Mobile and Embedded Computing

LINGI2146

Project report

Hanquin Benjamin

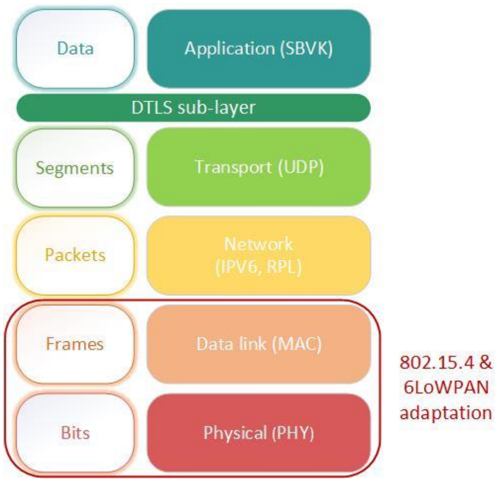
Croche Loïc

Cochez Benjamin

Aranibar Mondragon Vladimir

1. Communication protocol

We decided to use an approach similar to a lightweight version of MQTT, based on UDP and which implements security mechanisms. We called the following protocol “SBVK”. In this one, we find main components of the MQTT protocols which are the Publisher, the Broker and the Subscriber. Some of the names about message types are the same.
The protocol implementation is available at this Github link under the name “SBVK.c” and the header “SBVK.h” : *https://github.com/Bulby-Bull/EmbeddedProjectExam*
We can show on the following figure the implementation of the protocol on a layered model:



We can decompose this protocol in 4 main parts:

1.1. DTLS handshake

The first is dedicated for the secure connection initiation. We decided to implement the DTLS protocol. DTLS is similar to TLS but uses UDP instead of TCP. Thanks to DTLS, we decided to use the Ephemeral Diffie Hellman on Elliptic Curves (ECDHE) for the public key algorithm and ECDSA (Elliptic Curve Digital Signature Algorithm) for the signature algorithm. These algorithms can take a certain amount of resources so, for this reason, we assume that the client/sensor has the minimum requirements in terms of memory.

During this phase, the device and the broker determine their common secret key thanks to a key exchange mechanism. Once the secret key is determined, all payloads exchanged between publishers, subscribers and the broker will be encrypted.

Motivation of these choices : we decided to choose ECDHE with ECDSA because it’s really less resource consumption than the RSA algorithm. Moreover, ECDHE provides the Perfect Forward Secrecy (FPS), it’s not the case for ECDH or RSA. For the symmetric encryption, we choose the AES-GCM (Galois/Counter Mode) which is an authenticated encryption scheme and which provides authentication, integrity and confidentiality during data communication. We take a version with a 128-bits key instead of 256-bits to optimize device ressources. in terms of security, we follow the NIST recommendations for the minimum key size. For ECDHE/ECDSA we chose a P-256 key and for the Hash algorithm, used in ECDSA and AES-GCM, we decided to use SHA-256.

1.2. Login Phase

The second is the log in phase to allow the client to freely communicate with the broker and, by the way, with other devices. We can connect the device to the Broker with a CONNECT request and the broker answers with a CONNACK to inform the device.

1.3. Data transfer Phase

The third phase is for the data transfer like temperature from a sensor or any other data.

