Communication between industrial network and public network using Profinet and MQTT protocol

Bachelor's degree final project

Author: Junyang, Bu

Tutor: Víctor M.Sempere Payá

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación.

Valencia, 29 of June, 2016

Abstract

Connection has become the trend of development of world, since last century. From Internet to IoT (Internet of things), more and more physical devices will be connected to the network. However, industrial network as an important part of information source still can’t be connected to public network easily and safety because of its complicated industrial environment and particular communication requirements. Profinet as a standard based on industrial Ethernet provides an integrated solution to industrial communication and MQTT is becoming a potential protocol for M2M communication due to its lightweight and small code footprint. In this project these two solutions are applied to connect the industrial network and public network. Profinet is used for creating an industrial communication network and MQTT is used for connecting the industrial network to public network.

In chapter of Methodology the Profinet and MQTT protocol are introduced in detail. In following chapter development and results the configuration of Profinet and implementation of MQTT protocol in PLC S7-300 is described step by step. Finally the conclusion of project and its future work are discussed in chapter 6. In addition, all the source code (AWL) in this project is appended to the last chapter.

Contenido

1. Introduction

Nowadays, with the development of IT the market changes faster and becomes more unpredictable. The company needs to make fast and effective decision to adapt the changing market. So the up-to-date information about process status in production field is important not only for the production site but also for the management departments [5]. In the outside environment the customers or chain partners also need the information of industry system. Besides, as the development of IoT (Internet of Things) more and more information silos should be broken and more data will be extracted for using and investigating. For these purposes the industrial communication system should connect to commercial network system in a larger range.

In fact the integration idea is not new. In the 1970s the computer integrated manufacturing (CIM) concept was proposed, which already had included the integration idea. The past two decades the commercial network system (e.g., Internet) and automation system both have made great progress. The recent year many new technologies have appeared and try to interconnect the two systems. For example, the vertical integration was a popular solution in the past few years [6]. This solution is used in this project.

Until now the automation system can be designed very complex. To deal with the complexity, depend on the different functionality the components are placed in different places and structured into several hierarchical levels. Typically the hierarchical model is known as the automation pyramid which contains 4 to 6 functional levels in different version. However, in recent years with new technology, the structure is being simplified. Here we divide the automation system into 5 levels [6]. Figure 1 represents the typical structure of automation system with 5 levels.

To link those components an industrial communication system is necessarily, which connects the equipment not only in the same level but also in the different level. In each level there are appropriate protocols to use.

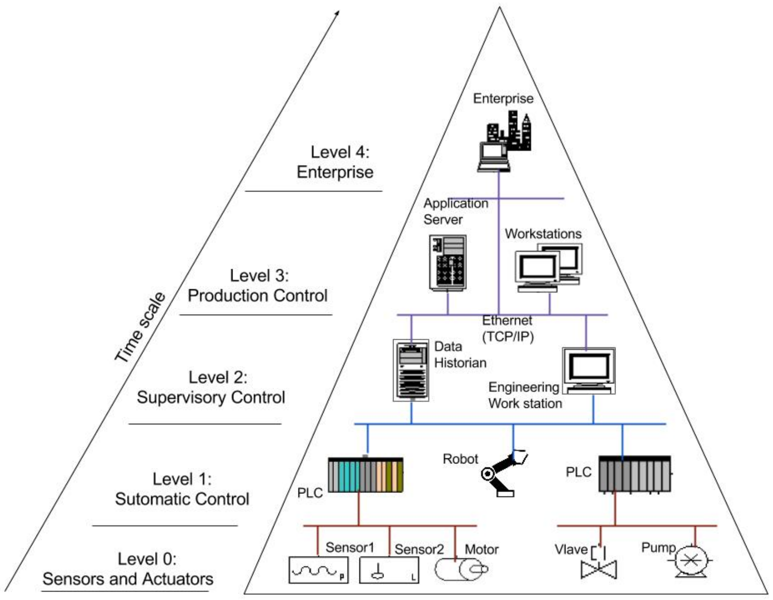


Fig.1: Hierarchy of an industrial automation system [2]

The level 0 is constituted by sensors, actuators that sense and manipulate the production process directly. The level 1 is an automatic control that contains typically hardware PLC, robot, I-Device, etc. These 2 levels consist in field level that is the lowest level in system. The communication is based on real-time system and the data are transmitted with a short but recycle that has the range from 1us to 0.1s depends on the application. Profibus or Profinet can be used for the network. From level 2 to level 4 usually they have the industrial computers with software running based on IT. These computers form large scale networks to gather, store and process industrial data by Ethernet communication. Level 3 is the production level that gathers the management information from lower levels and manages the whole automation system. Level 4 is the highest level in the system and it is in the environment of enterprise. There is Internet or WAN for management information exchange using Ethernet networks as a gateway to connect automation system [1] [2].

Now we focus on the industrial information flow of whole enterprise and outside context. The figure 1 shows us that how the information flow circulates in a typical company.

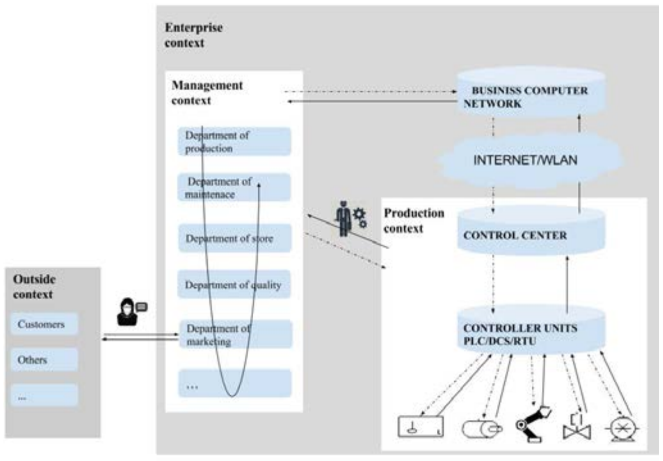


Fig.2: Industrial information flow

Firstly the data are created from production field and transmitted to the controller units. The controller units can be PLC, DCS or RTU units. After those controller units receive data, some data will be processed and response immediately to the production component and some data will be sent to the higher level the control center. The control center consists of engineering workstations, data historian, control server, etc. It receives stores and processes data from controller units. The data can be fed to data historian or after analysis of data the control center may send commands to PLCs or RTU. Finally, through Internet or WLAN the control center exchange information with business computer network [4]. Then the information from industrial context comes to the management context and circulates in different department.

From above explication there are some problems in the industrial information flow. Firstly, for the outside environment is very difficult to gather industrial information, because usually the communication with customers or chain partners is in charge of department of marketing. The automation system or production processes is like a black box for outside people. Secondly, not all the companies own the complete structure of automation system. So usually they constitute their system according to the actual demand. For example, maybe they only have the level 0 and level 1. In this case it's difficult to connect to the business network if they don't add more level of system. Lastly, in pyramid structure the higher level manage the whole automation system and also it connects to an open environment. That raises the risk of attack and vulnerability can cause very serious damage to the whole industrial system. So the security question always is the challenge for network interconnections.

In this project we try to use a simple and safe method to get the information from industrial system, which is MQTT server. MQTT is a lightweight publish/subscribe messaging protocol based on TCP/IP. It is designed for connection with remote locations in 1999 by IBM. Because of its easy implementation and limited resource occupied, it becomes the potential M2M connectivity protocol [7].

In figure 3 there is an example application in industrial communication system with protocol MQTT. If we compare the figure 2 and figure 3, it is quite obvious that the automation system is the same as before. The only thing that should be added is a small block of MQTT communication that is used for sending data to MQTT server. The block is so small and simple and can be installed in any level where uses TCP/IP based protocol for communication. The component which is installed that block is named publisher who will publish data to server. In other side of server the clients who will receive the message are named subscribers. After the subscribers subscribe a topic, they will receive massage which has that topic from publisher with real-time communication system. More detail will be discussed in the project.

Now we can see the advantage of MQTT in this system. Firstly it is easy to implement, we don't have to add more device. Secondly, any component in any level can transmit data from system. Thirdly, it is flexible for subscribers. They can receive the message that they want to receive. Fourthly, it is safe for automation system because of unidirectional communication mechanism. The publishers in automation system only send message will not receive anything from outside.

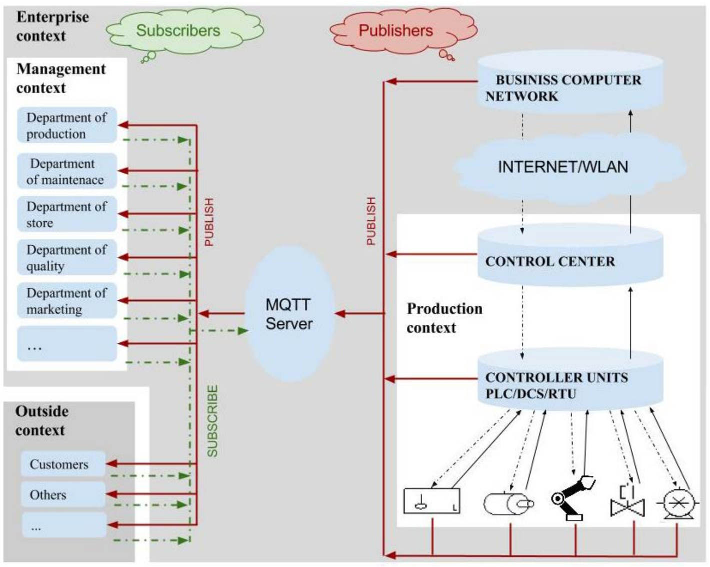


Fig3: Application of MQTT protocol in industrial communication system

The project is divided in four sections. The first sections describes the Profinet standard and MQTT protocol. Their histories, theories, communication mechanisms, securities are mentioned. In the second section an application is developed with Profinet and MQTT. It shows the configuration of Profinet step by step and the implements of MQTT are also shown in this section. The language of AWL is used for programming in this project. In the third section the equipment that is used during this project is listed and the last section the conclusion and its future work will be discussed.

2. Objectives

In this project the principal objective is to analyze, implement and verify the feasibility and the reliability of connection between automation system and public network system using MQTT server. Figure 4 shows the automation communication system that will be constructed. In the part of field context simulation one workstation, two controllers and one IO-Device consist in the hierarchical levels using protocol Profinet to set up communication network. The other part is the public network simulation which contains three PCs with application MQTTlens and an MQTT server to build a public network based on MQTT protocol. Finally the feasibility and reliability of this communication system is verified by long time experiments.

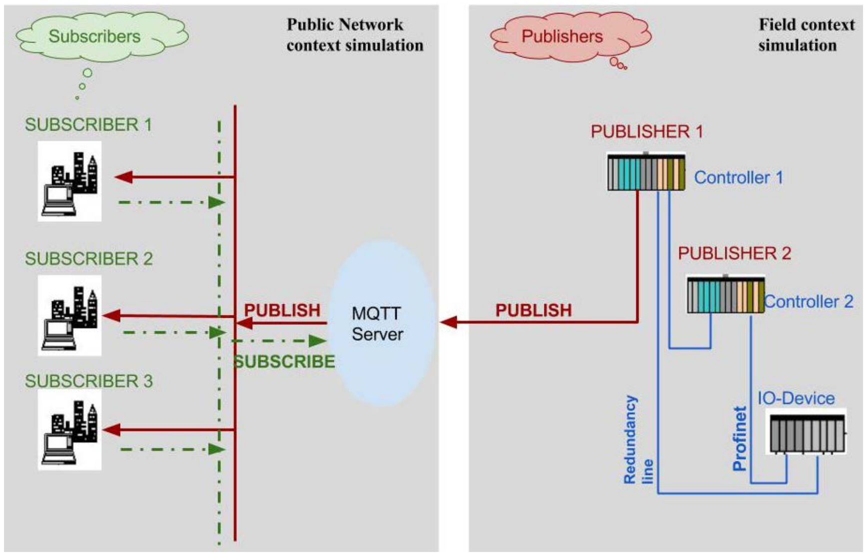


Fig.4: Automation communication system with MQTT

3. Methodology

3.1 Profinet

3.1.1 History of Profinet

If we want to the development history of Profinet, we should start with the Profibus and Industrial Ethernet, because the Profinet can be considered as the combination of these two standard systems.

Profibus is the abbreviation for process field bus. It is a standard for fieldbus communication in automation system. The start was in 1986 when the field device interface needs to be standardized because of the appearance of different technologies of fieldbus. It was defined by 21 companies and institutes in Germany with the push of government in order to create a uniform bit-serial fieldbus standard as an open national standard DIN 19245. Thereafter, a number of voluntary enterprises which use the standard established the Profibus User Organization (PNO) in 1989 to maintain and advance Profibus. Then a big group named Profibus International was formed in 1995, which works to set standards, develops new technologies and assures the quality [8]. Profibus can be divided into Profibus-DP, Profibus-PA and Profibus-FMS according to the application. Profibus-DP (Decentralized Periphery) is designed for the communication between the central automation devices and decentralized field devices [21].

Profibus-PA (Process automation): It is used for the communication between measuring and process equipment by the standard process of transmitting measured data in the process automation application.

Profibus-FMS (Fieldbus Message Specification): It is to solve the problem of versatility in workshop level communication providing a large number of communications services.

Meanwhile, in 1970s Ethernet became the standard solution of desktop communication systems based on TCP/IP in the layer 1 and 2 of ISO model. But because of its non-deterministic response time and the possibility of lost and delay of frames, it can’t satisfy the t requirements of automation network system that is based on the real-time control. To solve this problem, the Industrial Ethernet has been created since 1990s. Compare to the Ethernet the Industrial Ethernet uses a variety of methods to improve the performance and quality of Ethernet to meet the requirements of industry. Regarding the Ethernet network congestion problem, it uses Ethernet switch to reduce the conflict and erroneous transmission. To improve the confidentiality of Ethernet it uses the high-speed Ethernet because the error rate can be reduced greatly with the speed of transmission. To solve the problem of response time, the IEEE1588 defines the clock synchronization protocol. Moreover, the link redundancy also was added to Industrial Ethernet. Until now the Industrial Ethernet has become a big family with different solution for different problems. According to the features, the real-time capability, the QoS mechanism available etc., the appropriate solution in the family can be selected [21].

The Profinet is an Industrial Ethernet solution on the basis of Profibus developed by Profinet International group. So it takes the advantages of Industrial Ethernet and Profibus. It was started in 2000 and the first Profinet protocol is Profinet CBA which is designed to integrate the different complex machine using M2M communication based on TCP/IP. Then the second protocol is Profinet IO which dedicates to deal with the more-demanding distributed I/O [20].

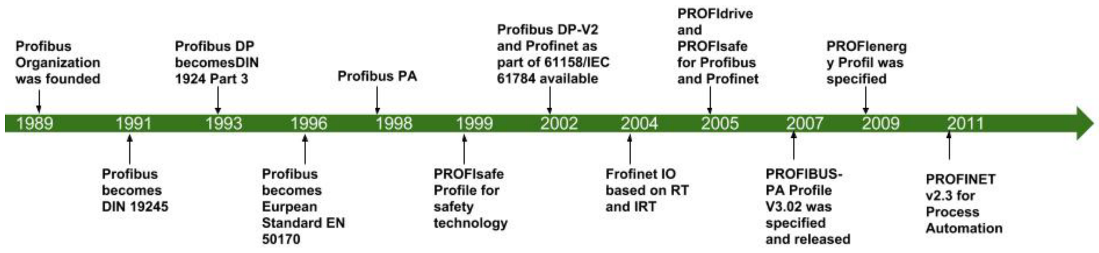

Profibus Organization was founded
Profibus DP becomesDIN 1924 Part 3
Profibus PA
Profibus DP-V2 and Profinet as part of 61158/IEC 61784 available
PROFIdrive and PROFIsafe for Profibus and Profinet
PROFIenergy Profil was specified
1989
1991
1993
1996
1998
1999
2002
2004
2005
2007
2009
2011
Profibus becomes DIN 19245
Profibus becomes Eurpean Standard EN 50170
PROFIsafe Profile for safety technology
Frofinet IO based on RT and IRT
PROFIBUS-PA Profile V3.02 was specified and released
PROFINET v2.3 for Process Automation

Fig.4: History of Profinet [11]

3.1.2 Introduction of Profinet

The Profinet is an Industrial Ethernet solution on the basis of Profibus developed by Profinet International group. So it takes the advantages of Industrial Ethernet and Profibus. Actually there are two versions of Profinet have been released. The first veision is Profinet CBA (component-based automation) which is designed to integrate the different complex components at higher level using M2M communication based on TCP/IP. The component in Profinet CBA covers all mechanical, electrical and IT variables and can be repeated with identifier or slightly modified. Every component can realize the required functionality by the program in it. In fact this is an object-oriented programming concept with which the protocol divides the whole system into different operating units. So in Profinet CBA these components only need to be controlled the input signals and configured to constitute a component-based architecture. Now the protocol is used in North American market and be considered a best solution for plant supervising and control [20].

Then the second family is Profinet IO (Input Output) which is dedicated to implement the more-demanding distributed I/O peripheral devices [6]. The Profinet IO is based on the Profibus DP Industrial Ethernet automation standard. It supports all the communication type, for example, the Non-RT, RT, RT class or IRT. The characteristics of this communication type will be talked about in the following sections. In this project we will only use Profinet IO, so we focus on Profinet IO here [13].

3.1.3 Profinet IO

As we have mentioned before, Profinet IO is designed on the Ethernet-based automation international standard of PROFIBUS. Profinet IO is based on Switched Ethernet full-duplex operation with bandwidth of 100 Mbits/s. The following figure shows an example of Profinet IO system.

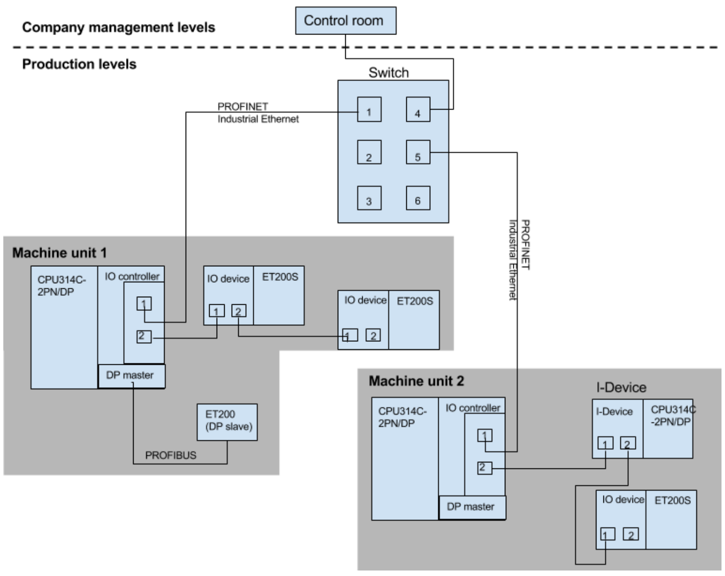


Fig.5: Example of Profinet IO system [18]

Compared with Profibus DP, some processes of data view are retained for migration to Profinet IO, e.g. the I/O data, Data records and connection to a diagnostics system. Profinet uses the provider/consumer model for data exchanges and the field devices in field level provides process data to consumer [18].

3.1.4 Profinet IO device classification

According to the function of devices, they can be divided into the following various classes.

IO Supervisor: This kind of devices can be PG(Programming Device), PC (Personal Computer), HMI(human machine interface), etc. They used for commissioning or diagnostic purposes.

IO Controller: In IO controller the typical device is PLC(Programmable Logic Controller) and it controls automation system with program running.

IO Device: It refers to the I/O field device distributed in field level [18].

3.1.5 Communication in Profinet IO

Profinet IO is a communication system that works in scalable real time based on layer 2 of fast Ethernet. RT communication is the basis for data exchange of Profinet IO. Just as we have mentioned before, to satisfy the requirements of industrial field the industrial Ethernet is developed from standard Ethernet. For example, in industrial automation the time behavior and isochronous operation can not be satisfied by standard Ethernet. To resolve the problem the Profinet IO adopts a scalable RT approach. With different requirements of communication, the different classes have been defined for data exchange. In terms of the performance, the Profinet IO differentiates the following classes.

Non-RT:

Non RT is non-time-critical data transmission based on TCP/IP and UDP/IP. It is used for configuration and parameterization. For example the IT landscape.

RT communication:

In Profinet IO the real-time communication uses standard Ethernet in devices and is used for time-critical process data. The data packages are prioritized according to IEEE802.1Q and the network components control the data flow on the basis of priority. The RT message frames have priority 6, the second highest level, so it ensures the reliability of communication in automation system. With RT it can realize the update time from 250us with RT. RT communication in Profinet IO is the optimum solution of for integrating I/O systems[17].

In terms of hardware it doesn’t need special hardware and uses commercially industrial switches as component.

Isochronous RT:

IRT is suitable for particularly sophisticated Motion Control and high performance application in automation system. IRT permits cycle times of up to 250us. To achieve that performance the communication cycle is divided into a deterministic part and an open part by reserving bandwidth, and then specified to the other station in the network by a sync master. The data can be transmitted without interference[17].

3.1.6 Profinet IO addressing

Every component in Profinet has three addresses, the MAC address, IP address and Device name. Profinet is a protocol on basis of TCP/IP, so the MAC address and IP address are essential for every module.

The MAC address is assigned by its factory as an identifier which is unique in all over the word and it can't be changes. The MAC address contains 6 bytes data and it is divided into two parts with 3 bytes. One part identifies factory and the other part identifies product. E.g. the MAC address 08-00-06-6B-80-C0.

08-00-06	6B-80-C0
Factory	Product

Table1: Interpretation of MAC address

The IP address is dynamic and it can be changed according to the requirement of network. It must be unique in one Network. The direction is consisted of 4 numbers with the range from 0 to 255 and it is separated by 3 points. IP direction defines the address of network and address of this device. E.g. a device has IP address 192.168.0.2 and mascara 255.255.0.0.

IP address (Binary)	11000000	10101000	00000000	00000010
Mascara(Binary)	11111111	11111111	00000000	00000000
Network address	11000000	10101000	00000000	00000000
Device address	00000000	00000000	00000000	00000010

Table 2: Interpretation of IP address

Device name is also an essential identifier for Profinet IO device. This procedure has been chosen, because name is much easier to handle than IP direction. The name has to be assigned to Profinet IO device by Profinet IO controller.

3.2 MQTT protocol

3.2.1 Introduction

MQTT was invented by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom in 1999. The background is that they wanted to create protocol with minimal battery loss and minimal bandwidth cost for the connection between oil pipelines and satellite. They want the protocol have the following characters [11]:

- Easy to implement,
- Provide a QoS data delivery,
- Lightweight and bandwidth efficient
- Data agnostic
- Continuous Session Awareness

So these characters become the core of MQTT. It is a lightweight client server messaging protocol with publish/subscribe model based on TCP/IP. It is designed lightweight, open, simple and to be easy to implement [9] and in the context of M2M or IoT it becomes a potential protocol. Then we will talk more detail about it. The figure 6 shows the principal mechanism of MQTT. The client who sends the message is named publisher and who receives the message is named subscribers and a client can be both subscriber and publisher. The MQTT broker is like a distributor which selects the message and sends it to the subscriber who has already subscribed the message. It means that the publishers and subscribers are independents. So they don’t have to know the existence of each other. This project we will establish the part of publisher, so we will focus on the publishers [9].

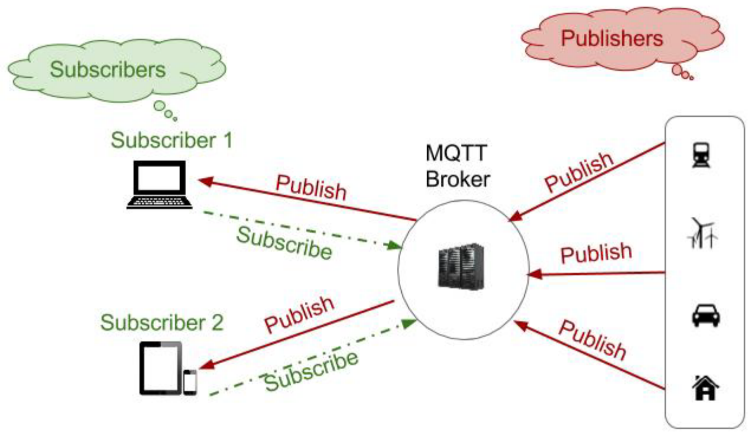


Fig.6: Publish/Subscribe mechanism

3.2.2 Clients

In introduction the clients are divided into publishers and subscribers according to their functionality in the system. In this project the PLCs are publishers and the PCs are subscribers.

3.2.3 Connection

- **CONNECT**

The MQTT is based on the basis of TCP/IP. It means that both client and broker need a TCP/IP stack. Firstly we should realize the connection of TCP and then realize the connection of MQTT. The figure 7 shows the mechanism of connection. The client sends a petition of CONNECT first. After that the MQTT broker receives the message and there is no problem, the broker will response a CONNACK. Then the connection has been established and until the client send a disconnect message to drop the connection [9].

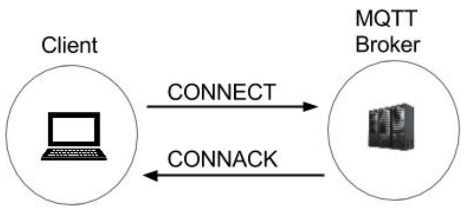
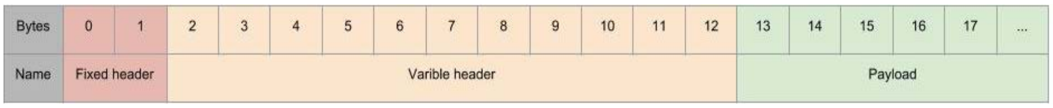


Fig.7: Connection of MQTT

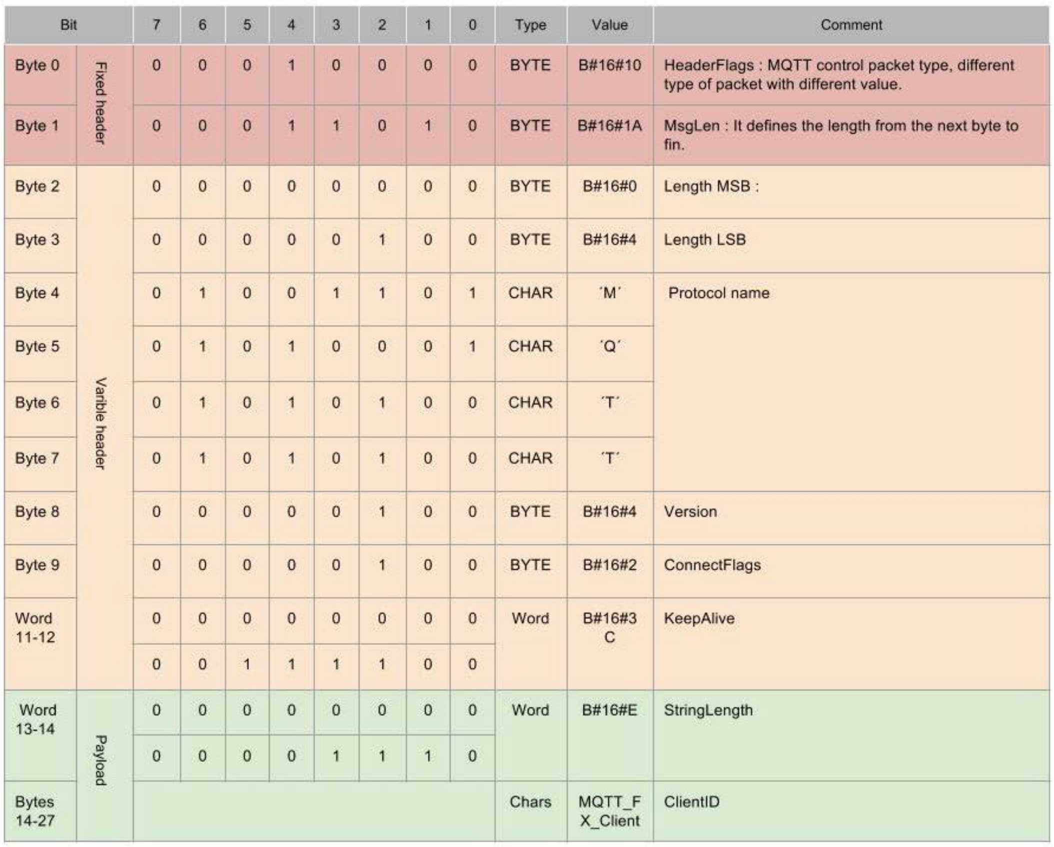
Now we focus on the CONNECT message, because we need to constitute it in the project. In table 3 the message is divided into three parts the fixed header with 2 bytes, the variable header with 11 bytes and the Payload with rest bytes [9].



Bytes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...
Name	Fixed header		Varible header											Payload					

Table3. Structure of CONNECT message [11]

In table 4 there is a simple example connect message.



Bit		7	6	5	4	3	2	1	0	Type	Value	Comment
Byte 0	Fixed header	0	0	0	1	0	0	0	0	BYTE	B#16#10	HeaderFlags : MQTT control packet type, different type of packet with different value.
Byte 1		0	0	0	1	1	0	1	0	BYTE	B#16#1A	MsgLen : It defines the length from the next byte to fin.
Byte 2	Varible header	0	0	0	0	0	0	0	0	BYTE	B#16#0	Length MSB :
Byte 3		0	0	0	0	0	1	0	0	BYTE	B#16#4	Length LSB
Byte 4		0	1	0	0	1	1	0	1	CHAR	´M´	Protocol name
Byte 5		0	1	0	1	0	0	0	1	CHAR	´Q´	
Byte 6		0	1	0	1	0	1	0	0	CHAR	´T´	
Byte 7		0	1	0	1	0	1	0	0	CHAR	´T´	
Byte 8		0	0	0	0	0	1	0	0	BYTE	B#16#4	Version
Byte 9		0	0	0	0	0	1	0	0	BYTE	B#16#2	ConnectFlags
Word 11-12		0	0	0	0	0	0	0	0	Word	B#16#3C	KeepAlive
		0	0	1	1	1	1	0	0			
Word 13-14	Payload	0	0	0	0	0	0	0	0	Word	B#16#E	StringLength
		0	0	0	0	1	1	1	0			
Bytes 14-27										Chars	MQTT_FX_Client	ClientID

Table 4: Example of CONNECT message[11]

In the fixed header byte 0 is a header flags. In its position of bits 7-4 indicates the type of packet and bits 3-0 is reserved for flags specific to this type. In table 6 shows the different control packet types corresponding the value in bits 7-4. From the table we can see the connect message CONNECT has the value 1, so in the position of bits 7-4 of CONNECT there is a 1. The byte 1 indicates the remaining length that counts from the byte 2 to the final byte.

In the variable header the message consists of 4 fields, the protocol name, the protocol level, the connect flags and keep alive. In table 4 the protocol name is from byte 2 to byte 7 using UTF-8 encoded string that represents “MQTT”. Usually the protocol name will not be changed. The protocol level is used for checking the protocol vision with MQTT server. For example here the protocol level is 4. If the server doesn’t support this level, is will response code 0x01 and disconnect the client. The third segment is CONNECT flags. It contains 8 flags with each bit in byte 9. It specifies the behavior of the connection and the necessary field in the payload. Table 5 shows the different flag in the different position of byte 9 [11].

Bit	7	6	5	4	3	2	1	0
Flag	User Name	Password	Will Retain	Will QoS		Will Flag	Clean Session	Reserved

Table 5: CONNECT flags [11]

- **User Name flag**
  0 : The payload must contain the name.
  1 : The payload mustn’t contain the name.
- **Password**
  0 : The payload must contain the password.
  1 : The payload mustn’t contain the password.
- **Will Retain**
  0 : The server must publish the message as a non-retain message.
  1 : The server mustn’t publish the message as retained message.
- **Will QoS**
  0 : The Will QoS flags must be 0.
  1 : The Will QoS flags can be 0, 1 or 2.
- **Clean Session**
  0 : The server must recover the communication with the client based on the state of ession. If there is no session created for the client, the server must create now one.
  1: The server must discard the previous session associated with this client and start the new connection.

The keep alive parameter is a time interval in second and indicates the permitted maximum time interval between the last finish of transmission of one control packet and next start of transmission. If the the keep alive value is nonzero and the server doesn’t receive any control packet from client within 1.5 times the keep alive value seconds, the server will consider that the network is failed and disconnect the connection. So the client must ensure that the interval time between two control packets doesn’t exceed the keep alive time period. The client can send the Pingreq packet any time to keep the connection [11].

The fields in payload are specified by the connect flags. These possible fields can be client identifier, will topic, will message, username or Password according to the flags.

- **Client Identifier**

The client identifier is unique and identifies the client to the server. It is located in the first field and encoded by UTF-8 string. In the figure 6 we can see the part of payload that only contains the client identifier [11].

- **Will topic**

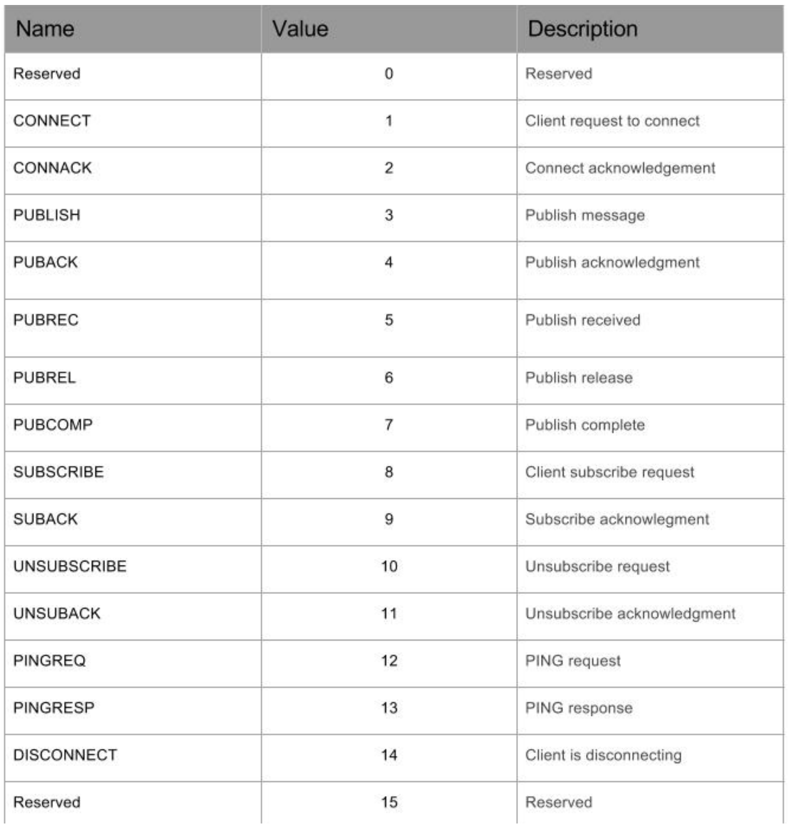
The will topic field will be represented, if the Will flag is set to 1. The will topic must be a UTF-8 encoded string.

- **User name**

As same as will topic, when the user name flag is set to 1, the user name field will be represented and will be a UTF-8 encoded string.

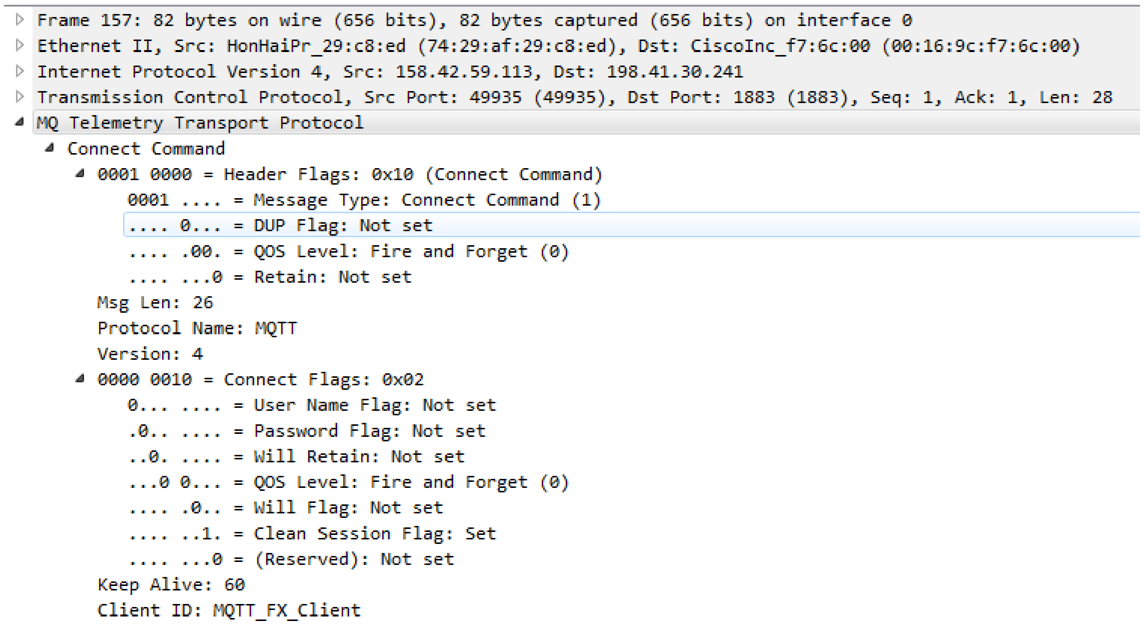
- **Password**

If the password flag is set to 1, the field of password need to be represented. The password field contain 0 to 65535 bytes of binary data prefixed with two byte length field which indicates the number of bytes of bytes [11].



Name	Value	Description
Reserved	0	Reserved
CONNECT	1	Client request to connect
CONNACK	2	Connect acknowledgement
PUBLISH	3	Publish message
PUBACK	4	Publish acknowledgment
PUBREC	5	Publish received
PUBREL	6	Publish release
PUBCOMP	7	Publish complete
SUBSCRIBE	8	Client subscribe request
SUBACK	9	Subscribe acknowlegment
UNSUBSCRIBE	10	Unsubscribe request
UNSUBACK	11	Unsubscribe acknowledgment
PINGREQ	12	PING request
PINGRESP	13	PING response
DISCONNECT	14	Client is disconnecting
Reserved	15	Reserved

Table 6: MQTT control packet type [11]

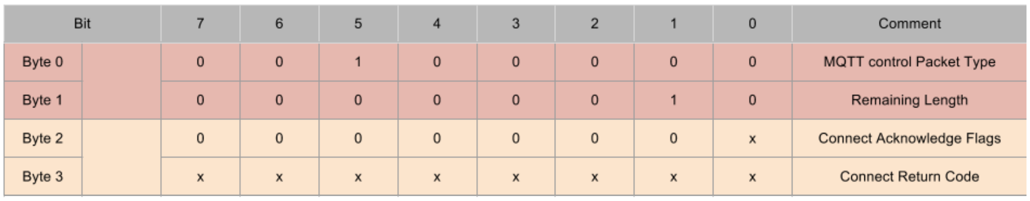


```
▷ Frame 157: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0
▷ Ethernet II, Src: HonHaiPr_29:c8:ed (74:29:af:29:c8:ed), Dst: CiscoInc_f7:6c:00 (00:16:9c:f7:6c:00)
▷ Internet Protocol Version 4, Src: 158.42.59.113, Dst: 198.41.30.241
▷ Transmission Control Protocol, Src Port: 49935 (49935), Dst Port: 1883 (1883), Seq: 1, Ack: 1, Len: 28
◢ MQ Telemetry Transport Protocol
   ◢ Connect Command
      ◢ 0001 0000 = Header Flags: 0x10 (Connect Command)
           0001 .... = Message Type: Connect Command (1)
           .... 0... = DUP Flag: Not set
           .... .00. = QOS Level: Fire and Forget (0)
           .... ...0 = Retain: Not set
        Msg Len: 26
        Protocol Name: MQTT
        Version: 4
      ◢ 0000 0010 = Connect Flags: 0x02
           0... .... = User Name Flag: Not set
           .0.. .... = Password Flag: Not set
           ..0. .... = Will Retain: Not set
           ...0 0... = QOS Level: Fire and Forget (0)
           .... .0.. = Will Flag: Not set
           .... ..1. = Clean Session Flag: Set
           .... ...0 = (Reserved): Not set
        Keep Alive: 60
        Client ID: MQTT_FX_Client
```

Fig. 8: CONNECT packet captured by Wireshark

- **CONNACK**

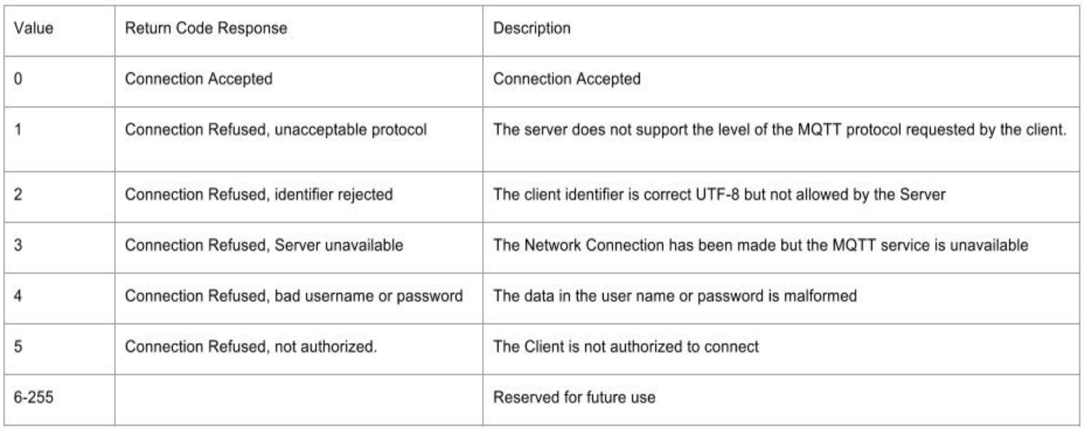
After that the server receives connect packet, the server will response a CONNACK message to client. If the client does not receive a CONNACK packet in a period of time, the client should close the connection and try again. Now we focus on the CONNACK packet in table 9.



Bit		7	6	5	4	3	2	1	0	Comment
Byte 0		0	0	1	0	0	0	0	0	MQTT control Packet Type
Byte 1		0	0	0	0	0	0	1	0	Remaining Length
Byte 2		0	0	0	0	0	0	0	x	Connect Acknowledge Flags
Byte 3		x	x	x	x	x	x	x	x	Connect Return Code

Table 9: Structure of CONNACK Packet [11]

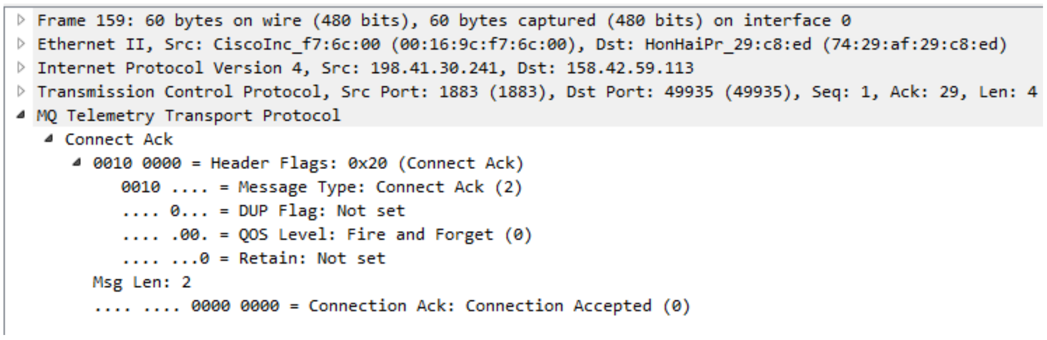
There are two parts in CONNACK, the fixed header with 2 bytes and the variable header with 2 bytes. In the byte 0 these bits in position 7-4 indicate the type of packet. In figure 7 it shows that the CONNACK has the value of 2. The byte 1 indicates the remaining length. In the byte 2 of variable header it only uses the position of bit 0 which presents the session present flag. If the MQTT server accepts a connection packet with clean session set to 1, the flag of session present must be set to 0. If the server accepts a connection with clean session set to 0 and the server has stored session state of the supplied client ID the session present flags will be set to 0. If the server hasn't stored the session state, it will be set to 1. The last byte contains connect return code from 0 to 5 and the each one represents one response. Table 10 shows the different connect return and its description [11].



Value	Return Code Response	Description
0	Connection Accepted	Connection Accepted
1	Connection Refused, unacceptable protocol	The server does not support the level of the MQTT protocol requested by the client.
2	Connection Refused, identifier rejected	The client identifier is correct UTF-8 but not allowed by the Server
3	Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	Connection Refused, bad username or password	The data in the user name or password is malformed
5	Connection Refused, not authorized.	The Client is not authorized to connect
6-255		Reserved for future use

Table 10: Connect return code [11]

If the connect return code of CONNACK packet contains non-zero, it means that the server is unable to process the connect packet for some reason. Then the server must close the Network Connection.



```
▷ Frame 159: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▷ Ethernet II, Src: CiscoInc_f7:6c:00 (00:16:9c:f7:6c:00), Dst: HonHaiPr_29:c8:ed (74:29:af:29:c8:ed)
▷ Internet Protocol Version 4, Src: 198.41.30.241, Dst: 158.42.59.113
▷ Transmission Control Protocol, Src Port: 1883 (1883), Dst Port: 49935 (49935), Seq: 1, Ack: 29, Len: 4
◢ MQ Telemetry Transport Protocol
  ◢ Connect Ack
    ◢ 0010 0000 = Header Flags: 0x20 (Connect Ack)
        0010 .... = Message Type: Connect Ack (2)
        .... 0... = DUP Flag: Not set
        .... .00. = QOS Level: Fire and Forget (0)
        .... ...0 = Retain: Not set
      Msg Len: 2
      .... .... 0000 0000 = Connection Ack: Connection Accepted (0)
```

Fig. 9: CONNACK packet captured by Wireshark

3.2.4 Publish

The MQTT offers 3 types of publish mechanism with different level of QoS. Figure 10, 11, 12 show the three mechanisms of publish with QoS 0,1,2.

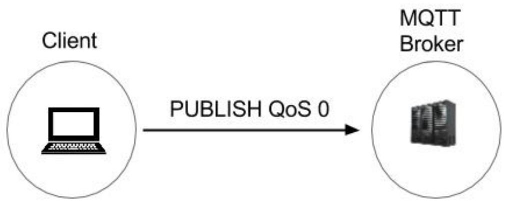


Fig. 10: Publishing mechanism with QoS=0.

When the QoS has the value of 0, the client will guarantee a best effort delivery. The server will not respond and acknowledge regardless the receiver has received the packet or not. So the sender only guarantees the effort but don't care about the safety of transmission.

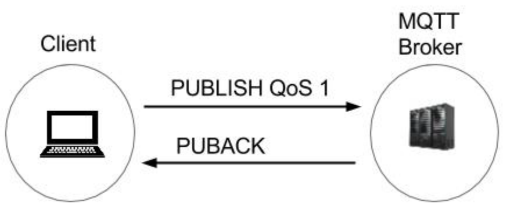


Fig. 11: Publishing mechanism with QoS=1.

When the packet with the QoS level 1 is received correctly, the receiver needs to acknowledge the sender. It is guaranteed that a message will be delivered at least once to the receiver. But the sender has failed the transmission, it should send more than once.

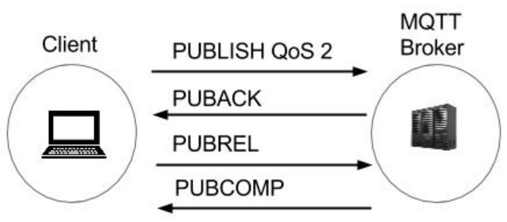
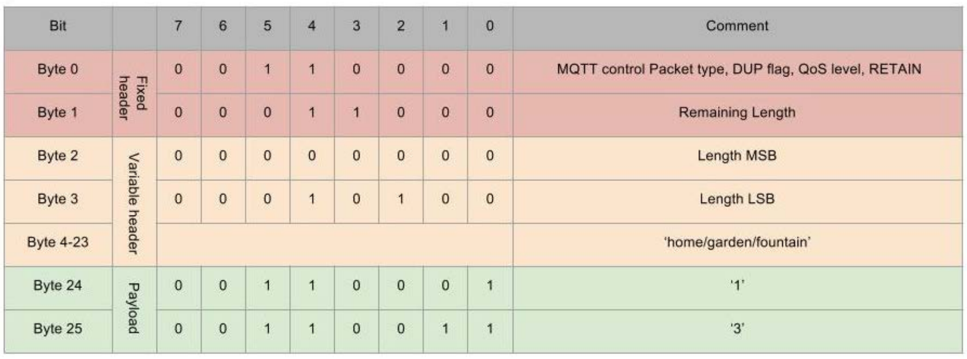


Fig. 12: Publishing mechanism with QoS=2.

The highest QoS level is 2. It is the safest type to transmit data, but with the lowest quality of service level. It will do 2 times of acknowledge once for the publish packet and the other for the puback. So it guarantees that each message is received only once by the counterpart.

3.2.4.1 Packets in publishing process with QoS=0

The publish control packet is sent from a publisher to server or from a server to subscriber. The principle function of the packet is to transmit the topic message information to server or subscriber. The table 11 shows an example publish message.



Bit		7	6	5	4	3	2	1	0	Comment
Byte 0	Fixed header	0	0	1	1	0	0	0	0	MQTT control Packet type, DUP flag, QoS level, RETAIN
Byte 1		0	0	0	1	1	0	0	0	Remaining Length
Byte 2	Variable header	0	0	0	0	0	0	0	0	Length MSB
Byte 3		0	0	0	1	0	1	0	0	Length LSB
Byte 4-23										'home/garden/fountain'
Byte 24	Payload	0	0	1	1	0	0	0	1	'1'
Byte 25		0	0	1	1	0	0	1	1	'3'

Table 11: The Publish Control Packet with QoS level 0 [11].

The message consists of three parts, the fixed header, the variable header and Payload. The fixed header contains some important information about the message. The figure 12 illustrates the flags in byte 0.

Bit	7	6	5	4	3	2	1	0
Byte 0	0	0	1	1	x	x	x	x
Flag	MQTT Control Packet type				DUP flag	QoS level		RETAIN

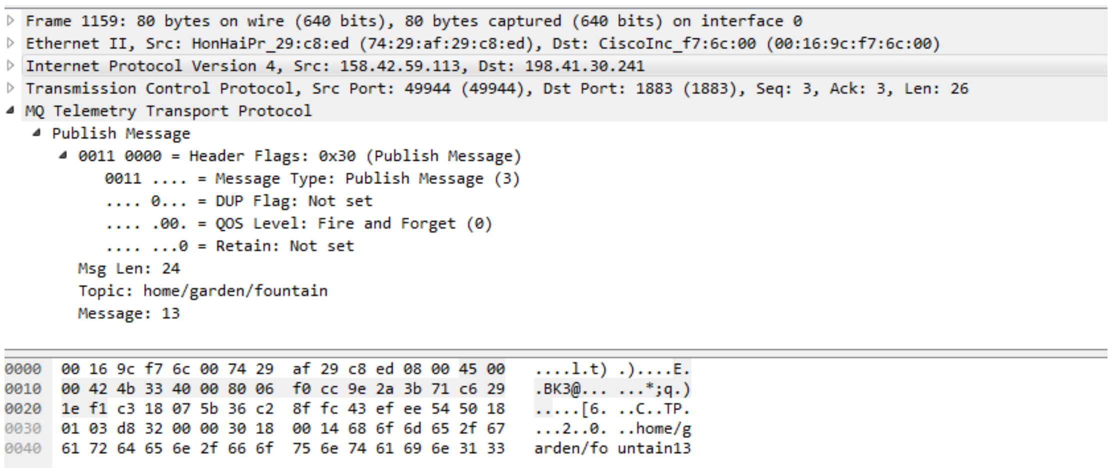
Table 12: The byte 0 [11]

In the position 7-4 of this byte 0 these bits indicate control packet type and in this case the value is 3. In bit 3 if the UDP flag is set to 0, it means the message is the first occasion that the client or server send and if the flag is set to 1, it means the message is redelivery of an earlier attempt to send the Packet. In bits 2-1 contains the value of QoS level which indicates the level of assurance for delivery of an application message [11].

QoS = 0 : At most once delivery.
QoS = 1 : At least once delivery.
QoS = 2 : Exactly once delivery.

The last bit represents a retain flag. Firstly, we discuss the publish message which is sent to server by client. If the retain flag of publish packet from client to server is set to 1, the server must store the application message and its QoS and the server must deliver the message to future subscribers whose subscriptions match its topic name. So when a new subscription is established, the last retained message which match the topic will be sent to the subscriber. If the client want to change the retain message, it should send a publish message with RETAIN flag set to 1 and QoS set to 0. Then the server must discard any message retained for that topic and store the new QoS 0 message as the new retain message for that topic. It may choose to discard it at any time, if this happens there will be no retained message for that topic. A publish message with a retain flag set to 1 and a payload containing zero bytes will be processed as normal by the server, but any retained message with the same topic name must be removed and there will be no retained message for that topic. When the message is sent to client by server,the retain flag must to be set to 1, if the message is sent because of the a new subscription being made by client. In other case, the retain flag must to be set to 0, no matter how the flag was set in the message when is was received [11].

The byte 1 indicates length of variable header and payload. The variable header contains topic name field and packet identifier field. The topic name must be present in the first filed and be a UTF-8 encoded string. The packet identifier field is only used when the QoS is 1 or 2. So in table 11 there is no packet identifier. The last part is the payload. It contains the data which is application specific. It is valid for a publish packet to contain a zero length payload [11].



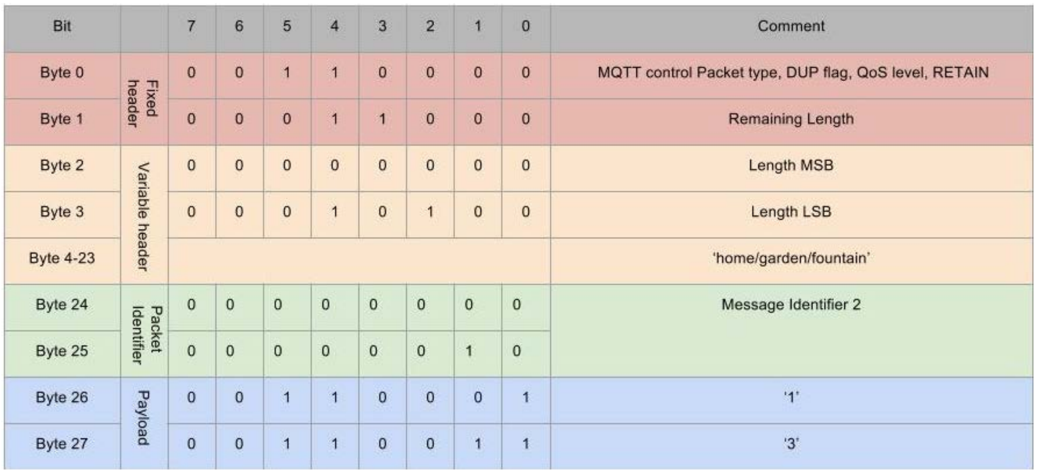
```
▷ Frame 1159: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
▷ Ethernet II, Src: HonHaiPr_29:c8:ed (74:29:af:29:c8:ed), Dst: CiscoInc_f7:6c:00 (00:16:9c:f7:6c:00)
▷ Internet Protocol Version 4, Src: 158.42.59.113, Dst: 198.41.30.241
▷ Transmission Control Protocol, Src Port: 49944 (49944), Dst Port: 1883 (1883), Seq: 3, Ack: 3, Len: 26
◢ MQ Telemetry Transport Protocol
  ◢ Publish Message
    ◢ 0011 0000 = Header Flags: 0x30 (Publish Message)
        0011 .... = Message Type: Publish Message (3)
        .... 0... = DUP Flag: Not set
        .... .00. = QOS Level: Fire and Forget (0)
        .... ...0 = Retain: Not set
      Msg Len: 24
      Topic: home/garden/fountain
      Message: 13

0000  00 16 9c f7 6c 00 74 29  af 29 c8 ed 08 00 45 00   ....l.t) .)....E.
0010  00 42 4b 33 40 00 80 06  f0 cc 9e 2a 3b 71 c6 29   .BK3@... ...*;q.)
0020  1e f1 c3 18 07 5b 36 c2  8f fc 43 ef ee 54 50 18   .....[6. ..C..TP.
0030  01 03 d8 32 00 00 30 18  00 14 68 6f 6d 65 2f 67   ...2..0. ..home/g
0040  61 72 64 65 6e 2f 66 6f  75 6e 74 61 69 6e 31 33   arden/fo untain13
```

Fig.13: PUBLISH (QoS=0) packet captured by Wireshark

3.2.4.2 Packets in publishing process with QoS=1

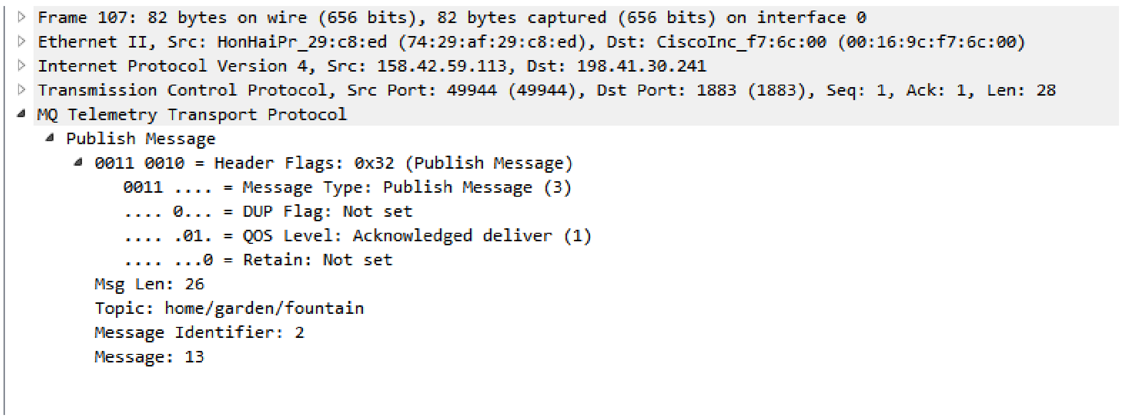
- **Publish packets with QoS=1**

The publish packet with QoS level 1 is almost the same as the packet with QoS level 0. The only field must be added is the packet identifier which is used for acknowledging the packet.



Bit		7	6	5	4	3	2	1	0	Comment
Byte 0	Fixed header	0	0	1	1	0	0	0	0	MQTT control Packet type, DUP flag, QoS level, RETAIN
Byte 1		0	0	0	1	1	0	0	0	Remaining Length
Byte 2	Variable header	0	0	0	0	0	0	0	0	Length MSB
Byte 3		0	0	0	1	0	1	0	0	Length LSB
Byte 4-23										'home/garden/fountain'
Byte 24	Packet identifier	0	0	0	0	0	0	0	0	Message Identifier 2
Byte 25		0	0	0	0	0	0	1	0	
Byte 26	Payload	0	0	1	1	0	0	0	1	'1'
Byte 27		0	0	1	1	0	0	1	1	'3'

Table 13: PUPLISH packet with QoS =1 [11].

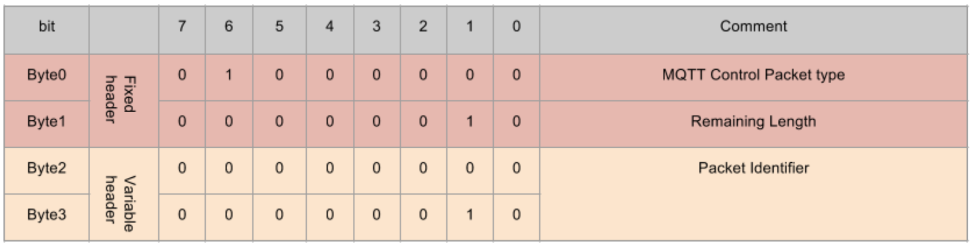


```
▷ Frame 107: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0
▷ Ethernet II, Src: HonHaiPr_29:c8:ed (74:29:af:29:c8:ed), Dst: CiscoInc_f7:6c:00 (00:16:9c:f7:6c:00)
▷ Internet Protocol Version 4, Src: 158.42.59.113, Dst: 198.41.30.241
▷ Transmission Control Protocol, Src Port: 49944 (49944), Dst Port: 1883 (1883), Seq: 1, Ack: 1, Len: 28
◢ MQ Telemetry Transport Protocol
   ◢ Publish Message
      ◢ 0011 0010 = Header Flags: 0x32 (Publish Message)
           0011 .... = Message Type: Publish Message (3)
           .... 0... = DUP Flag: Not set
           .... .01. = QOS Level: Acknowledged deliver (1)
           .... ...0 = Retain: Not set
        Msg Len: 26
        Topic: home/garden/fountain
        Message Identifier: 2
        Message: 13
```

Fig. 14: Capture of Publish (QoS=1) packet by Wireshark.

- **Puback**

The receiver must send a PUBACK, if it has received the publish message with QoS level 1 and the sender will store the message until it gets the puback packet. If the sender doesn't receive the acknowledge in a reasonable amount of time, it will send the same PUBLISH message one more time. If a sender receives the PUBACK message, it will compare the packet identifiers in publish packet and puback packet to ensure the arrive of publish message. Table 14 shows us the structure of PUBACK [11].



bit		7	6	5	4	3	2	1	0	Comment
Byte0	Fixed header	0	1	0	0	0	0	0	0	MQTT Control Packet type
Byte1		0	0	0	0	0	0	1	0	Remaining Length
Byte2	Variable header	0	0	0	0	0	0	0	0	Packet Identifier
Byte3		0	0	0	0	0	0	1	0	

Table 14: PUBACK Packet [11]

The packet only contains two headers, the fixed header and variable header. The fixed header defines the type of packet and remaining length. In PUBACK packet variable header only have 2 bytes, so the value of remaining length is 2. The variable header contains the identifier of publish packet [11].

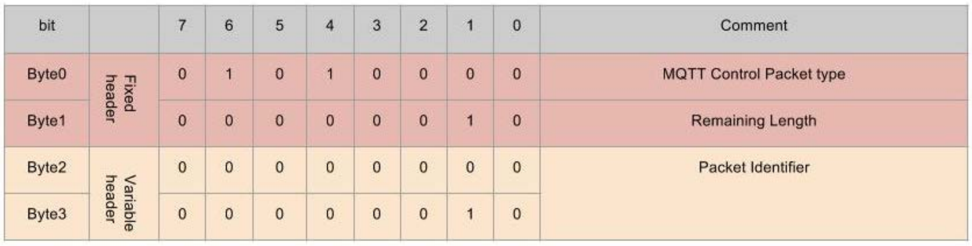
3.2.4.3 Packets in publishing process with QoS=2

- **PUBLISH packet with QoS level 2**

The publish message with QoS level 2 is the same as the message with QoS level 1

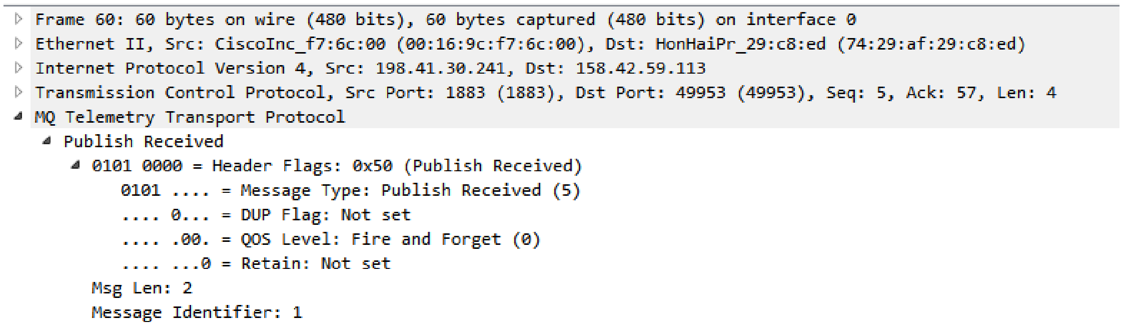
- **PUBREC- Publish received**

The publish received message is the same as the puback message in structure and only changes the MQTT control packet type code to 2#0101.



bit		7	6	5	4	3	2	1	0	Comment
Byte0	Fixed header	0	1	0	1	0	0	0	0	MQTT Control Packet type
Byte1		0	0	0	0	0	0	1	0	Remaining Length
Byte2	Variable header	0	0	0	0	0	0	0	0	Packet Identifier
Byte3		0	0	0	0	0	0	1	0	

Table 15: PUBREC packet [11]

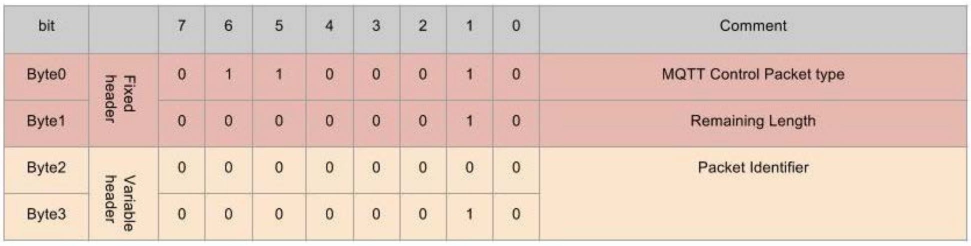


```
▷ Frame 60: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▷ Ethernet II, Src: CiscoInc_f7:6c:00 (00:16:9c:f7:6c:00), Dst: HonHaiPr_29:c8:ed (74:29:af:29:c8:ed)
▷ Internet Protocol Version 4, Src: 198.41.30.241, Dst: 158.42.59.113
▷ Transmission Control Protocol, Src Port: 1883 (1883), Dst Port: 49953 (49953), Seq: 5, Ack: 57, Len: 4
◢ MQ Telemetry Transport Protocol
  ◢ Publish Received
    ◢ 0101 0000 = Header Flags: 0x50 (Publish Received)
        0101 .... = Message Type: Publish Received (5)
        .... 0... = DUP Flag: Not set
        .... .00. = QOS Level: Fire and Forget (0)
        .... ...0 = Retain: Not set
      Msg Len: 2
      Message Identifier: 1
```

Fig. 15: Capture of PUBREC by Wireshark.

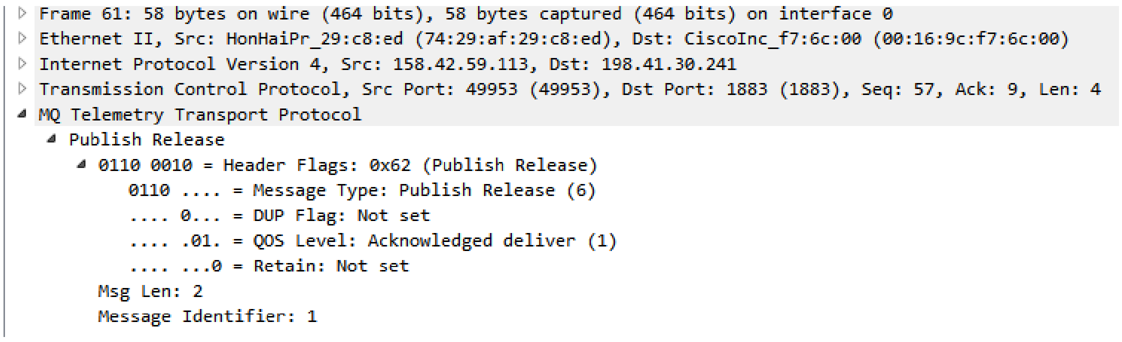
- **PUBREL - Publish release**

The publish release message is the same as the puback message in structure and only changes the first byte code to 2#0110 0010.



bit		7	6	5	4	3	2	1	0	Comment
Byte0	Fixed header	0	1	1	0	0	0	1	0	MQTT Control Packet type
Byte1		0	0	0	0	0	0	1	0	Remaining Length
Byte2	Variable header	0	0	0	0	0	0	0	0	Packet Identifier
Byte3		0	0	0	0	0	0	1	0	

Table 16: PUBREL packet [11]



```
▷ Frame 61: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
▷ Ethernet II, Src: HonHaiPr_29:c8:ed (74:29:af:29:c8:ed), Dst: CiscoInc_f7:6c:00 (00:16:9c:f7:6c:00)
▷ Internet Protocol Version 4, Src: 158.42.59.113, Dst: 198.41.30.241
▷ Transmission Control Protocol, Src Port: 49953 (49953), Dst Port: 1883 (1883), Seq: 57, Ack: 9, Len: 4
◢ MQ Telemetry Transport Protocol
  ◢ Publish Release
    ◢ 0110 0010 = Header Flags: 0x62 (Publish Release)
        0110 .... = Message Type: Publish Release (6)
        .... 0... = DUP Flag: Not set
        .... .01. = QOS Level: Acknowledged deliver (1)
        .... ...0 = Retain: Not set
      Msg Len: 2
      Message Identifier: 1
```

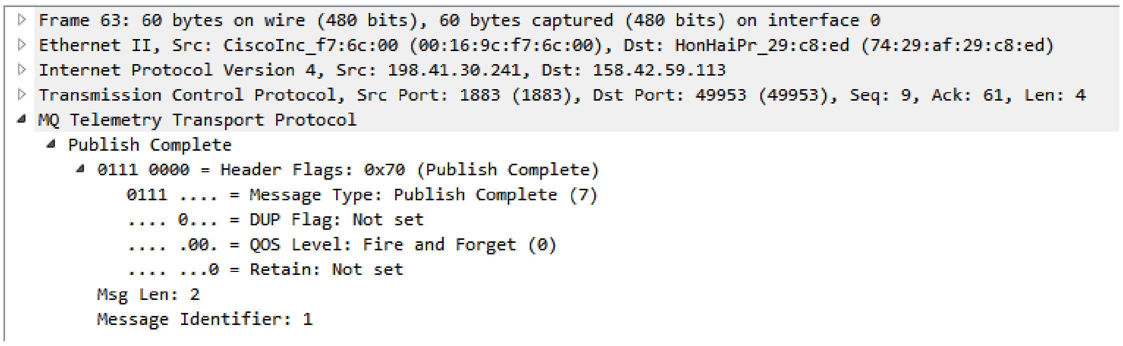
Fig. 16: Capture of PUBREL by Wireshark.

- **PUBCOMP - Publish complete**

The publish complete message is the same as the PUBACK message in structure and only changes the MQTT control packet type code to 2#0111.

bit		7	6	5	4	3	2	1	0	Comment
Byte0	Fixed header	0	1	1	1	0	0	0	0	MQTT Control Packet type
Byte1		0	0	0	0	0	0	1	0	Remaining Length
Byte2	Variable header	0	0	0	0	0	0	0	0	Packet Identifier
Byte3		0	0	0	0	0	0	1	0	

Table 17: PUBCOMP packet.



```
▷ Frame 63: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▷ Ethernet II, Src: CiscoInc_f7:6c:00 (00:16:9c:f7:6c:00), Dst: HonHaiPr_29:c8:ed (74:29:af:29:c8:ed)
▷ Internet Protocol Version 4, Src: 198.41.30.241, Dst: 158.42.59.113
▷ Transmission Control Protocol, Src Port: 1883 (1883), Dst Port: 49953 (49953), Seq: 9, Ack: 61, Len: 4
◢ MQ Telemetry Transport Protocol
   ◢ Publish Complete
      ◢ 0111 0000 = Header Flags: 0x70 (Publish Complete)
           0111 .... = Message Type: Publish Complete (7)
           .... 0... = DUP Flag: Not set
           .... .00. = QOS Level: Fire and Forget (0)
           .... ...0 = Retain: Not set
        Msg Len: 2
        Message Identifier: 1
```

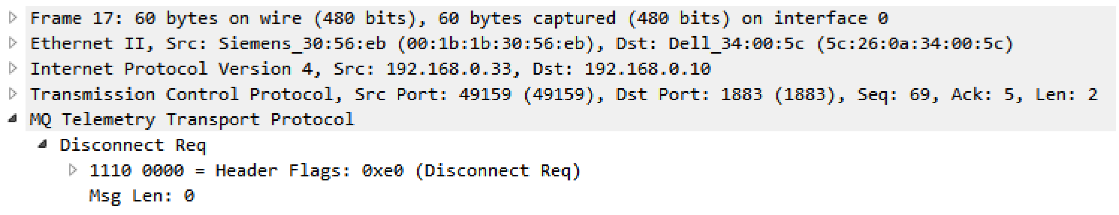
Fig. 17: Capture of PUBCOMP by Wireshark.

3.2.5 Disconnection

The disconnect packet is used for closing the connection and is sent from client to server. The figure 25 shows the structure of disconnect packet. There is only a fixed header. The remaining length value is 0, because there is no variable header.

bit		7	6	5	4	3	2	1	0	Comment
Byte0		1	1	1	0	0	0	0	0	MQTT Control Packet type
Byte1		0	0	0	0	0	0	0	0	Remaining Length

Table 18: DISCONNECT packet.



```
▷ Frame 17: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▷ Ethernet II, Src: Siemens_30:56:eb (00:1b:1b:30:56:eb), Dst: Dell_34:00:5c (5c:26:0a:34:00:5c)
▷ Internet Protocol Version 4, Src: 192.168.0.33, Dst: 192.168.0.10
▷ Transmission Control Protocol, Src Port: 49159 (49159), Dst Port: 1883 (1883), Seq: 69, Ack: 5, Len: 2
◢ MQ Telemetry Transport Protocol
   ◢ Disconnect Req
      ▷ 1110 0000 = Header Flags: 0xe0 (Disconnect Req)
        Msg Len: 0
```

Fig.18: Capture of DISCONNECT packet

3.2.6 Security

As we have mentioned before, this project is trying to connect between automation system and business network and the automation system likes a brain of factory, so the attack of the communication system can harm the machine even the operator. In other hand, the some kind of data is important and maybe we don’t want to share to the outside people, so the information security is also an aspect to be considered. The security of industrial communication is always an important aspect that should be thought before application.

Security in MQTT is divided into different layers, the network level, the transport level and the application level. On the network level the VPN provides a trusty way of connection between broker and clients. On the transport level TLS/SSL can be used to encrypt messages in order to that except both sides nobody can read the messages. On the application level the MQTT provides the authentication mechanism. In MQTT protocol, the client identifier and username/password credentials are used to authenticate the client [11].

4. Development and results

4.1 Introduction of project

In this chapter the theories of chapter 3 are applied to realize the project. Firstly, the industrial network is constituted using Profinet IO system in part 4.2. Then the MQTT protocol is implemented in PLC with AWL in part 4.3. The fig.4 shows the profile of the project which contains two parts, the industrial network and public network.

4.2 Configuration of industrial network

The following figure shows the industrial network that is designed. It contains one workstation, two controllers and a IO-Device. These devices consist in a redundancy topology which has 3 hierarchical levels the supervisory control, the automatic control and sensors and actuators.

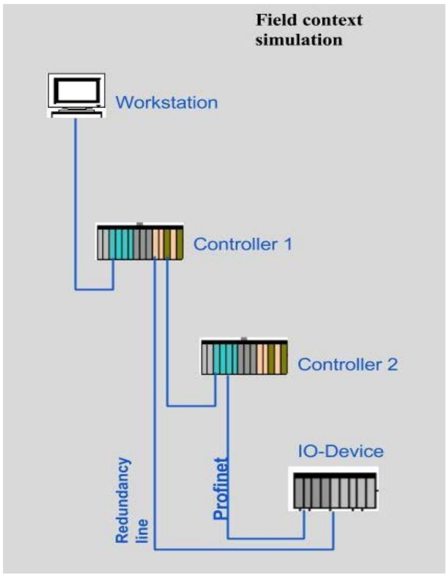


Fig. 19: Design of industrial network

4.2.1 Hardware configuration

- Device;

1. Using CPU314-2PN/DP + CP343-1 Advanced as Controller 1
2. Using CPU314-2PN/DP as Controller 2
3. Using IM-131-3PN as un Io-Device

- Configuration steps:

1. Adjust interfaz of STEP 7

   <u>Herramientas > Ajuste interfaz PG/PC</u>

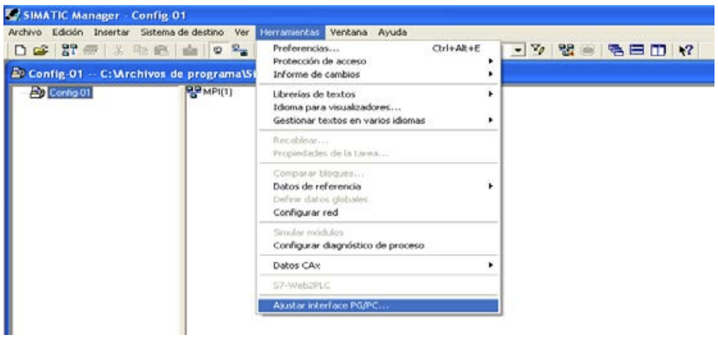
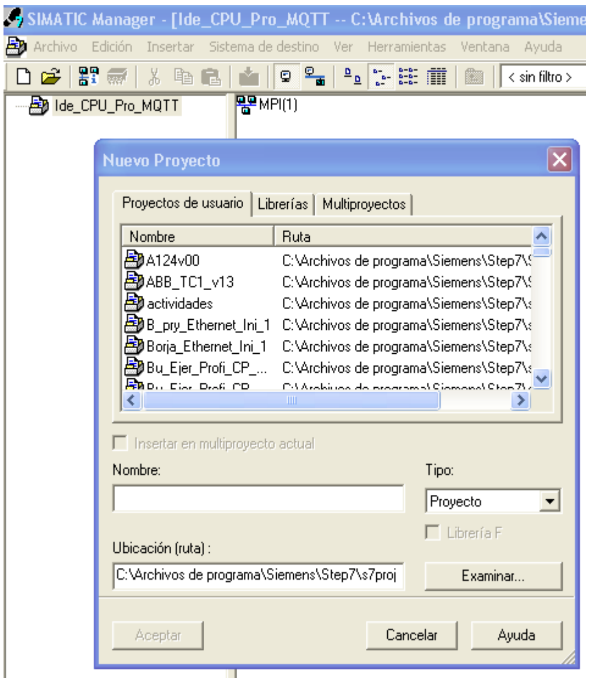

SIMATIC Manager - Config 01
Archivo Edición Insertar Sistema de destino Ver Herramientas Ventana Ayuda
Config-01 -- C:\Archivos de programa\S
Config 01
MPI(1)
Preferencias... Ctrl+Alt+E
Protección de acceso
Informe de cambios
Librerías de textos
Idioma para visualizadores...
Gestionar textos en varios idiomas
Recablear...
Propiedades de la tarea...
Comparar bloques...
Datos de referencia
Definir datos globales
Configurar red
Simular módulos
Configurar diagnóstico de proceso
Datos CAx
S7-Web2PLC
Ajustar interface PG/PC...

Fig. 20: Interface PG/PC

Here there are two kinds of the interface to be used, the Broadcom NetLink(TM) of Ethernet and CP 5512 of MPI. Each of both can be used to connect PLC and STEP 7 perfectly. Nevertheless, they are totally different communication protocols. So they use different cable and for MPI it needs an adapter. For industrial Ethernet it uses the connector of RS485 without adapter and for MPI it uses D-sub 9 pins with adapter of SIMATIC NET. Once the interface is set, it shows in the status bar.

2. Create a project named IDE_CPU_MQTT

<u>Archivo > Nuevo</u>


SIMATIC Manager - [Ide_CPU_Pro_MQTT -- C:\Archivos de programa\Sieme
Archivo Edición Insertar Sistema de destino Ver Herramientas Ventana Ayuda
< sin filtro >
Ide_CPU_Pro_MQTT
MPI(1)
Nuevo Proyecto
Proyectos de usuario
Librerías
Multiproyectos
Insertar en multiproyecto actual
Aceptar
Cancelar
Ayuda
Examinar...
Librería F

Nombre	Ruta
A124v00	C:\Archivos de programa\Siemens\Step7\
ABB_TC1_v13	C:\Archivos de programa\Siemens\Step7\
actividades	C:\Archivos de programa\Siemens\Step7\
B_pry_Ethernet_Ini_1	C:\Archivos de programa\Siemens\Step7\
Borja_Ethernet_Ini_1	C:\Archivos de programa\Siemens\Step7\
Bu_Ejer_Profi_CP_...	C:\Archivos de programa\Siemens\Step7\

Nombre:

Tipo: Proyecto

Ubicación (ruta) : C:\Archivos de programa\Siemens\Step7\s7proj

Fig.21: Creation of project/

Fill in the namespace with the name of 1IDE_CUP_MQTT. Then accept the dialog, the project will be created. The next step is to insert new object to the project.

3. Insert object

<u>Right-click the project > Insertar nuevo objeto > SIMATIC 300</u>

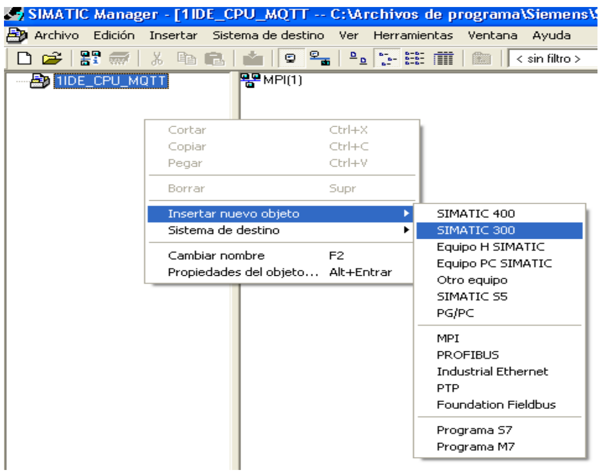


Fig. 22: SIMATIC Station

The following picture shows the project that has been created.

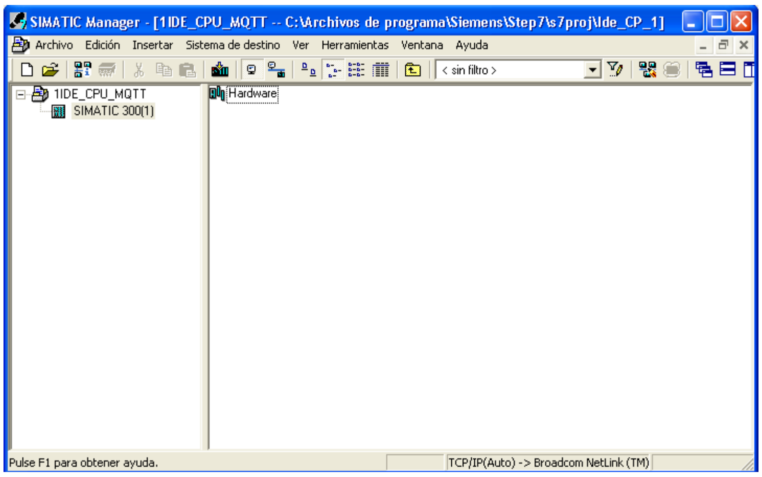

SIMATIC Manager - [1IDE_CPU_MQTT -- C:\Archivos de programa\Siemens\Step7\s7proj\Ide_CP_1]
Archivo
Edición
Insertar
Sistema de destino
Ver
Herramientas
Ventana
Ayuda
< sin filtro >
1IDE_CPU_MQTT
SIMATIC 300(1)
Hardware
Pulse F1 para obtener ayuda.
TCP/IP(Auto) -> Broadcom NetLink (TM)

Fig.23: SIMATIC Manager.

4. Insertion of the component of hardware

Double-click the hardware icon in windows, and then it shows the window of configuration of hardware. There are three parts in the window. At the upper part of left side it shows the component that is selected and installed and at the lower part of the left side it shows more detail about the component, for example, the serial number of product, the memory address etc. On the right side it is the hardware catalog. Here we can choose the component.

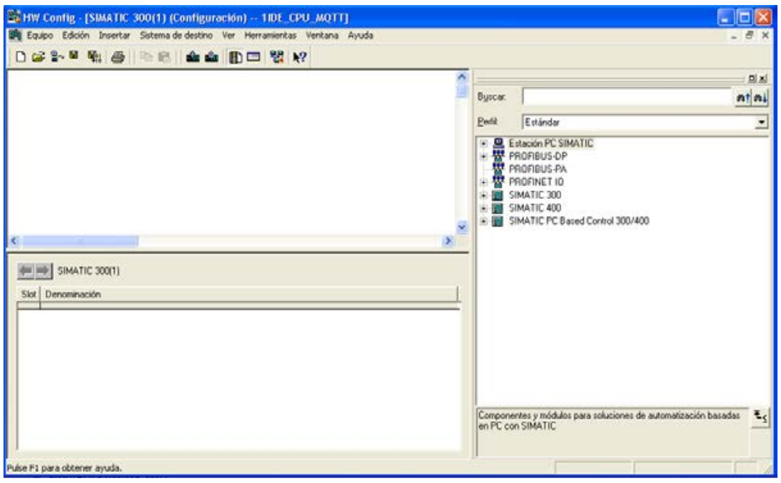

HW Config - [SIMATIC 300(1) (Configuración) -- 1IDE_CPU_MQTT]
Equipo
Edición
Insertar
Sistema de destino
Ver
Herramientas
Ventana
Ayuda
Buscar:
Perfil:
Estándar
Estación PC SIMATIC
PROFIBUS-DP
PROFIBUS-PA
PROFINET IO
SIMATIC 300
SIMATIC 400
SIMATIC PC Based Control 300/400
SIMATIC 300(1)
Slot
Denominación
Componentes y módulos para soluciones de automatización basadas en PC con SIMATIC
Pulse F1 para obtener ayuda.

Fig. 24: HW configuration window.

According to the real equipment insert the SP, CPU and other module. Usually the first component is rack and each rack has several slots to install equipment. The following picture shows the complete configuration of hardware.

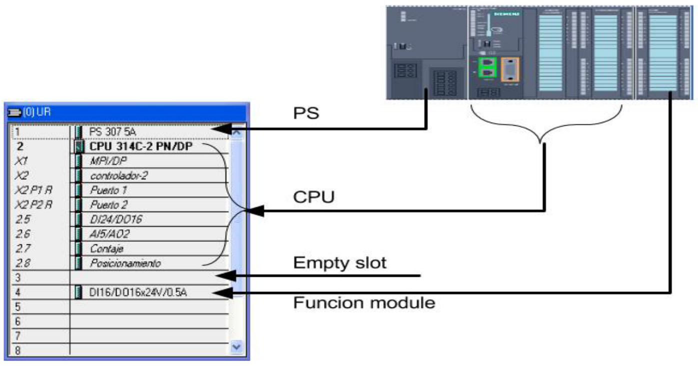


Fig. 25: HW configuration

Slot 1: The power supply
Slot 2: CPU 314C-2
Slot 3: Free slot (Reserved for the interface module)
Slot 4: IN/OUT 16 digital module

Right click on controller 2 and select the insert Profinet IO system like the following picture. The Profinet IO system connects with IO-Device.

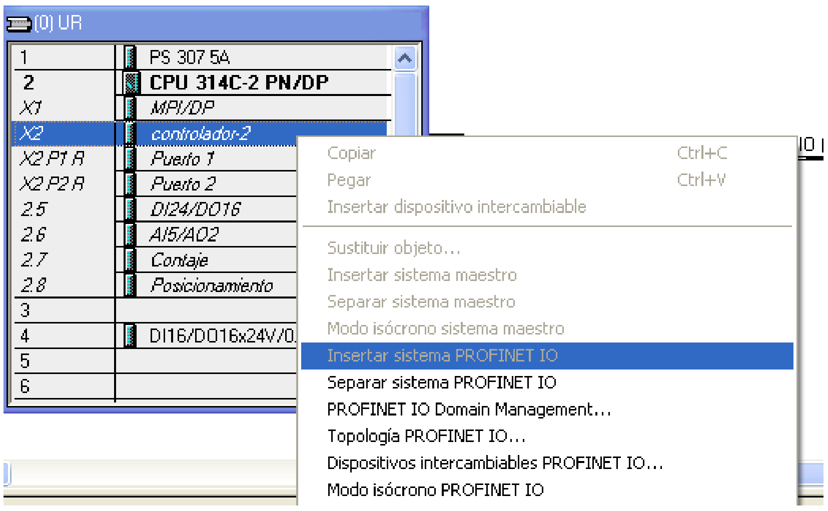
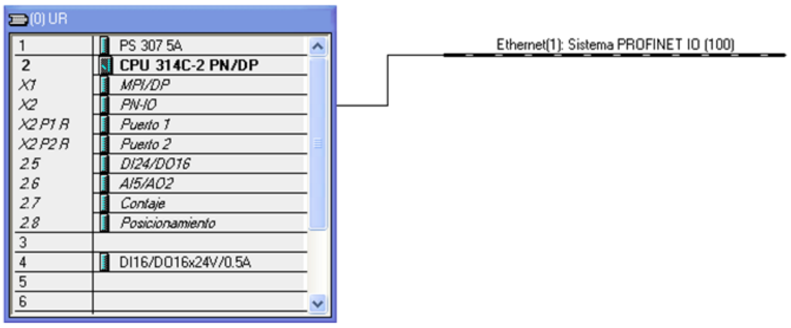


Fig.26: Add Profinet IO system.

The Profinet IO system has been created in fig.27 and if we want, the Profinet IO name can be changed.

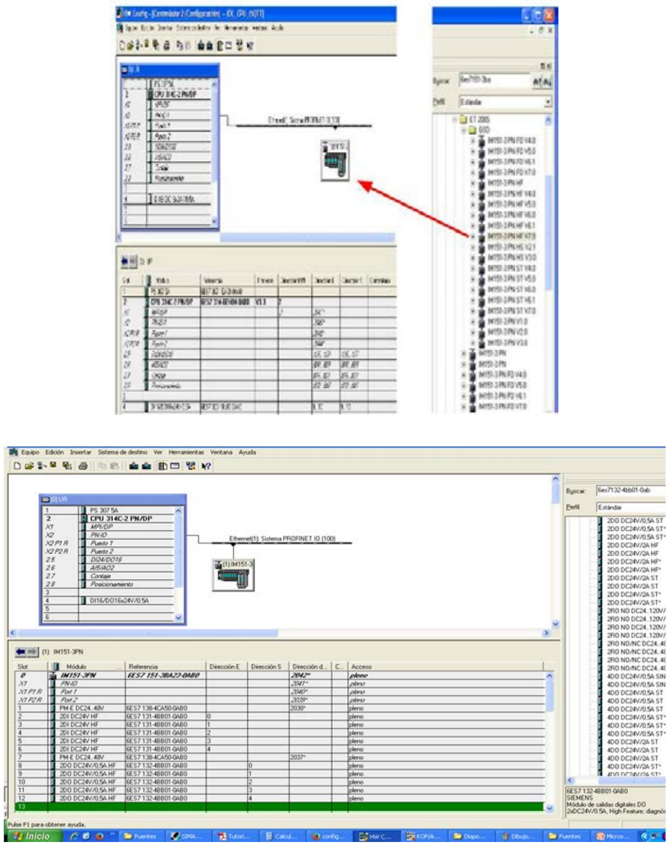


(0) UR

1	PS 307 5A
2	CPU 314C-2 PN/DP
X1	MPI/DP
X2	PN-IO
X2 P1 R	Puerto 1
X2 P2 R	Puerto 2
2.5	DI24/DO16
2.6	AI5/AO2
2.7	Contaje
2.8	Posicionamiento
3	
4	DI16/DO16x24V/0.5A
5	
6	

Fig.27: Profinet IO

Now the IM151 should be inserted into the Profinet IO system. In the hardware catalog window, select the IM151 and add it to Profinet IO system. Then drag the IO components to IM151 system like fig.28.



(1) IM151-3PN

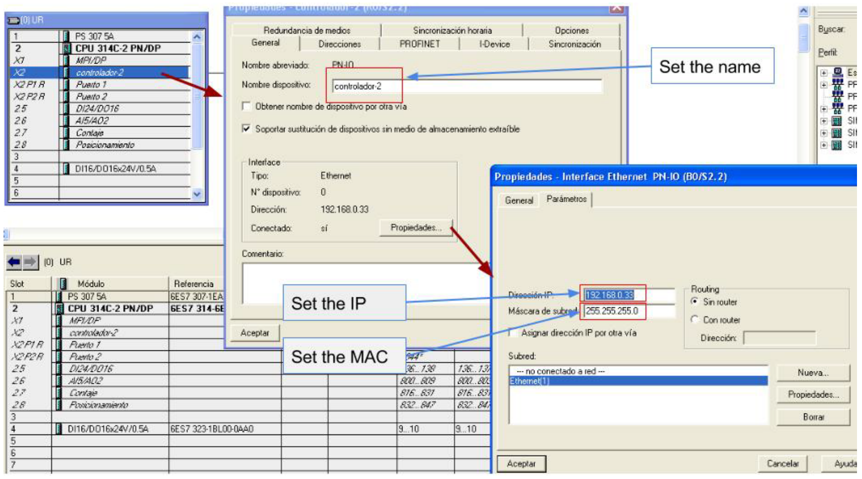
Slot	Módulo	Referencia	Dirección E	Dirección S	Dirección d...	C...	Acceso
0	IM151-3PN	6ES7 151-3BA23-0AB0			2042*		pleno
X1	PN-IO				2041*		pleno
X1 P1 R	Port 1				2040*		pleno
X1 P2 R	Port 2				2039*		pleno
1	PM-E DC24..48V	6ES7 138-4CA50-0AB0			2038*		pleno
2	2DI DC24V HF	6ES7 131-4BB01-0AB0	0				pleno
3	2DI DC24V HF	6ES7 131-4BB01-0AB0	1				pleno
4	2DI DC24V HF	6ES7 131-4BB01-0AB0	2				pleno
5	2DI DC24V HF	6ES7 131-4BB01-0AB0	3				pleno
6	2DI DC24V HF	6ES7 131-4BB01-0AB0	4				pleno
7	PM-E DC24..48V	6ES7 138-4CA50-0AB0			2037*		pleno
8	2DO DC24V/0.5A HF	6ES7 132-4BB01-0AB0		0			pleno
9	2DO DC24V/0.5A HF	6ES7 132-4BB01-0AB0		1			pleno
10	2DO DC24V/0.5A HF	6ES7 132-4BB01-0AB0		2			pleno
11	2DO DC24V/0.5A HF	6ES7 132-4BB01-0AB0		3			pleno
12	2DO DC24V/0.5A HF	6ES7 132-4BB01-0AB0		4			pleno
13							

Ethernet(1): Sistema PROFINET IO (100)
(1) IM151-3
Pulse F1 para obtener ayuda.

Fig.28: Configuration of IO-Device

6. Configuration of name and IP of controller 2

Double-click controller 2 and the next dialog windows can set the CUP name. Here the PLC is set name as controlador-2. Then click on property in this dialog IP and MAC can be set easily. If it needs a specific router, the router should be selected and the direction should be filled in.



Propiedades - controlador-2 (R0/S2.2)

General | Direcciones | PROFINET | I-Device | Sincronización | Redundancia de medios | Sincronización horaria | Opciones

Nombre abreviado: PN-IO
Nombre dispositivo: controlador-2
☐ Obtener nombre de dispositivo por otra vía
☑ Soportar sustitución de dispositivos sin medio de almacenamiento extraíble

Interface
Tipo: Ethernet
N° dispositivo: 0
Dirección: 192.168.0.33
Conectado: sí

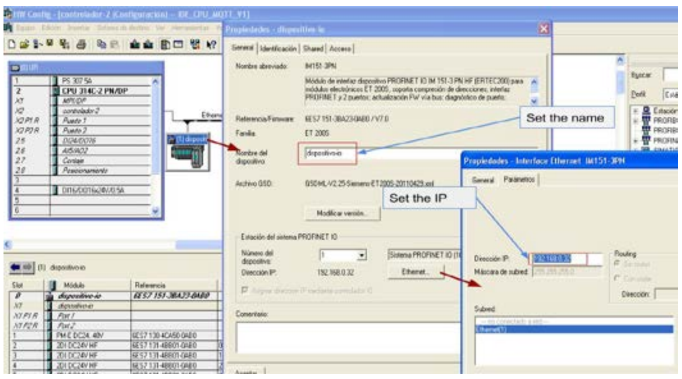
Propiedades - Interface Ethernet PN-IO (R0/S2.2)

Dirección IP: 192.168.0.33
Máscara de subred: 255.255.255.0
Routing: (●) Sin router ( ) Con router
Subred: --- no conectado a red --- / Ethernet(1)

Slot	Módulo	Referencia
1	PS 307 5A	6ES7 307-1EA
2	CPU 314C-2 PN/DP	6ES7 314-6E
X1	MPI/DP	
X2	controlador-2	
X2 P1 R	Puerto 1	
X2 P2 R	Puerto 2	
2.5	DI24/DO16	
2.6	AI5/AO2	
2.7	Contaje	
2.8	Posicionamiento	
3		
4	DI16/DO16x24V/0.5A	6ES7 323-1BL00-0AA0
5		
6		
7		

Fig. 29 : Configuration of the name and IP of controller 2.

For the IM151 firstly click on the icon of component. Then in the dialog window we can set name and go on to click on the property to set IP or Router like before.



Propiedades - dispositivo-io

Nombre abreviado: IM151-3PN
Referencia/Firmware: 6ES7 151-3BA23-0AB0 / V7.0
Familia: ET 200S
Nombre del dispositivo: dispositivo-io
Dirección IP: 192.168.0.32

Fig. 30: Configuration of the name and IP of IO-Device.

7. Then the changes configuration should be saved and compiled. If there is no problem of configuration, download the configuration to the PLC2.

Until now the configuration of fig. 31 has been finished.

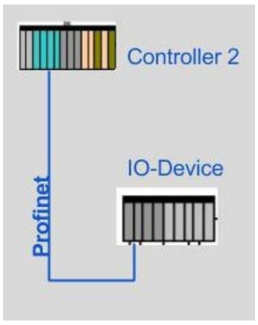


Fig. 31: Connection of controller 2 and IO-Device

8. Add another station of SIMATIC 300

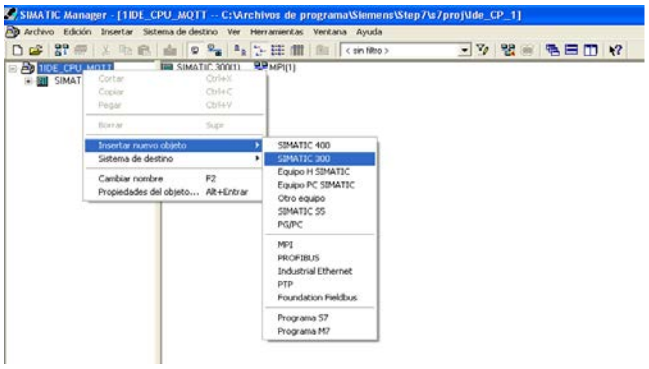
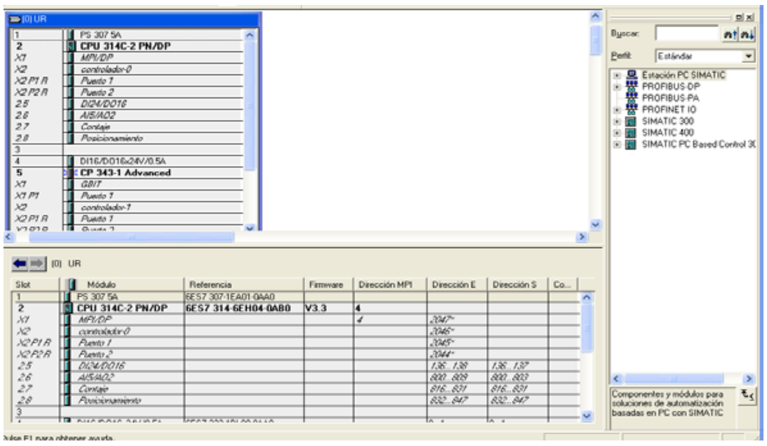


Fig. 32: Add station of SIMATIC 300

9. Hardware configuration of controller 1

The processes are the same as controller 1. The only component that should be added is CP 343 module. The following figure shows the complete configuration of controller 1.


(0) UR
Buscar:
Perfil: Estándar
Estación PC SIMATIC
PROFIBUS-DP
PROFIBUS-PA
PROFINET IO
SIMATIC 300
SIMATIC 400
SIMATIC PC Based Control 3
Componentes y módulos para soluciones de automatización basadas en PC con SIMATIC
Pulse F1 para obtener ayuda.

1	PS 307 5A
2	**CPU 314C-2 PN/DP**
X1	*MPI/DP*
X2	*controlador-0*
X2 P1 R	*Puerto 1*
X2 P2 R	*Puerto 2*
2.5	*DI24/DO16*
2.6	*AI5/AO2*
2.7	*Contaje*
2.8	*Posicionamiento*
3	
4	DI16/DO16x24V/0.5A
5	**CP 343-1 Advanced**
X1	*GBIT*
X1 P1	*Puerto 1*
X2	*controlador-1*
X2 P1 R	*Puerto 1*

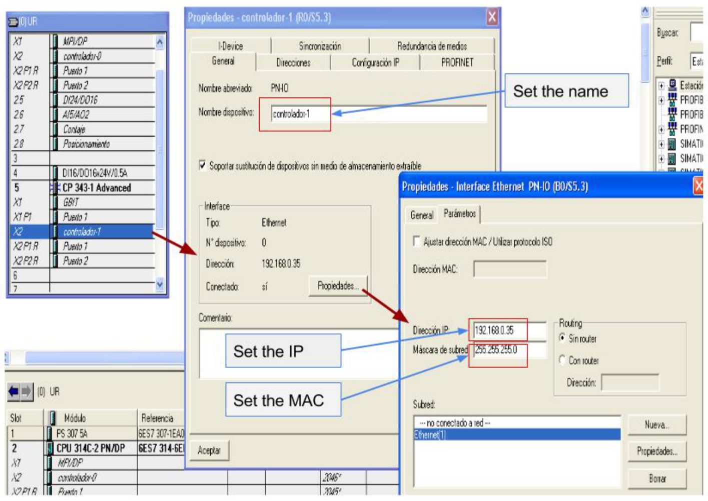
(0) UR

Slot	Módulo	Referencia	Firmware	Dirección MPI	Dirección E	Dirección S	Co...
1	PS 307 5A	6ES7 307-1EA01-0AA0					
2	**CPU 314C-2 PN/DP**	**6ES7 314-6EH04-0AB0**	**V3.3**	**4**			
X1	*MPI/DP*			*4*	*2047**		
X2	*controlador-0*				*2046**		
X2 P1 R	*Puerto 1*				*2045**		
X2 P2 R	*Puerto 2*				*2044**		
2.5	*DI24/DO16*				*136...138*	*136...137*	
2.6	*AI5/AO2*				*800...809*	*800...803*	
2.7	*Contaje*				*816...831*	*816...831*	
2.8	*Posicionamiento*				*832...847*	*832...847*	
3							

Fig. 32: Hardware configuration of controller 1

10. Setting IP in PLC1

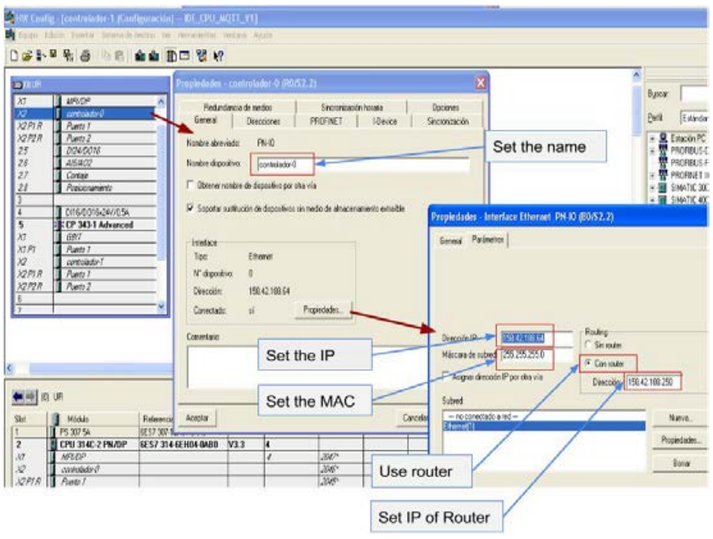
In PLC1 the communication module CP343-1 will be used for consisting in a redundancy circle. Click on c1 in module CP434-1. The configuration is as the same as before. There are 3 processes, set name, IP controller1 and Mac. In the following picture show the steps.



Nombre abreviado: PN-IO
Nombre dispositivo: controlador-1
Dirección IP: 192.168.0.35
Máscara de subred: 255.255.255.0
Routing: Sin router

Fig. 33: Configuration of IP and name in PLC1

In the CPU module the IP, MAC and name are also need to be set. But because we will use a specific router to connect to the server, the router selection should be selected and the router address should be filled.



Nombre abreviado: PN-IO
Nombre dispositivo: controlador-0
Dirección IP: 150.42.100.64
Máscara de subred: 255.255.255.0
Routing: Con router
Dirección: 150.42.100.250

Fig. 33: Configuration of IP and name in PLC1 with specific router

11. Port connection

Hardware configuration must correspond to the real equipment. If not, the system will show the system error. Fig.34 shows the hardware that will be configurated. The port 1 of controller 1 connects to port 1 of IO-Device, the port 2 of controller 1 connects to port 1 of controller 2 and the port 2 of controller 2 connects to the port 2 of IO-Device. Finally, the cables form a redundancy ring.

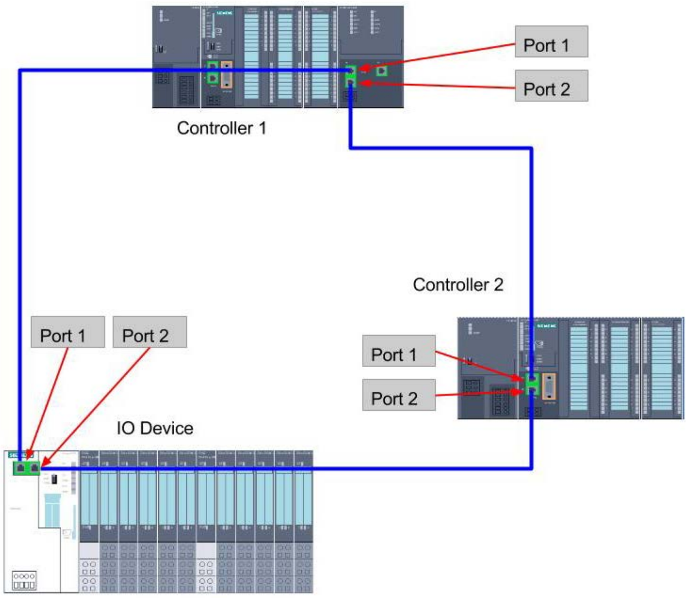
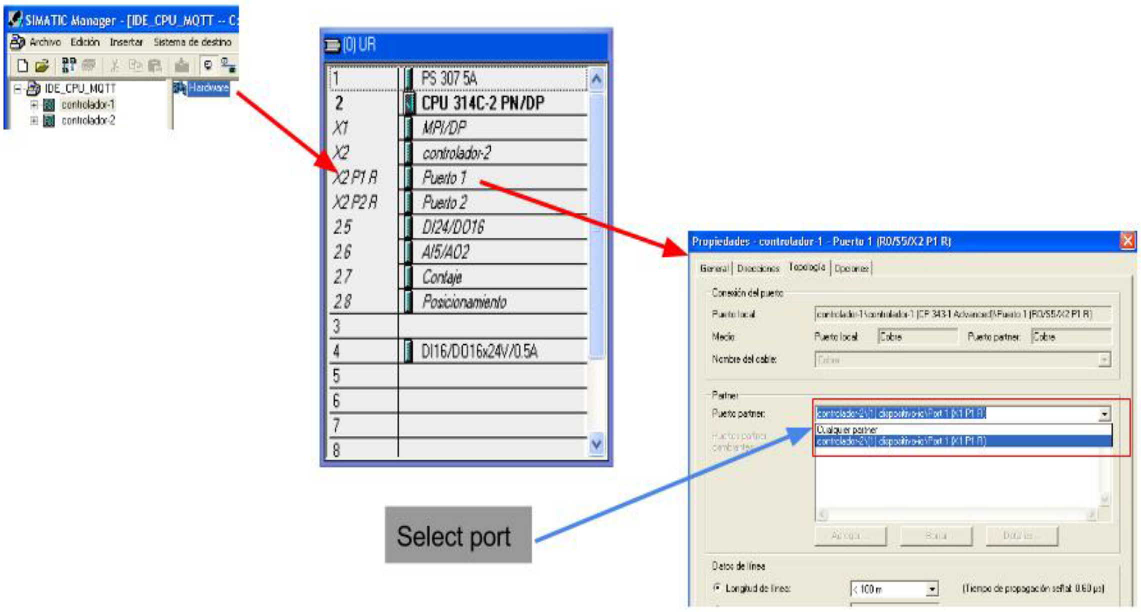


Fig. 34: Connection of ports

To connect the port firstly the hardware configuration window should be opened and double-click the port that we want to configure. The following figure shows how to configure the port 1 of controller 1. In the dialog of property of Port1, the port partner offers us the possible port that we can choose.


SIMATIC Manager - [IDE_CPU_MQTT -- C:

Archivo Edición Insertar Sistema de destino

IDE_CPU_MQTT
controlador-1
controlador-2
Hardware

(0) UR

1	PS 307 5A
2	CPU 314C-2 PN/DP
X1	MPI/DP
X2	controlador-2
X2 P1 R	Puerto 1
X2 P2 R	Puerto 2
2.5	DI24/DO16
2.6	AI5/AO2
2.7	Contaje
2.8	Posicionamiento
3	
4	DI16/DO16x24V/0.5A
5	
6	
7	
8	

Propiedades - controlador-1 - Puerto 1 (R0/S5/X2 P1 R)

General | Direcciones | Topología | Opciones

Conexión del puerto
Puerto local: controlador-1\controlador-1 (CP 343-1 Advanced)\Puerto 1 (R0/S5/X2 P1 R)
Medio: Puerto local: Cobre   Puerto partner: Cobre
Nombre del cable: Cobre

Partner
Puerto partner: controlador-2\(1) dispositivo-io\Port 1 (X1 P1 R)
Cualquier partner
controlador-2\(1) dispositivo-io\Port 1 (X1 P1 R)
Puertos partner cambiantes

Datos de línea
Longitud de línea: < 100 m   (Tiempo de propagación señal: 0.60 µs)

Fig. 35: Selection of port

4.2.2. Software configuration of industrial network

Until now the hardware configuration has been finished and software configuration will be started. The software configuration determines the communication and functionality of system automation. In this network MRP is used for enhancing the security of communication line and enlace is used for supporting the communication between two controllers. Fig.36 displays the network that should be configured.

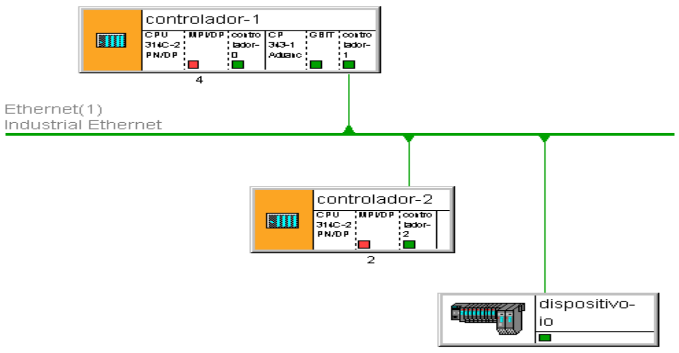
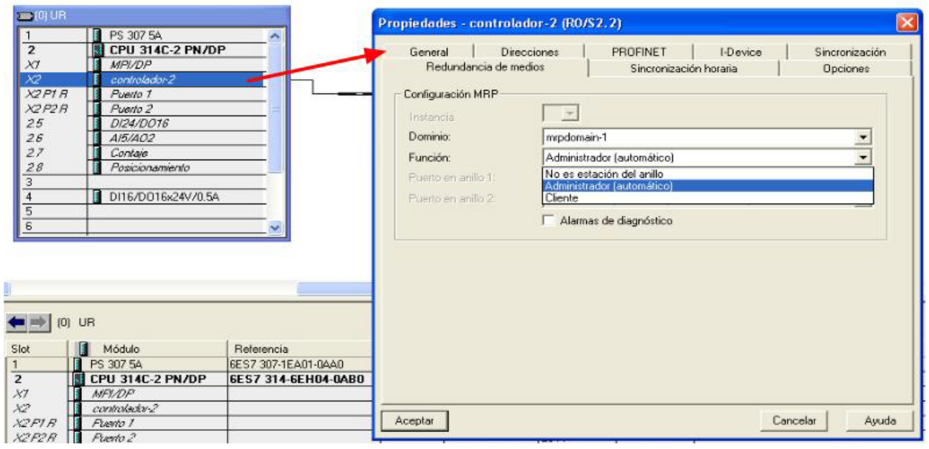


Fig. 36: Software configuration of industrial network.

4.2.2.1 Configuration of redundancy ring

One redundancy ring needs an administrator and others are clients. In this case the controller 2 is used as administer. The following configures show the configuration of different CUPs.

In controller 1 and 2：



(0) UR

1	PS 307 5A
2	CPU 314C-2 PN/DP
X1	MPI/DP
X2	controlador-2
X2 P1 R	Puerto 1
X2 P2 R	Puerto 2
2.5	DI24/DO16
2.6	AI5/AO2
2.7	Contaje
2.8	Posicionamiento
3	
4	DI16/DO16x24V/0.5A
5	
6	

(0) UR

Slot	Módulo	Referencia
1	PS 307 5A	6ES7 307-1EA01-0AA0
2	CPU 314C-2 PN/DP	6ES7 314-6EH04-0AB0
X1	MPI/DP	
X2	controlador-2	
X2 P1 R	Puerto 1	
X2 P2 R	Puerto 2	

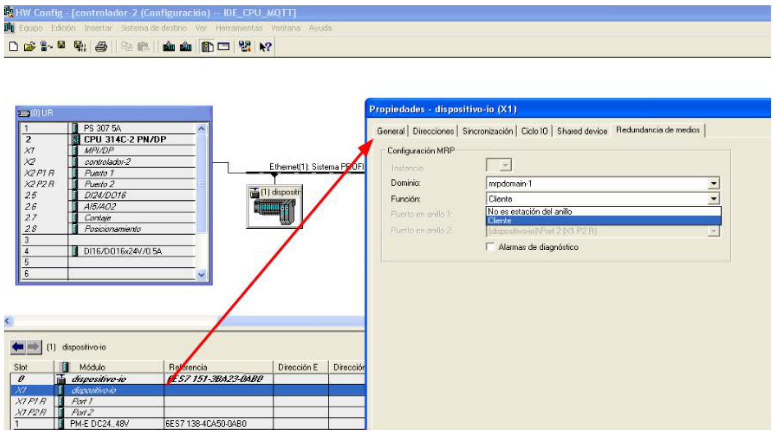
Propiedades - controlador-2 (R0/S2.2)

Redundancia de medios — Configuración MRP

Instancia:
Dominio: mrpdomain-1
Función: Administrador (automático)
Puerto en anillo 1:
Puerto en anillo 2:
Alarmas de diagnóstico

Fig. 37: Configuration of redundancy ring in controller

In IO-Device：


HW Config - [controlador-2 (Configuración) -- IDE_CPU_MQTT]
Propiedades - dispositivo-io (X1)
General
Direcciones
Sincronización
Ciclo IO
Shared device
Redundancia de medios
Ethernet(1): Sistema PROFI
(1) dispositivo-io

Configuración MRP
Dominio: mrpdomain-1
Función: Cliente
No es estación del anillo
Cliente
Puerto en anillo 2: [dispositivo-io\Port 2 (X1 P2 R)]
Alarmas de diagnóstico

Fig. 38: Configuration of redundancy ring in IO-Device

4.2.2.2 Creation of enlace between controller 1 and controller 2

Firstly open the platform of NetPro and click right of the CUP of controller 1 like the config 39.

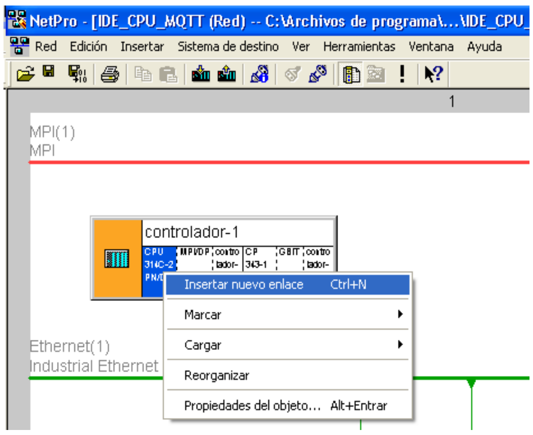
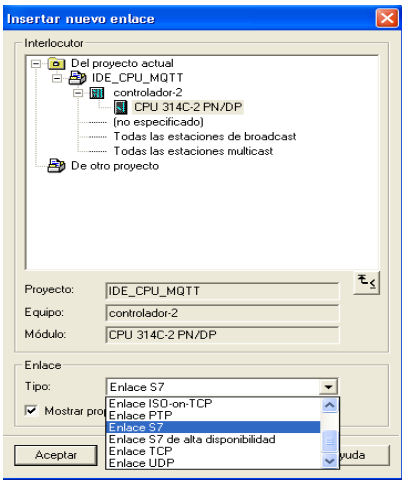


Fig. 39: Creation of new enlace

Select insertar nuevo enlace,


Insertar nuevo enlace

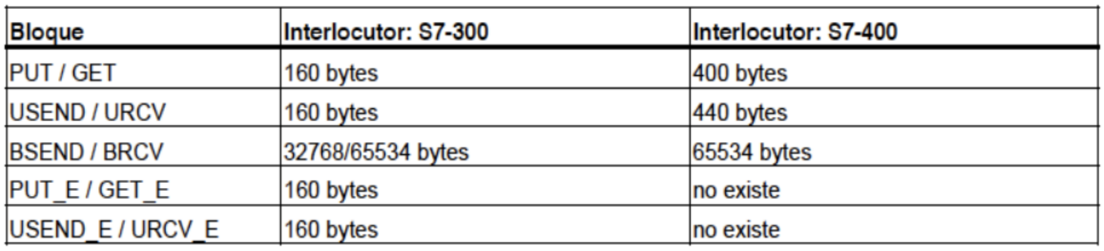
Interlocutor
- Del proyecto actual
  - IDE_CPU_MQTT
    - controlador-2
      - CPU 314C-2 PN/DP
    - (no especificado)
    - Todas las estaciones de broadcast
    - Todas las estaciones multicast
- De otro proyecto

Proyecto: IDE_CPU_MQTT
Equipo: controlador-2
Módulo: CPU 314C-2 PN/DP

Enlace
Tipo: Enlace S7
- Enlace ISO-on-TCP
- Enlace PTP
- Enlace S7
- Enlace S7 de alta disponibilidad
- Enlace TCP
- Enlace UDP

Fig. 39: Selection of type of enlace.

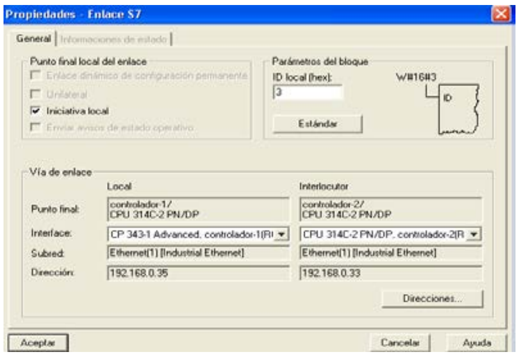
Here the enlace S7 will be selected, although there are more different enlaces, e.g. the TCP, UDP etc. Every kind of enlace can realize the communication perfectly. But each kind of enlace has its FB/FCs to use. The above figure shows the characteristics of FB/FCs in S7 enlace.



Bloque	Interlocutor: S7-300	Interlocutor: S7-400
PUT / GET	160 bytes	400 bytes
USEND / URCV	160 bytes	440 bytes
BSEND / BRCV	32768/65534 bytes	65534 bytes
PUT_E / GET_E	160 bytes	no existe
USEND_E / URCV_E	160 bytes	no existe

Table 19: FB/FC in S7

Once click the accept, the following dialog will appear.


Propiedades - Enlace S7

General | Informaciones de estado

Punto final local del enlace
- ☐ Enlace dinámico de configuración permanente
- ☐ Unilateral
- ☑ Iniciativa local
- ☐ Enviar avisos de estado operativo

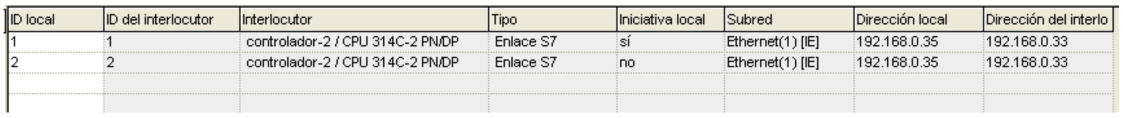
Parámetros del bloque
ID local (hex): 3
W#16#3

Vía de enlace

	Local	Interlocutor
Punto final:	controlador-1/ CPU 314C-2 PN/DP	controlador-2/ CPU 314C-2 PN/DP
Interface:	CP 343-1 Advanced, controlador-1(R	CPU 314C-2 PN/DP, controlador-2(R
Subred:	Ethernet(1) [Industrial Ethernet]	Ethernet(1) [Industrial Ethernet]
Dirección:	192.168.0.35	192.168.0.33

Fig. 40: Configration of enlace

The ID defines the enlace identifier. When the FB/FC is called to communicate, the FB/FC should know which enlace will be used. Other important selection is initiative local that defines which side will solicit enlace first. Usually the side which will send a message should be marked as an initial local and other side will be passive local. Here two enlaces are created in table 20, one for sending and the other one for receiving.



ID local	ID del interlocutor	Interlocutor	Tipo	Iniciativa local	Subred	Dirección local	Dirección del interlo
1	1	controlador-2 / CPU 314C-2 PN/DP	Enlace S7	sí	Ethernet(1) [IE]	192.168.0.35	192.168.0.33
2	2	controlador-2 / CPU 314C-2 PN/DP	Enlace S7	no	Ethernet(1) [IE]	192.168.0.35	192.168.0.33

Table 20: Information of enlace

4.2.2.3 Program for testing industrial network communication

For testing the industrial network that is configured before the program should be created in both controllers. Fig. 41 and fig. 42 display structures of the program and fig.43 present flowchart in controller 1 and 2. In controller 1 DB112 stores the data that will be sent to controller2 and DB 13 stores the date that has been received. FB8 is used for sending data and FB9 is used for receiving data. In controller 2 the program has the same structure and flowchart in terms of communication with controller1. Due to the connection with IO-Device, controller 2 should communicate with IO-Device. The IO-Device works as extended module of controller 2. The source code of AWL is appended to the chapter 8.

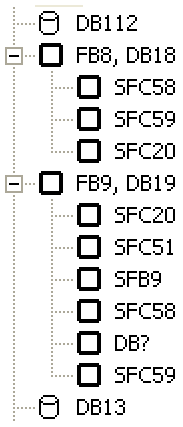


Fig. 41: Structure of program in controller 1

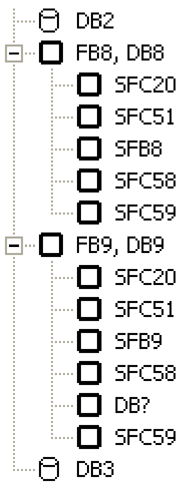


Fig. 42: Structure of program in controller 2

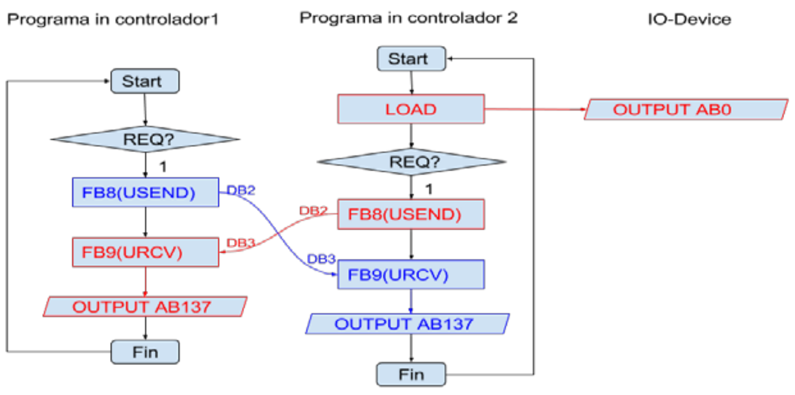
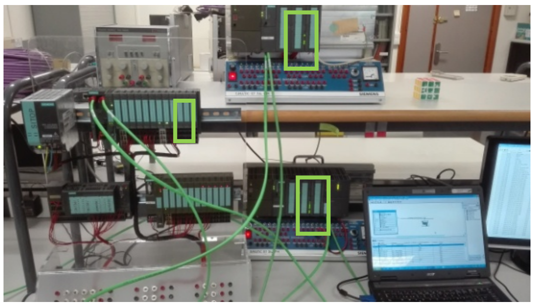
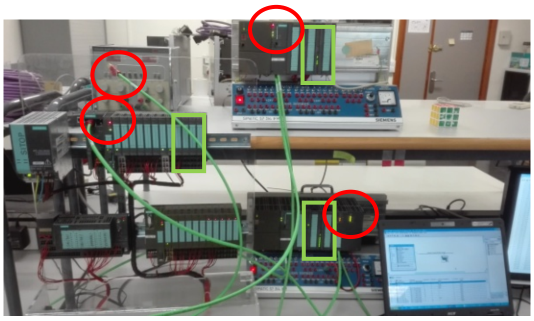
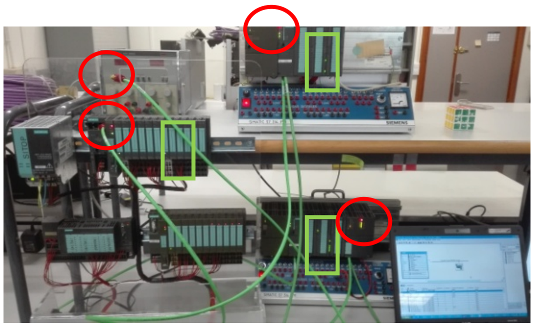


Fig 43: Flowchart of program in controller 1 and 2.

Fig. 44 represents the re result of testing**.** The first picture shows that every equipment works correctly without error; controller 1 is receiving data from controller 2; controller 2 is receiving data from 1; IO device is receiving data from controller 2. The picture 2, 3 and 4 exhibit the function of redundancy ring. E.g. the picture 2 shows the result of the system, when the connection between IO-Device and controller 2 has been removed. We can see that obviously the industrial system is still working, although it appears SF error. As the same as picture 2, picture and picture 4 show the same testing result. It reveals the configuration of industrial network is successful.


1


2


3

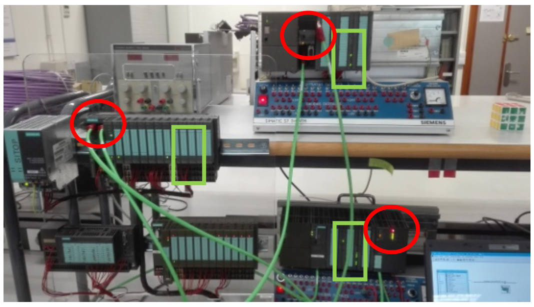

4

Fig. 44: Result of the testing of industrial network

4.3 Implementation of Communication between industrial network and MQTT broker

This section shows how to implement the communication between PLCs and server MQTT. The CPU of controller 1 is used as client and publishes the message to HIVEWQ server. Fig 25 and fig 46 demonstrate the structure and flowchart of the program. There are five FBs, FB 300 for connection, FB 301 for publishing with OoS=0, FB 301 for publishing with OoS=0, FB 301 for publishing with OoS=0 and FB 304 for disconnection. Firstly, the controller 1 sets up connection between PLCs and MQTT broker with FB300. Secondly, controller 1 publishes message according to the set of QoS level. If the signal of disconnection is active, it will close the connection. More details will be discussed in the following parts and the source code of AWL is appended to the chapter 8.

.

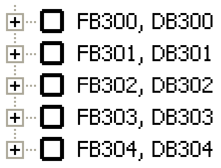


Fig45: Structure of OB1.

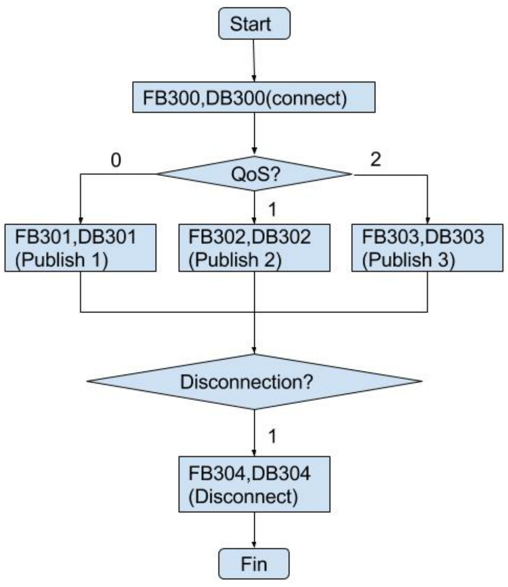


Fig. 46: Flowchart of communication between PLCs and MQTT broker.

4.3.1 Implementation of connection (FB300, DB300)

From the part 3.5 we know that in mechanism of connection the client sends a CONNECT packet and then receives a CONACK. In PLC there is no SFB/SFC for MQTT transmission, so it should be programmed and fig.47 shows the structure of functional block CONNECTS. FB1 is a functional block for constituting CON.

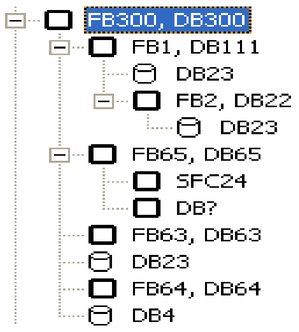


Fig.47: Structure of FB300

The following fig. reveals the mechanism of connection. Firstly, the PLC establishes the connection TCP with server. FB65 is functional block for the connection of TCP. Secondly, PLC sends a packet of CONNECT to broker. Lastly, the PLC will wait for the CONNACK packet. If all the processes have been finished without error, the connection of MQTT will be accomplished. The Fig.48 exhibits the capture a successful connection between controller 1 and MQTT broker.

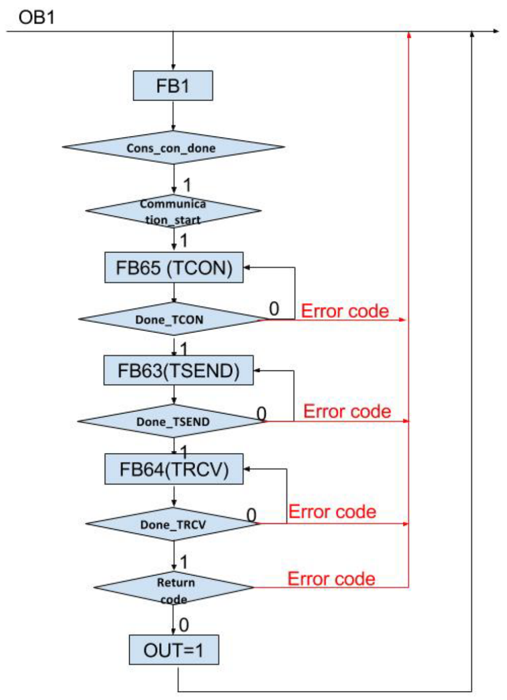
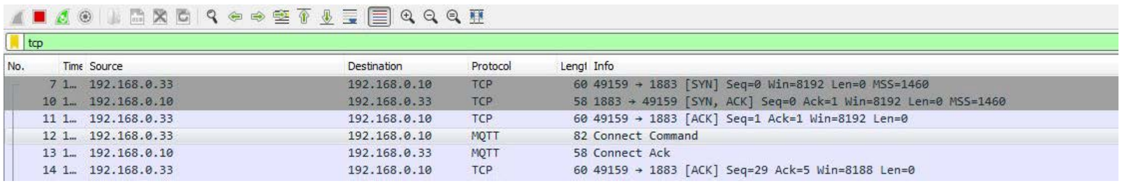


Fig.47: Flowchart of connection



```
tcp
No.    Time Source          Destination     Protocol  Lengt Info
    7 1…  192.168.0.33    192.168.0.10    TCP        60 49159 → 1883 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
   10 1…  192.168.0.10    192.168.0.33    TCP        58 1883 → 49159 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
   11 1…  192.168.0.33    192.168.0.10    TCP        60 49159 → 1883 [ACK] Seq=1 Ack=1 Win=8192 Len=0
   12 1…  192.168.0.33    192.168.0.10    MQTT       82 Connect Command
   13 1…  192.168.0.10    192.168.0.33    MQTT       58 Connect Ack
   14 1…  192.168.0.33    192.168.0.10    TCP        60 49159 → 1883 [ACK] Seq=29 Ack=5 Win=8188 Len=0
```

Fig.48: Capture of connection with Wireshark

4.3.2 Implementation of publish with QoS=0 (FB301, DB301)

When QoS = 0, it means the packet will be delivered at most once. The sender only cares about the effort but not the safety of transmission. The receiver will not respond and acknowledge to sender.

The following figure shows structure of FB301. There are only a FB3 for making publish packet DB25 and FB 63 for sending the publish packet.

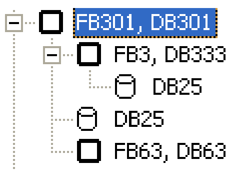


Fig.49: Structure of FB301

From fig 50 we know that firstly the FB301 calls FB3 to make DB25. If there is no problem, it calls FB63 to send the packet to server. If there are errors during these processes of sending, it will keep the error code and returns it to OB1, then the maintenance can know the error and repair it.

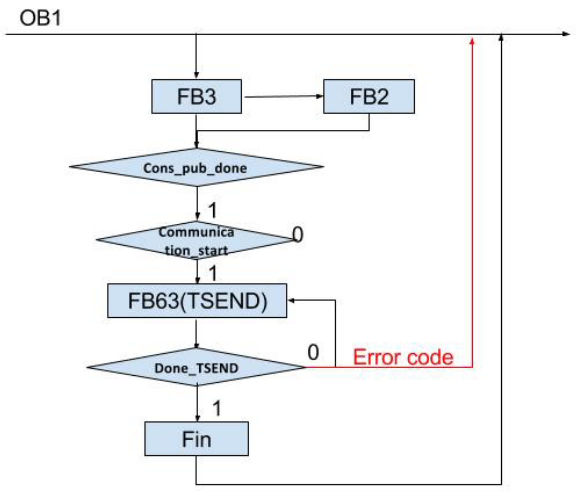
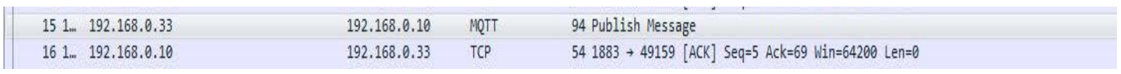


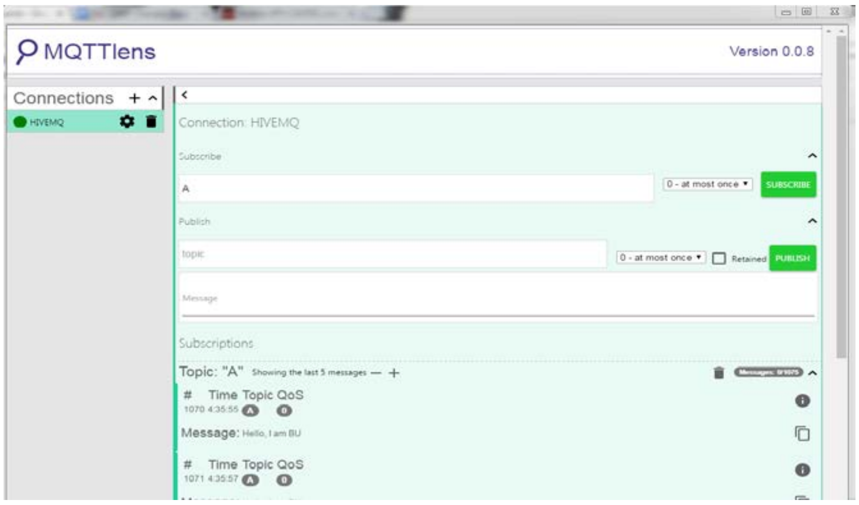
Fig.50: Flowchart of FB301

Fig 51 shows the capture of publish (QoS=0) with Wireshark. Now the automation system is connected to MQTT broker HIVEMQ and publishes messages every 2 seconds. Start up the MQTTlens and subscribe the topic “A”. Then the MQTTlens gets the message “Hello, I am BU” from MQTT broker. Fig 52 shows the experiment of publish with HIVEMQ broker. During the experiment of one week, controller 1 sends message perfectly without loss.



```
15 1… 192.168.0.33    192.168.0.10   MQTT    94 Publish Message
16 1… 192.168.0.10    192.168.0.33   TCP     54 1883 → 49159 [ACK] Seq=5 Ack=69 Win=64200 Len=0
```

Fig 51: Capture of publish (QoS=0) with Wireshark



Connection: HIVEMQ

Subscribe: A — 0 - at most once

Publish: 0 - at most once

Topic: "A" Showing the last 5 messages

#	Time	Topic	QoS
1070	4:35:55	A	0

Message: Hello, I am BU

#	Time	Topic	QoS
1071	4:35:57	A	0

Fig 52: Experiment of publish (QoS=0) with HIVEMW broker.

4.3.3 Implementation of publish with QoS=1 (FB302, DB302)

Just as it has been discussed before, when the publish packet with QoS level 1, the receiver needs to acknowledge to sender. If the first time of send has failed, the sender should send more than once to guarantee the safety of transmission. So this kind of publish is called "at least once delivery". In figure 52 shows the structure of FB302. There are FB3 for constituting DB25, FB63 for sending packet and FB64 for receiving packet.

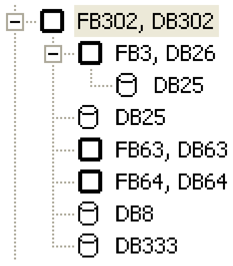
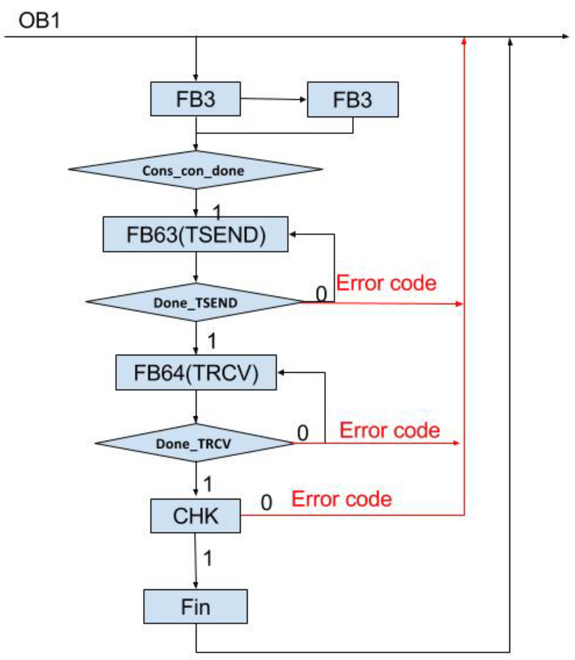


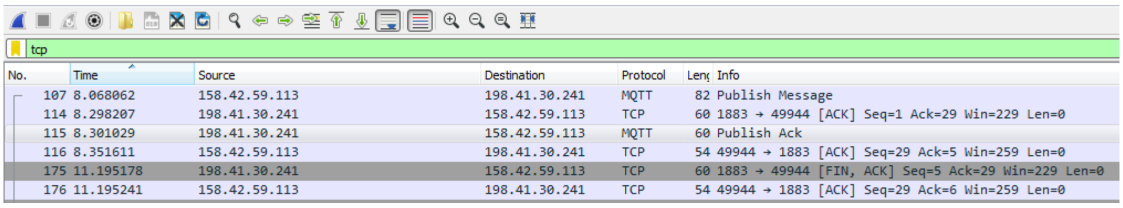
Fig 52: Structure of FB302

Fig.53 shows the mechanism of FB302. Firstly the FB3 constitutes a publish packet with QoS=1. Then FB 63 TSEND sends the packet to broker. Due to the second level of QoS, the broker will return an acknowledge PUBACK. So the FB 302 receives the PUBACK message with TRCV. When FB 302 receives the message, it should check the correction of message. If there is no error during these processes, the publish FB will be finished and go back to OB1, otherwise, the publish FB will return error code.



Fg.53: Flowchart of FB302

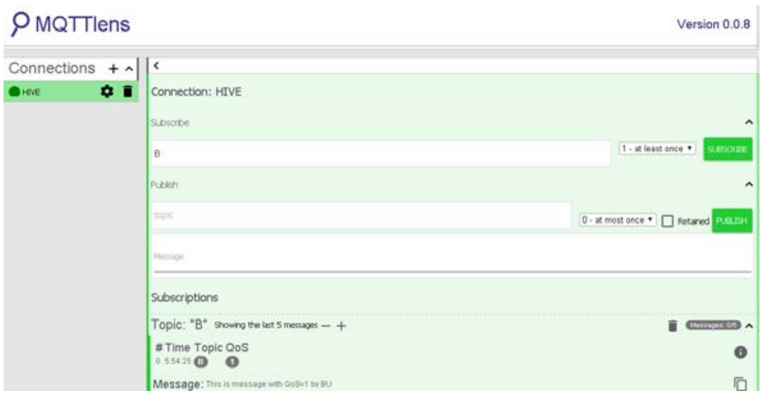
The experiment is as the same as before. Firstly Wireshark captures the data flow and check the correction of processes. Fig 54 demonstrates the result of capture. Then we connect the PLCs to HIVEMQ broker. Start up the MQTTlens and subscribe the topic “B”. Then the MQTTlens gets the message “This is message with QoS=1 by BU” from MQTT broker. Fig 55 shows the experiment of publish with HIVEMQ broker. During the experiment of one week, controller 1 sends message perfectly without lost.



tcp

No.	Time	Source	Destination	Protocol	Leng	Info
107	8.068062	158.42.59.113	198.41.30.241	MQTT	82	Publish Message
114	8.298207	198.41.30.241	158.42.59.113	TCP	60	1883 → 49944 [ACK] Seq=1 Ack=29 Win=229 Len=0
115	8.301029	198.41.30.241	158.42.59.113	MQTT	60	Publish Ack
116	8.351611	158.42.59.113	198.41.30.241	TCP	54	49944 → 1883 [ACK] Seq=29 Ack=5 Win=259 Len=0
175	11.195178	198.41.30.241	158.42.59.113	TCP	60	1883 → 49944 [FIN, ACK] Seq=5 Ack=29 Win=229 Len=0
176	11.195241	158.42.59.113	198.41.30.241	TCP	54	49944 → 1883 [ACK] Seq=29 Ack=6 Win=259 Len=0

Fig 54: Capture of PUBLISH (QoS=1) with Wireshark



Connection: HIVE

Subscribe: B — 1 - at least once

Publish: topic — 0 - at most once

Subscriptions

Topic: "B" Showing the last 5 messages

Time Topic QoS
0 5:54:25 B 1

Message: This is message with QoS=1 by BU

Fig 55: Experiment of publish (QoS=1) with HIVEMW broker.

4.3.4 Implementation of publish with QoS=2 (FB303, DB303)

This kind of publish is the safest type to transmit data and the lowest quality of service level. It will do 2 times of acknowledge. It also use FB3 to constitute publish packet, FB63 to send packet y FB 64 to receive packet. Fig 56 displays the structure of FB302.

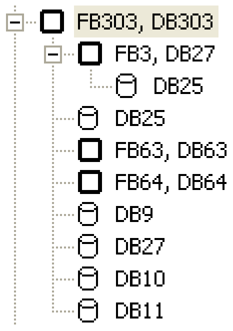


Fig 56: Structure of FB302

This kind of publish is more complicated than the others two. The first step is also to constitute a publish packet with QoS=2. Then the packet is sent to broker by TSEND. Now if the broker receives the packet, it will return a PUBREC message to sender as acknowledgment. After the sender gets the PUBREL, it compares the packet identifier of PUBREC with the packet identifier of publish packet. Until now if there is no error, the FB303 will send a PUBREL packet to MQTT Broker. After the broker gets the PUBREL, it can discard stored state and return a PUBCOMP message. Then the FB 303 receives the PUBCOMP packet with TRCV and checks its packet identifier as before. During these processes the FB303 will not discard its stored state until the second successful check of identifier, because the sender will send more time, if some packets loses.

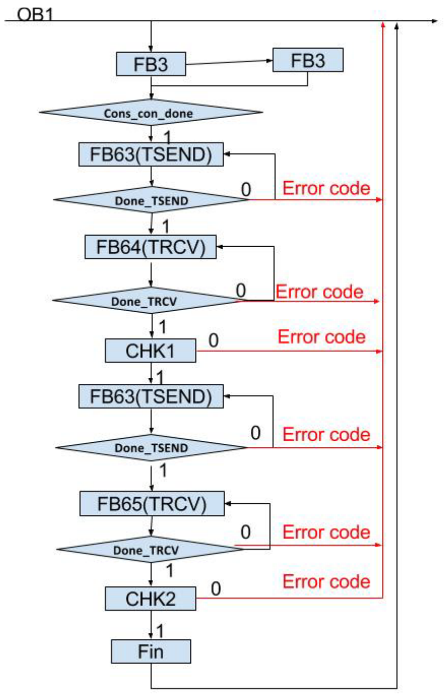
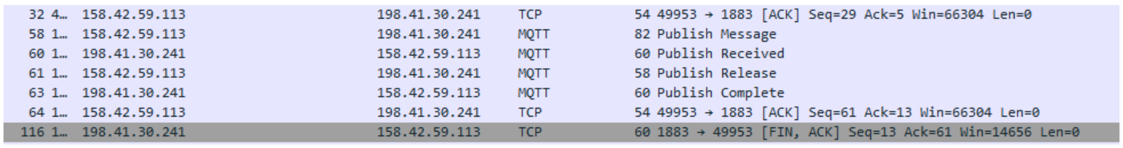


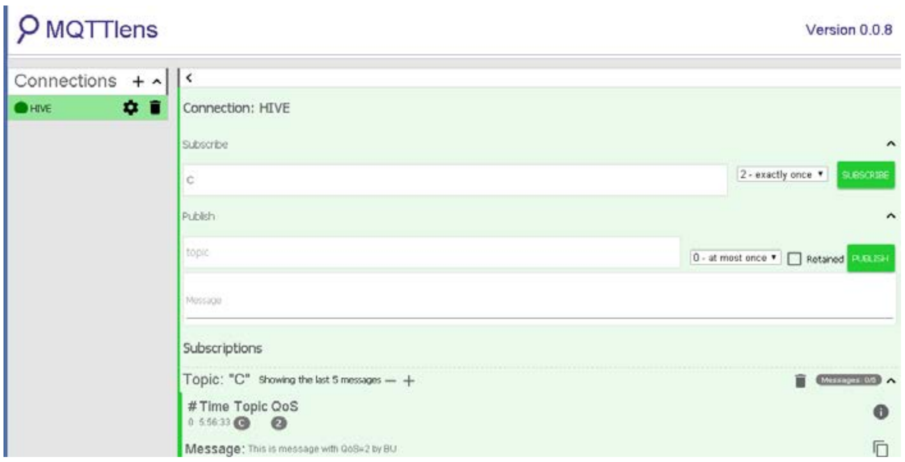
Fig.57: Flowchart of FB303

The experiment is as the same as before. Firstly Wireshark captures the data flow and check the correction of processes. Fig 58 demonstrates the result of capture. Then we connect the PLCs to HIVEMQ broker. Start up the MQTTlens and subscribe the topic “C”. Then the MQTTlens gets the message “This is message with QoS=2 by BU” from MQTT broker. Fig 59 shows the experiment of publish with HIVEMQ broker. During the experiment of one week, controller 1 sends message perfectly without loss.



```
 32 4…  158.42.59.113      198.41.30.241   TCP    54 49953 → 1883 [ACK] Seq=29 Ack=5 Win=66304 Len=0
 58 1…  158.42.59.113      198.41.30.241   MQTT   82 Publish Message
 60 1…  198.41.30.241      158.42.59.113   MQTT   60 Publish Received
 61 1…  158.42.59.113      198.41.30.241   MQTT   58 Publish Release
 63 1…  198.41.30.241      158.42.59.113   MQTT   60 Publish Complete
 64 1…  158.42.59.113      198.41.30.241   TCP    54 49953 → 1883 [ACK] Seq=61 Ack=13 Win=66304 Len=0
116 1…  198.41.30.241      158.42.59.113   TCP    60 1883 → 49953 [FIN, ACK] Seq=13 Ack=61 Win=14656 Len=0
```

Fig. 58: Capture of PUBLISH (QoS=2) with Wireshark.



Connection: HIVE

c — 2 - exactly once

0 - at most once

Topic: "C" Showing the last 5 messages

Time Topic QoS
0 5:56:33 C 2

Message: This is message with QoS=2 by BU

Fig 59: Experiment of publish (QoS=2) with HIVEMW broker.

4.3.5 Implementation of disconnection (FB304, DB304)

FB304 is used for closing network connection. It only sends a packet of disconnect to broker, so the structure is simple in fig. 60 and the fig. 61 shows the mechanism of FB304. It sends the disconnect packet to MQTT broker, then the network connection will be closed and the client will not do anything.

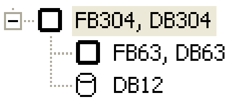


Fig.60: Structure of FB304

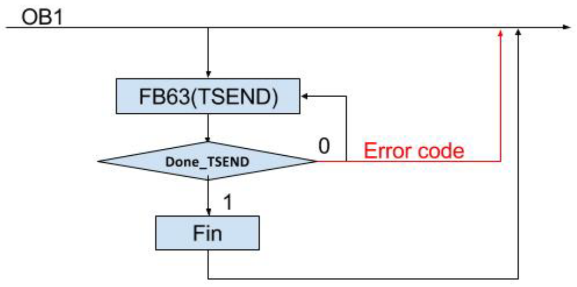
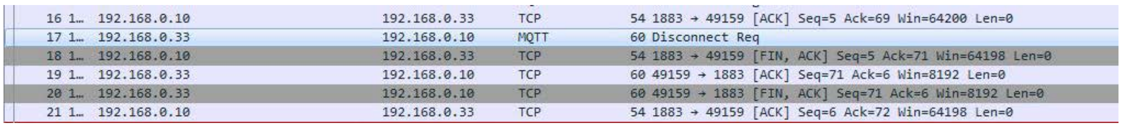


Fig.61: Flowchart of FB304



```
16 1…  192.168.0.10      192.168.0.33   TCP    54 1883 → 49159 [ACK] Seq=5 Ack=69 Win=64200 Len=0
17 1…  192.168.0.33      192.168.0.10   MQTT   60 Disconnect Req
18 1…  192.168.0.10      192.168.0.33   TCP    54 1883 → 49159 [FIN, ACK] Seq=5 Ack=71 Win=64198 Len=0
19 1…  192.168.0.33      192.168.0.10   TCP    60 49159 → 1883 [ACK] Seq=71 Ack=6 Win=8192 Len=0
20 1…  192.168.0.33      192.168.0.10   TCP    60 49159 → 1883 [FIN, ACK] Seq=71 Ack=6 Win=8192 Len=0
21 1…  192.168.0.10      192.168.0.33   TCP    54 1883 → 49159 [ACK] Seq=6 Ack=72 Win=64198 Len=0
```

Fig 62: Capture of disconnection with Wireshark.

5. Equipment used

5.1 Hardware

1. PS307 x2
2. SIEMENS PLC 300 314C-2PN/DP x2
3. DI16/DO16xDC24V x2
4. SIEMENS SITOP 120/230V-500V 50/60Hz
5. IO IM 151-3PN

5.2 Software

1. Step 7 V5.5
2. Server MQTT mosquitto.
3. MQTT lens of Google application

4. Proneta
5. SIMATIC TIA PORTAL 12.0
6. HIVEMQ MQTT server
7. Wireshark

6. Conclusion and Future work

6.1 Main Conclusions

An industrial communication system using Profinet protocol and MQTT protocol has been successfully developed in this project. The feasibility and the reliability of interconnection between industrial system and public network system using Profinet protocol and MQTT protocol have been completely proved.

6.2 Future improvements

Despite the project success, in order to offer better service it is impossible to include more improvements in the project. Firstly, in the part of automation system the following improvements can be included:

1. Optimize the algorithm to improve the performance of calculation.
2. Complete the diagnostic system to improve the reliability of operation.

In the part of business network the application MQTT lens is used during this project for subscribing. It is very simple application for testing and can´t be applied in the real industrial context. So an application of data-using will be necessary in the future. The following figure shows the application of data-using.

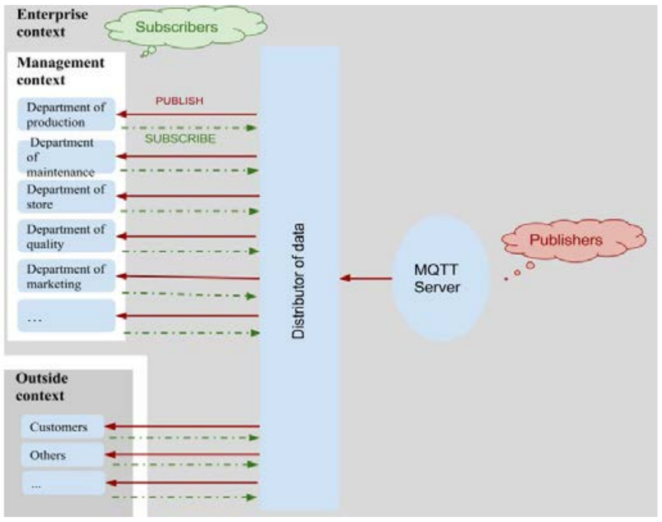


Fig.63: Application of data-using

The distributor of data is used to distribute different data to the different clients. On the one hand it can improve efficiency in the using of data, because in a company there are several departments and each department needs different information. For example, for apartment maintenance they need the data of machine. However, the quality apartment only needs product data .on the other hand it can protect important data, because with this tool the clients can only see the data that the company wants to offer.

7. Bibliography

[1] Abdu Idris Omer Taleb M.M., PhD “ARCHITECTURE OF INDUSTRIAL AUTOMATION SYSTEMS” European Scientific Journal January 2014

[2] IGOR BÉLAI, PETER DRAHOŠ “THE INDUSTRIAL COMMUNICATION SYSTEMS PROFIBUS AND PROFInet” Applied Natural Sciences 2009

[3] Rockwell Automation “Safety and Security Standards Development and Implementation”

[4] “SCADA”

https://en.wikipedia.org/wiki/SCADA#SCADA_architectures [Online].

[5] DACFEY DZUNG, MEMBER, IEEE, MARTIN NAEDELE, THOMAS P. VON HOFF, AND MARIO CREVATIN, MEMBER, IEEE "Security for Industrial Communication Systems"

[6] Thilo Sauter, Stefan Soucek, Martin Wollschlaeger, "Vertical integration"

The industrial electronics handbook industrial communication system

[7] "MQTT"

https://en.wikipedia.org/wiki/MQTT [Online].

[8] "Profinet"

http://www.rtaautomation.com/technologies/profibus/ [Online]

[9] "MQTT Essentials: Part 1 – Introducing MQTT"

http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt [