Course notes, module 8 week 12
Message passing design patterns

*Kjeld Jensen kjen@sdu.dk*

# Agenda

1. Mid term review conclusions
2. Recap of previous module and preparation for next module
3. Mini project description
4. Current module theory
5. Exercises
6. Optional exercises

# 1. Mid term review conclusions

First of all thank you very much for taking time to provide a good feedback. I am really happy about it, and I find it very useful. It will definitely help me making the rest of the course better and improve it for next years students.

**The good**
Learning goals fulfilled ok
Level of difficulty is rated suitable
Teaching methods ok
LEO1 kits ok
Good atmosphere
Optional exercises are good

**Room for reflections and improvements** (we will discuss in class)
There is a clear request from many of you on more theoretical material in the form of slides, reading materials, references etc. This also links to some confusion about the exam.

Many of you like the level of the help/support you get in the weekly notes to complete exercises, however there is also a desire for a more thorough understanding of what happens and some more room for experimenting.

Some of you are having difficulties with the electronics.

Some of you suggests weekly small report.

**Questions for the class**

If you were to choose another form of examination, what would you prefer? This can only be for next years students though.

Is there a challenge with LEO1 overlapping with other courses?

LEO2 "linux project" will likely not be offered in the next semester, but would you choose it if you had the chance?

## 2. Recap of previous module and preparation for next module

In the previous module we worked with linux security with regards to remote login using ssh.

In the next module we will continue working on message passing design patterns and the project description.

No preparations is needed for next module beyond conducting the exercises, however please see section 3 below.

## 3. Mini project description

At itslearning there is now a Plan named "Mini project". It contains the mini project information document that we reviewed in module 7 and now also an example of a mini project description.

By Thursday 31/3 at 12.00 you will need to prepare a draft of your mini project description and submit it via itslearning.

Be as concrete as you can, but it is ok to use bullets and keywords. This is a draft iteration and you will receive feedback from me as part of the next module. There will be one more iteration after this submission, before the mini project begins.

## 4. Current module theory

In this module we will focis on on message passing design patterns and the project description.

## 5. Exercises

In module 6 we worked with HTTP POST requests as a means of message passing.  In this module we will similarly work with message passing, however this time we will use Message Queuing Telemetry Transport (MQTT) for message passing.

### 5.1 Configuring ESP8266 to send data as a MQTT client

In this exercise we will configure the Adafruit ESP8266 as a MQTT client sending the number of push button counts like we did using HTTP POST requests in module 6.

To be able to test up against a MQTT broker we will in this exercise use an external IoT platform. We could use ThingSpeak like we did last time, but to learn more about some of the different platforms available we will use Adafruit IO instead. First sign up for an account at:

https://accounts.adafruit.com/users/sign_up

Remember to verify your account via the email sent to your email address.

Now go to https://io.adafruit.com

Click on `My Key` to get the key for use with the Arduino Sketch

We will now create a feed that receives the data from our ESP8266. Click on `Feeds` and then on `view all`.

Click Create a new feed and enter `count` as the name.

To view the feed data we will configure a Dashboard. Click on `Dashboards` and then on `view all`.

Click `New Dashboard` and name it "Count dashboard"

Open the dashboard and choose `Create a new block` then choose `Stream`. Mark the count stream. Now Adafruit IO is ready to receive data from the ESP8266.

Using the Arduino IDE we will upload another firmware to the ESP8266. First we need to install the Adafruit MQTT library (only that library is needed).

Go to `Tools - Manage libraries...`

Install `Adafruit MQTT Library` (click no to installing other required libraries)

Then use the Arduino IDE to flash the Arduino Sketch `esp8266_count_mqtt.ino` onto the ESP8266. Please notice that you will have to update the Arduino Sketch with your (internet access) wifi settings and your Adafruit IO user name and key.

### 5.2 Installing a MQTT broker on the RPi

Now that we have the ESP8266 configured as a MQTT client and we have tested it agains Adafruit IO, it is time to configure the RPi as a MQTT broker. We will use the MQTT broker named Mosquitto. To install this run the following commands:

Get the repository GPG key:

```
$ wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
$ sudo apt-key add mosquitto-repo.gpg.key
```

The add the Mosquitto repository to the apt package system. Please notice that the below commands assume that you are using the Raspbian version 11 (Bullseye):

```
$ cd /etc/apt/sources.list.d
$ sudo wget http://repo.mosquitto.org/debian/mosquitto-bullseye.list
$ sudo apt-get update
```

Now install Mosquitto:

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

To start Mosquitto:

```
$ sudo systemctl enable mosquitto.service
```

To test Mosquitto open two RPi ssh shells and one the  mosquitto test clients. To start a subscriber run in the first window:

```
$ mosquitto_sub -h localhost -p 1883 -t my_topic -d
```

In the second window run a publisher:

```
$ mosquitto_pub -h localhost -p 1883 -t my_topic -m my_message
```

If you see the subscriber in the first window receive the message `my_message`, then Mosquitto has been installed successfully.

The default Mosquitto installation allows anonomous publishing and subscribing. This may be acceptable in some system but as a general rule you should configure Mosquitto to require a username and password. Even though the password is not encrypted during transmission it is better than not having one:

```
$ sudo nano /etc/mosquitto/mosquitto.conf
```

Remove this last line:

```
include_dir /etc/mosquitto/conf.d
```

And insert those lines:

```
allow_anonymous false
password_file /etc/mosquitto/pwfile
listener 1883
```

To add a user use the below command. You may replace `my_user` by your preferred username, just remember to replace `my_user` in subsequent commands as well:

```
$ sudo mosquitto_passwd -c /etc/mosquitto/pwfile my_user
```

You will be asked to enter a password for this user.

Now we need to restart the service:

```
$ sudo systemctl restart mosquitto.service
```

If you now repeat the publish-subscribe test above then you will see that it doesn't work anymore. Instead you get the message "`Connection error: Connection Refused: not authorised`". If you add the username and password to both commands, then it works again:

In the first window run a subscriber:

```
$ mosquitto_sub -d -h localhost -p 1883 -u my_user -P my_password -t my_topic
```

In the second window run a publisher:

```
$ mosquitto_pub -h localhost -p 1883 -u my_user -P my_password -t my_topic -m
my_message
```

### 5.3 Configuring the ESP8266 to send data to the RPi Mosquitto broker

Now we wish to have the ESP8266 send MQTT messages to the RPi rather than the Adafruit IO IoT platform.

To do this we need to change the configuration in the Arduino Sketch `esp8266_count_mqtt.ino` to:

```
#define WIFI_SSID         "LEO1_TEAM_XX"
#define WIFI_PASSWORD     "embeddedlinux"

#define MQTT_SERVER       "192.168.1.1"
#define MQTT_SERVERPORT   1883
#define MQTT_USERNAME     "my_user"
#define MQTT_KEY          "my_password"
#define MQTT_TOPIC        "/count"
```

Then flash the Arduino Sketch onto the ESP8266. Using the Arduino IDE Serial Monitor you can follow what happens on the ESP8266.

On the RPi you should now be able to see the incomming messages using this command:

```
$ mosquitto_sub -d -h localhost -u my_user -P my_password -t my_user/count
```

If you see a received PUBLISH message like the below every 15 seconds, then you have successfully configured your ESP8266 to send data to the RPi Mosquito broker.

```
Client (null) received PUBLISH (d0, q0, r0, m0, 'my_user/count', ... (1 bytes))
0
```

### 5.4 Creating a database for the ESP8266 data

We need a database to store data from the ESP8266 sent via MQTT messages. We will install influxdb[1] which is a time series database.

Add the influxdb key:

```
$ curl https://repos.influxdata.com/influxdb.key | gpg --dearmor | sudo tee
/usr/share/keyrings/influxdb-archive-keyring.gpg >/dev/null
```

Install the influxdb

```
$ echo "deb [signed-by=/usr/share/keyrings/influxdb-archive-keyring.gpg]
https://repos.influxdata.com/debian $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
```

```
$ sudo apt update
```

```
$ sudo apt install influxdb
```

Now we configure the influxdb to start at boot time:

1 https://www.influxdata.com/products/influxdb-overview/#

```
$ sudo systemctl unmask influxdb
$ sudo systemctl enable influxdb
$ sudo systemctl start influxdb
```

You can check if the installation wen't well by running the command:

```
$ systemctl status influxdb
```

We will create a simple database to store the data in. Run the command below to launch the InfluxDB shell:

```
$ influx
```

You will then see the InfluxDB shell prompt > Run the commands:

```
create database esp8266_count
use esp8266_count
create user telegraf with password 'leo1'
grant all on esp8266_count to telegraf
exit
```

In the next exercise we will be directing data from Mosquito to the database.

### 5.5 Saving the ESP8266 data received via MQTT in the database

We now need to save the data received from the ESP8266 into the database. For this we will use Telegraf[2] which is a server-based agent for collecting and sending all metrics and events from databases, systems, and IoT sensors.