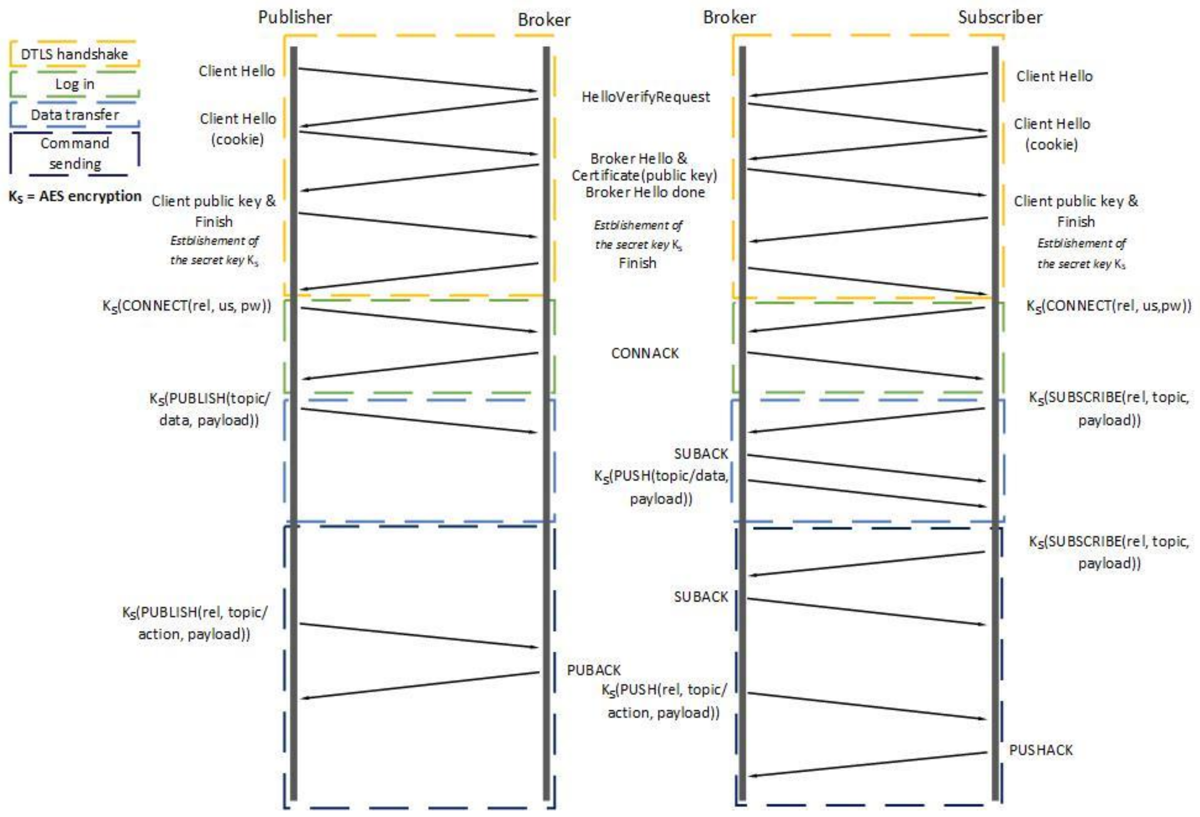
For this communication we can choose a level of reliability, similar to a quality of service (QoS) in MQTT. The difference with MQTT is that we only have two levels of Rel, reliable (1) and non reliable(0). If the Rel is set to reliable an *-ACK is created upon reception and sent back to the source to acknowledge that the data have been received. If the Rel is set to non reliable no ACK is created and thus the data could have never reached the destination.

We can see that if the publisher publishes data with a PUBLISH, there is no acknowledgement because, by default Rel is set to non-reliable. Furthermore, if a subscriber wants to subscribe to a target topic, in this case it receives an acknowledgement called a SUBACK because the Rel has been set to reliable. When the device is subscribed, the broker can transfert the target data present under the topic name with a PUSH.

1.4. Command sending Phase

The last phase concerns the command sending. Considering a connected lamp subscribed to a topic, when a client wants to send a specific command to this device like switching on a light, the client (publisher) can publish its command to the Broker mentioning the topic and action to take. The Broker answers with a PUBACK (because Rel is set to be reliable for commands). Because the device is subscribed to the topic, the broker transfers the command with a PUSH to the concerned device. In this case, the subscriber answers with a PUSHACK because Rel is set to reliable.

1.5. General communication:


Publisher
Broker
Broker
Subscriber
DTLS handshake
Log in
Data transfer
Command sending
K_S = AES encryption
Client Hello
HelloVerifyRequest
Client Hello (cookie)
Broker Hello & Certificate(public key) Broker Hello done
Client public key & Finish
Estblishement of the secret key K_s
Estblishement of the secret key K_s Finish
K_S(CONNECT(rel, us, pw))
CONNACK
K_S(PUBLISH(topic/data, payload))
K_S(SUBSCRIBE(rel, topic, payload))
SUBACK
K_S(PUSH(topic/data, payload))
K_S(PUBLISH(rel, topic/action, payload))
PUBACK
K_S(PUSH(rel, topic/action, payload))
PUSHACK

1.6. Protocol definition

Packet format :

bit	7	6	5	4	3	2	1	0
byte 1	MsT				Rel			
byte 2 - n	Header option							
byte n- m	Payload							

Format of SBVK messages

Message type (MsT) (4 bit):

Int value	Bit value	Meaning	Int value	Bit value	Meaning
0	0000	HELLO	8	1000	UNSUB
1	0001	PUBLISH	9	1001	UNSUBACK
2	0010	PUBACK	10	1010	PINGREQ
3	0011	SUBSCRIBE	11	1011	PINGRESP

4	0100	SUBACK		12	1100	PUSH
5	0101	DISCONNECT		13	1101	PUSHACK
6	0110	CONNECT		14	1110	UNUSED
7	0111	CONNACK		15	1111	UNUSED

Rel (1 bit) :

int value	bit value	Meaning
0	0	non-reliable
1	1	reliable

Header Option (max 20 bytes) :

This header option field contains the additional information needed to perform the message type action

MsT	Additional Information
PUBLISH	Topic name
SUBSCRIBE	Topic name
UNSUBSCRIBE	Topic name

Payload (max 50 bytes) :

The data needed by the message type, this payload complete to additional information provided in the header option.

