Grüezi YouTubers. Here is the guy with the Swiss accent. With a new episode around sensors and microcontrollers.

In video #140 we hacked the new IKEA Tradfri Smart lighting system using the remote control as a basis. It worked, but had some major disadvantages. So, I decided to change the strategy and use the IKEA gateway instead of the remote control. This should have some advantages. But if you want to hack the gateway, you quickly discover, that it uses COAP, the second important IOT protocol. Today, we will dig a little into this protocol and compare it with our well known MQTT protocol. At the end

* We will understand the relevance of the COAP protocol and where it is used
* We will have an idea about the concept of RESTful APIs I already used in video #27
* We will understand the basic differences between COAP and MQTT
* We are able to use COAP on our beloved ESP8266
* We will learn a little on how to “spy” networks using Wireshark
* And of course, we will be prepared to hack the IKEA gateway in the next video

But, before we can start, I want to show all my Swedish viewers the reason, why I thought, Tradfri is Danish. If you enter Tradfri into Google translate, it detects automatically Danish language. If you force it to Swedish, it does not translate. I apologize for any inconveniences. BTW, this mistake created the biggest discussion on this channel so far!

But let’s continue with technology!

COAP or Constrained Application Protocol has its roots in the well-known HTTP protocol, which is used for most of our web pages. But it is adapted to the needs and restrictions of small devices. So, it is somehow a “competitor to MQTT.

It uses the concept of RESTful APIs. This concept is widely used for webservices. Most of the websites offer such an API interface.

RESTful calls are easily recognizable if you look at the URL in your browser: It usually starts with an address of the webservice, some additional stuff which defines the particular service you want to use and then a question mark with some more stuff.

https://api.twitter.com/1.1/search/tweets.json?q=%40spiessa

This example searches twitter for all tweets from my user.

Very important is, that with this concept, services can offer four different methods: Read information (with the GET command), update information (with PUT), create new information (using POST) and even delete information with the method DELETE. So, this is a very versatile concept for webservices.

And what is now COAP? COAP is a protocol which can transport RESTful calls over “thin” networks. And it is also adapted to the small resources of IOT devices. This is, why it uses the more efficient UDP, and not like HTTP, the TCP protocol. And it only needs small memories and slow processors for the nodes. Exactly like MQTT. But this is the only commonality as we will see:

Let’s first look at a typical network. We all know the concept of MQTT with one broker and many nodes. There is no direct connection from node to node. All communication is done via the broker.

With COAP, the communication goes directly from a client to a server. No intermediate is necessary. Exactly like webservices. One node also can connect to many different servers. So, somehow, the node is in the center of this network, not the broker.

MQTT uses a publish and subscribe concept. As soon as the broker gets a message from a node, its sends it to the network. Theoretically, all connected nodes can get all messages. Because each node can subscribe to “interesting topics”, it only gets the messages concerning the topics it subscribed. MQTT only transfers information. It has no concept to alter information on the opposite node defined into the standard. If you want to change information, you have to define your own messages to “emulate” GET, PUT, POST, or DELETE.

On the other hand, MQTT has a concept of “Quality of service” and “store and forward” implemented in the protocol, which is superior to COAP, because it also works, if one node or the connection to a node is partially down. This is particularly important if you want to sleep your nodes for a certain time and only connect to the broker at defined times. It is also possible to create such a concept using COAP, but you have to program it yourself. It is not part of the protocol definition and you do not have a broker which is always on, if you do not build it yourself. At least this is, what I understood so far.

So, summarized:

* MQTT is very good at transferring data or commands from and to remote devices over unstable connections because of its unique QOS and store and forward concept
* With MQTT, the broker is the center of the network
* COAP is very well suited to work with servers over relatively stable connections. Nodes can connect to many different servers at the same time and can execute “commands”
* Both protocols are well suited for low volume networks and for low power devices
* Both protocols are heavily used in IOT applications
* Both protocols can use secure lines. With COAP, the preferred cryptographic method is DTLS

So, let’s play a little with COAP. In principle, it is very easy to use it. Just replace the HTTP with COAP://. However, if we do that in Chrome, we do not get the expected effect, because Chrome cannot deal with this protocol. If we use Firefox with an extension called “Copper”, we get an answer. First, we see, that COAP runs on port 5683. Then, we see, that this server exposes a service, called LED. This mechanism of displaying all available services is called “discovery”. So, the client does not need to know exactly, which services are offered by the server. This is a nice feature we will use later with our IKEA setup.

But where does this message come from? People who know me can guess: Of course, it comes from the COAP server running on an ESP8266. And this ESP also has an LED. First, we can request the status of the LED by issuing a GET command for the LED service. We see in the incoming message, that the LED is on. Now, we want to switch the LED off by issuing a PUT “0” command. And really, it switches the LED off. We also can check its new status. It is 0. So, this whole thing seems to work.

At the beginning of this episode, I mentioned, that the COAP protocol works on constraint networks. Let’s check, if this is true and if the messages really are short. To get this information, we can use Wireshark on our PC or on the Raspberry Pi. It is open source and helps with many hacking or network problems. We trigger to the UDP port 5683, which is the port used by COAP. And we see the messages flowing through the network. Their length is usually around 50 to 100 bytes. The longer ones are the discover messages, which are not often used. So, the promise is kept. The packets are small. But I think, they are a little bigger than with MQTT.

How did I build theESP8266 COAP server? Simple! I just installed this library and used the example provided. And of course, there is also an example for a COAP client available. And with this client example file, I am even able to connect to my “Alma Mater”, the Swiss Technical University (ETH Zurich), with my tiny ESP8266. They run one of the COAP test servers. How cool is that?

If we discover all services offered by this server, we see the possibilities of COAP, which are huge. Much bigger, than with MQTT, but, because I do not understand all of them, I will stop here.

Now, we have to focus on one COAP feature, which will be important for our IKEA Hack: Encryption. As with HTTP and its secure sister, HTTPS, COAP also has a secure sister, COAPS. If we replace COAP with COAPS, we see, that Firefox and Copper cannot handle this protocol. Because IKEA correctly uses the secure version of the protocol, we still have to go a long way till we are able to get these damn lamps switching on and off. But you can be sure, we will succeed!

For today, we built a basis for our next step:

* We looked at the relevance of the COAP protocol as the second important IOT protocol together with MQTT
* We learned the meaning of RESTful APIs and played with GET and PUT methods
* COAP is very well suited for client-server applications over stable networks. MQTT, because of its built-in QOS levels, is well suited for applications with unreliable networks or where the nodes are in deep sleep for a long time. The broker can store the messages as long as the node is not available
* We found a library to use our ESP8266 as a COAP server and also as a COAP client and were able to test it in both scenarios
* Wireshark helped us to trace network traffic and to prove, that the COAP messages are really short
* And in the next episode, we will use our Raspberry Pi as a secure COAPS client and connect it to the IKEA gateway. And I will show you a “hot” secret of the IKEA intelligent bulbs

So, stay tuned!

I hope, this video was useful or at least interesting for you. If true, then like. Bye