Run these commands to install Telegraf:

```
$ sudo apt install telegraf
$ sudo systemctl enable telegraf
$ sudo systemctl start telegraf

$ sudo systemctl status telegraf
```

Now edit the configuration file to tell Telegraph to retrieve the data via MQTT and save in the database we just created.

```
$ sudo nano /etc/telegraf/telegraf.conf
```

Search the file for the section `inputs.mqtt_consumer` and make that section look like below:

```
[[inputs.mqtt_consumer]]
servers = ["tcp://localhost:1883"]
username = "my_user"
password = "my_password"
data_format = "value"
data_type = "integer"
```

2 https://www.influxdata.com/time-series-platform/telegraf/

```
topics = [
  "my_user/count"
]
```

Then search the file for the section `outputs.influxdb` and make that section look like below:

```
[[outputs.influxdb]]
   urls = ["http://127.0.0.1:8086"]
   database = "esp8266_count"
   username = "telegraf"
   password = "leo1"
```

Telegraf is now configured to store data from MQTT into the InfluxDB database. We just need to restart Telegraf.

```
$ sudo systemctl reload telegraf
```

To verify that it works you can logon to the InfluxDB shell and monitor the database:

```
$ influx

select * from esp8266_count
```

### 5.6 Viewing ESP8266 data on a graphical dashboard

We will install Grafana which is a tool for creating graphical dashboards.

Add the Grafana key:

```
$ wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key
add -
```

Install Grafana:

```
$ echo "deb https://packages.grafana.com/oss/deb stable main" |
sudo tee -a /etc/apt/sources.list.d/grafana.list

$ sudo apt-get update
$ sudo apt-get install grafana
```

Start Grafana

```
$ sudo systemctl enable grafana-server
$ sudo systemctl start grafana-server
```

Grafana now runs on the RPi and you can log in via your browser:

```
http://10.0.0.10:3000
```

First time log in to Grafana with the default username `admin`, and the default password `admin` You will then be asked to change the password. For simplicity you could use: `leo1_admin` as password.

Then you first need to add the InfluxDB database as a data source:

Click Data sources

Then click InfluxDB

Under HTTP configure the URL as:

`URL: http://localhost:8086` (it is already listed in grey text but you have to write it)

Under InfluxDB Details enter your database info:

```
Database: esp8266_count
User: telegraf
Password: leo1
```

Click Save & test

New we will create our first Dashboard. Click the Grafana logo at the top left of the page to get back to the main page.

Click New Dashboard

Click Add a new panel

Under Data source select InfluxDB

In line `From` change `select measurement` to `mqtt_consumer`.

You should now see a graph showing a time series of the number of button clicks you have made since the ESP8266 started sending MQTT meassages to the InfluxDB

Click the Apply button to save the Dashboard.

You have now successfully established the full path from the ESP8266 pushbutton to visualizing the number of times the button is pushed in Grafana.

# 6. Optional exercises

### 6.1 MQTT efficiency metrics

MQTT is marketed[3] as "is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth."

Now the question begs, how much more efficient than HTTP POST requests?

In this exercise you should study this. In module 6 you created a similar setup with HTTP POST requests from the ESP8266. We will now compare the two setups by performing a 10 minute test.

- We will measure the number of bytes transferred over the Wifi device on the RPi. That is you ensure that only the ESP8266 uses the wifi during this test and you use ifconfig before and after the test to determine the number of bytes transferred.

- We will measure the approximate power consumption of the ESP8266 during the test. To this end Kjeld has a couple of meters you can add to the ESP8266 to measure the power consumption.

- We will measure the average time to complete a single HTTP POST vs. MQTT transmission. We do this by printing the  Arduino Sketch millis() function output to the serial port right before and right after the transmission.

It is recommended that you configure both the Arduino Sketches to transmit data every 5 seconds to maximise the difference in consumption.

Publish your results on the itslearning forum to compare with the results of other teams.

### 6.2 Add another sensor to MQTT for visualization in Grafana.

Put the below command line into a script and configure Telegraf to save this data into a database. Visualize in Grafana.

```
$ echo "systemp temp=`cat /sys/class/thermal/thermal_zone0/temp`"
```

### 6.3 Mosquitto - InfluxDB - Telegraf - Grafana

It is well worth the time wokring more with this setup. Try adding  another sensor input to the ESP8266 and perform the configuration needed to visualize this in Grafana as well. Try having Grafana control a servo via the Raspberry Pico microcontroller**.**

3 https://mqtt.org/