MsT	Additional Information
PUBLISH	Data to send
PUBLISH	Action to perform

1.7. Some assumptions

Because it's a proof of concept and due to the time we must assume some things. First of all, we assume that a public/private key pair is generated at the start of each device. Moreover, we assume that we use a DTLS compressed version because we are on a 6LoWPAN network thanks to the limited resources.

Furthermore, we decided to apply the security on the transport layer and not on the applicatif side. It would be possible to apply the security on the application layer and it would be more efficient because we could only encrypt headers and payload that we want to save some resources. In addition, we could choose other cryptographic algorithms more lightweight than AES as Saturnin for example.

2. IoT Devices

Because the implementation we need to do is a proof of concept, we did not integrate the security part (dtls) into the environment. Because in the simulation, this would consume too much resources and dtls may not work properly. Nonetheless, we did choose to integrate comment in the code where we would have integrated such security. (Ex. where the key exchange starts, where the dtls communication begins etc..).

All of the protocol implementation is developed in C code in Cooja.. All codes are available at the Github link.

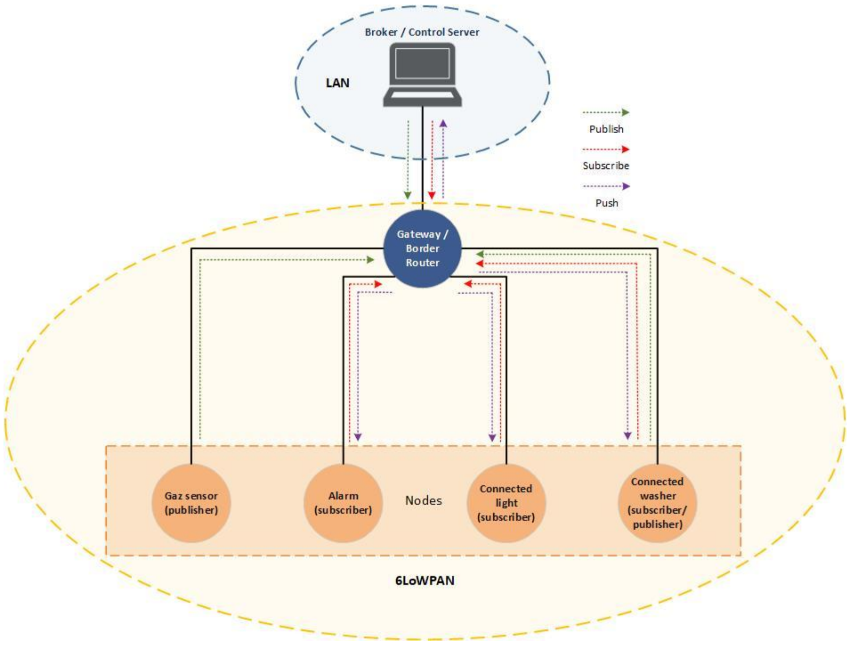
To implement our new protocol, we decided to use some devices:

- ➔ A gas sensor which just publishes its captured values (simple publisher).
- ➔ An alarm which subscribes to the broker for the gas topic (simple subscriber).
- ➔ A connected light which subscribes to the broker to receive some command like switch on/off (simple subscriber).
- ➔ A connected washer which is capable of subscribing from the broker to receive some commands like set a new wash cycle but it also can publish some information to the broker like the current state. For example, if a cycle is finished or in progress, the client can be informed.
- ➔ A broker/control server, the brain of the iot network. This brain dispatches packets to the other nodes, register them, store who is subscribed to what and forward packets to who need them.It also allows the user to send his command and set automations.

All these devices are connected together in a RPL network with as a root a border router device. This border router allows the traffic to flow from the 6LowPan RPL network to a ipv6 network where is located the broker/control server. The server hosts an application that is used by the users to communicate with the devices. The interface allows the user to enter protocol commands and interact with the RPL network like a device (Publish, subscribe, …). The server is also the broker .

For the simulation, the server is deployed on our host (not cooja) and communicates with Cooja via the border router.

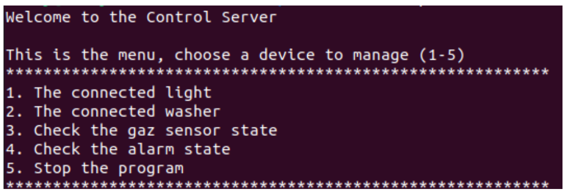
We choose this architecture because it is scalable. We have added 4 types of motes but we could add 10 mores motes and the protocol and the structure will work the same way.



Proof of concept implementation

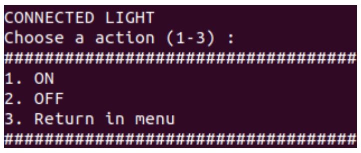
3. Control server program

For this part, we decided to implement the interface in C code. The code is available with the github link in the same directory of the protocol implementation code under the directory name "ControlServer".
We can see in the following picture the main menu.

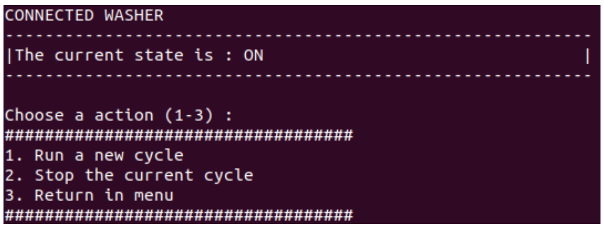

```
Welcome to the Control Server

This is the menu, choose a device to manage (1-5)
**********************************************************
1. The connected light
2. The connected washer
3. Check the gaz sensor state
4. Check the alarm state
5. Stop the program
**********************************************************
```

In this control server interface, it's possible to manage some interactions with IoT devices. When we send a command, the current UDP connection is used and a packet is sent through the border router. On this interface, we can carry out some functionalities:

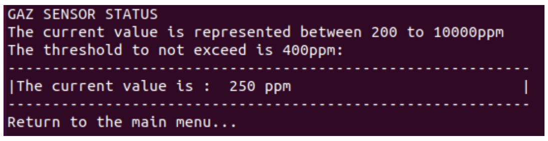
➔ We can switch the light on/off if we choose the first point.


```
CONNECTED LIGHT
Choose a action (1-3) :
####################################
1. ON
2. OFF
3. Return in menu
####################################
```

➔ We can see the current state and send a command to run/stop a new cycle in the washer if we choose the second point.

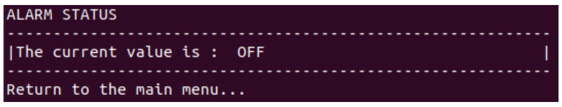

```
CONNECTED WASHER
--------------------------------------------------------------
|The current state is : ON                                   |
--------------------------------------------------------------

Choose a action (1-3) :
####################################
1. Run a new cycle
2. Stop the current cycle
3. Return in menu
####################################
```

➔ For the third point, we can see the state of the gas sensor and all information about the limit to not exceed is provided.



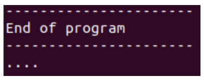
```
GAZ SENSOR STATUS
The current value is represented between 200 to 10000ppm
The threshold to not exceed is 400ppm:
---------------------------------------------------------
|The current value is :  250 ppm                        |
---------------------------------------------------------
Return to the main menu...
```

➔ The fourth point allows us to check the alarm state.



```
ALARM STATUS
----------------------------------------------------------
|The current value is :  OFF                             |
----------------------------------------------------------
Return to the main menu...
```

➔ The last possibility is destined to stop the program.



```
---------------------
End of program
---------------------
....
```

4. Automation

We imaginated multiple automations to show the potential of our ecosystem.

The first one, in the control server, is about the gas sensor. The control server is subscribed to the gas sensor topic, and thus receives data from it. The user can set a threshold to the gas data, and if the gas sensor sends data higher than the threshold fixed, the control server sends a command to the alarm to activate it. The alarm is subscribed to a command topic that allows the control server to send him his command.

The second one, the user is able to set an hour of the day when the light will be lighted up and when the light will be lighted off. The light is subscribed to a command topic that allows the control server to send him his command.

The third automation is the ability for a user to set up a cycle schedule. The user will be able to choose a date (recurring or not) and an hour to schedule his Washing machine cycle. When the cycle is going, the user is able to see the progress of it, and stop it if he wants. The washing machine subscribes to a command topic that allows the command center to send his orders. In the other way, the command center is subscribed to an info topic that allows him to receive status and progression from the washing machine.