). To publish an MQTT message, write in the “Publish Message” section sample Topic and Payload and click [Publish]:

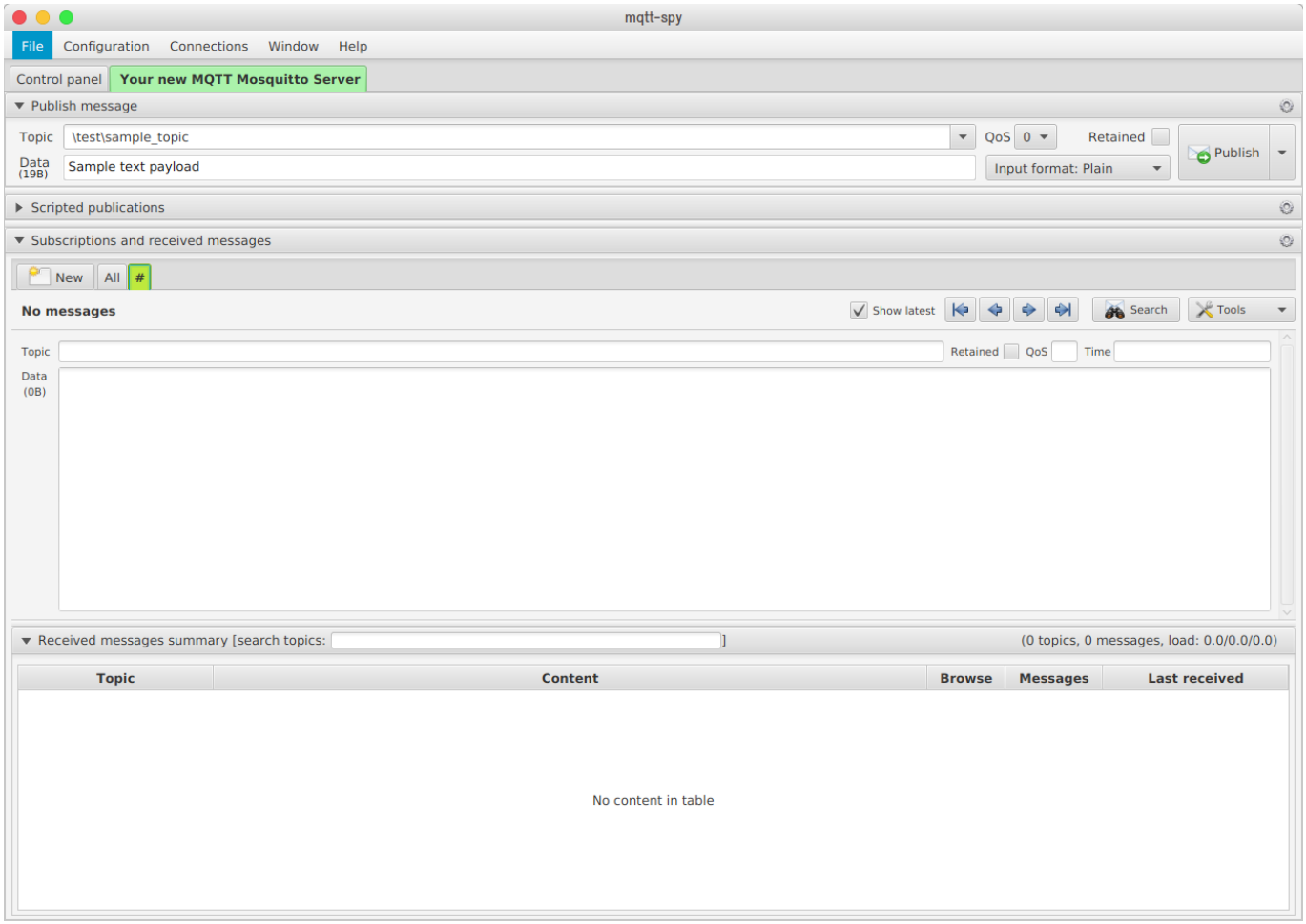


Fig. 8: MQTT-Spy - Publishing a message

Right afterward, you should see the forwarded message through the MQTT broker to appear in the “Subscriptions and received messages” section, presenting Topic and Payload you entered during sending, as below:

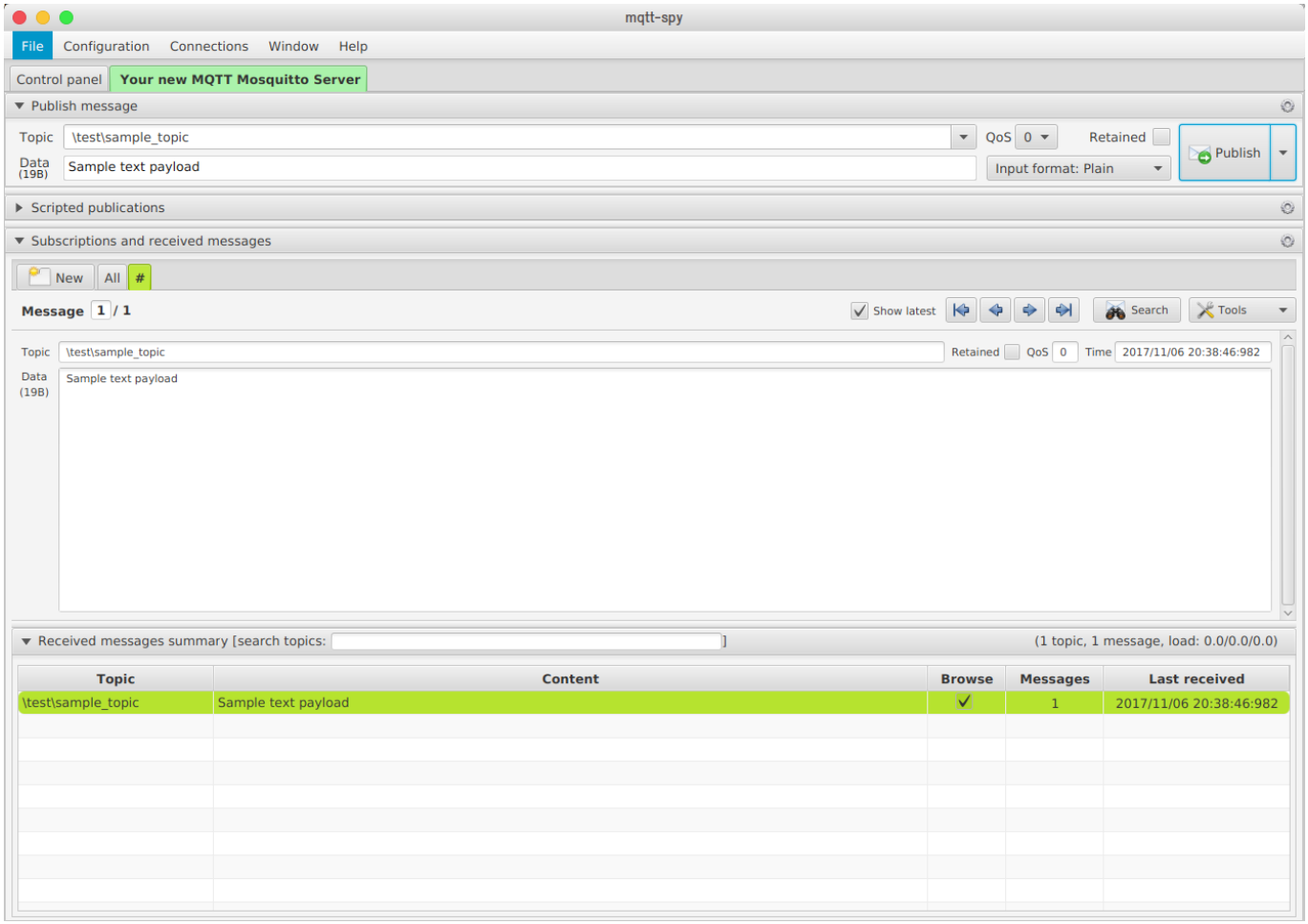


Fig. 9: MQTT-Spy - Message received

3. ESP8266 01

Programming (flashing) ESP8266 01 dongle.

There are 2 versions of the ESP01 (ESP-01 / ESP9266 01) dongle: 512kB and 1MB (8Mbit) If you have 512kB version, do not even think about programming it - most of the firmware available is already for at least 1MB flash size.If your memory is bigger, then you need to adjust addresses where appropriate section of the flash memory starts (see Flash Memory Map, note it is not the same as Memory Map).

The code below shows some flashing scenarios (Ubuntu 17.10 x64, python 2.7).
Erase contents:

```
pczekalski@s317-pc01:~$ sudo ~/.local/lib/python2.7/site-packages/esptool.py
--chip esp8266 --port /dev/ttyUSB0 erase_flash
```

Flash original firmware (Esprif):

```
pczekalski@s317-pc01:~$ pczekalski@s317-pc01:~$ sudo
~/.local/lib/python2.7/site-packages/esptool.py --chip esp8266 --port
/dev/ttyUSB0 write_flash --flash_freq 20m --flash_size detect --flash_mode
dio 0x0000 ~/Downloads/ESP8266_NONOS_SDK-2.1.0/bin/boot_v1.7.bin 0x01000
~/Downloads/ESP8266_NONOS_SDK-2.1.0/bin/at/512+512/user1.1024.new.2.bin
0x3FC000 ~/Downloads/ESP8266_NONOS_SDK-2.1.0/bin/esp_init_data_default.bin
```

```
0x7e000 ~/Downloads/ESP8266_NONOS_SDK-2.1.0/bin/blank.bin 0x3fe000
~/Downloads/ESP8266_NONOS_SDK-2.1.0/bin/blank.bin
```

Write firmware for router:

```
sudo ~/.local/lib/python2.7/site-packages/esptool.py --chip esp8266 --port
/dev/ttyUSB0 write_flash --flash_freq 20m --flash_size detect --flash_mode
dio 0x0000 ~/Documents/Git/esp_wifi_repeater/firmware/0x00000.bin 0x10000
~/Documents/Git/esp_wifi_repeater/firmware/0x10000.bin
```

4. Easycam EC-230D camera

Live stream is available at:

```
rtsp://IP:554/user=admin&password=password&channel=1&stream=0.sdp?real_strae
am--rtp-caching=100
```

Only Internet Explorer works. Camera requires external converter from the stream above, to the http-based one, i.e. Motion server.

Stream parameters are:

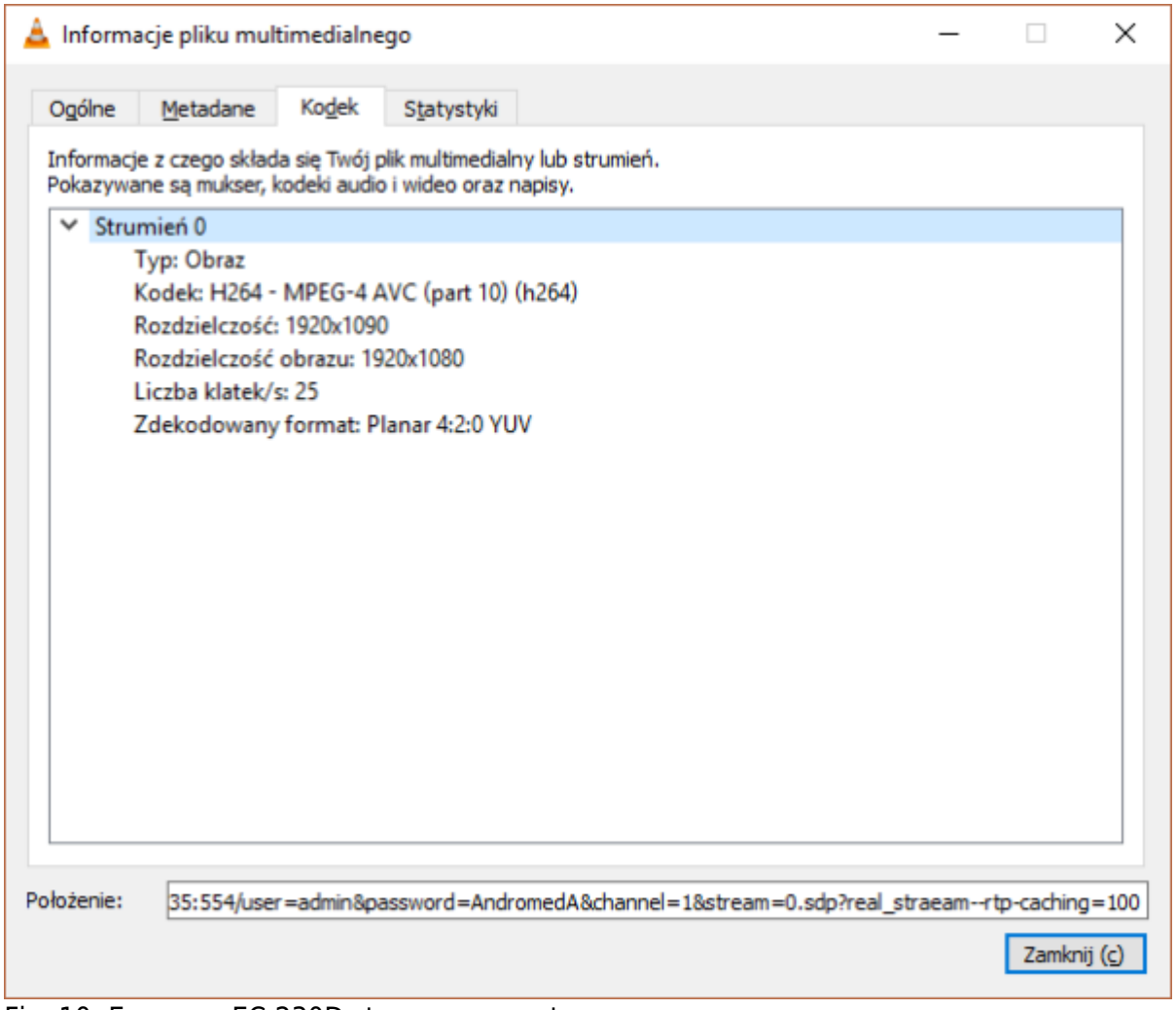


Fig. 10: Easycam EC-230D stream parameters

5. Laboratory ideas

5.1. Network enabled power monitor

#smartmetering Create a network-enabled system that monitors high voltage and current for the home system.
Hardware is ESP32 with WiFi connection. It should display it locally on the OLED display (an integrated ESP32 board) and should report it to the MQTT server.
Configuration should enable following features:

- Individual identification of the devices via some identifier in the flash memory.
- Remote configuration via AP and via connected wireless interface:
  - At the beginning system should expose access point.
  - Once configured and successfully connected to the infrastructure, management interface should be available via the infrastructure address.
  - If device cannot connect to the infrastructure, it should present the AP again (timeout to be provided)
- There should be a button to reset configuration and provide clean startup
- Configuration should provide MQTT client config (MQTT server address, credentials if any and message string and period).
- Summary power consumption should be present and stored in persistent memory periodically (to be adjusted)

5.2. A sensor network for the III level hall of the Faculty, mesh-based

#sensornetwork\\
A sensor network (temperature / humidity DHT22) of the ESP8266-01 devices, battery operated, identifiable, constituting a mesh network. Each device shall have reporting capability. The system should also consist of middleware (MQTT + NodeRED + database + web server) to store history and present data to the end user over the web (as a whole network, not a separate device). Note - those devices are to be battery powered thus power usage optimisation is a must!

5.3. Web-enabled coffee machine

#smarthouse\\
An interface for our office coffee machine to switch it on and off remotely via relays. Level monitoring of the Coffee (possibly optical) is also very welcome. An interface should send notifications to the users, i.e. via social media.

5.4. A WiFi interface to the ToLC equipment

#teaching #education\\
A module with a network interface (possibly ESP32 or WeMOS D1 R2 or WeMOS Mini D1) providing

digital outputs that you can individually set or reset via a WEB interface. Web Access via infrastructure or directly via AP (configuration of the infrastructure via AP).

5.5. A LoRA based logical gateway

#networking\\
A solution for a gateway collecting sensor data through WiFi and forwarding it for a long distance connection over LoRA networks.

5.6. A mobile station for measuring PM10 and PM2.5 levels

#environment\\
Battery operated mobile device presenting dust level PM2.5 and PM10 (possibly others, sensor dependent), with networking capability. The PMS sensors are to be used in this project (requires porting of the existing library from C# to the chosen language.

5.7. A stationery solution for measuring and reporting PM10 and PM 2.5 levels

#environment\\
A sensor device for PM2.5 and PM10 (based on existing PMS sensor) with the capability to forward the measurements, store and present them in the cloud solution.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:macro:iot:project

Last update: **2019/03/07 11:08**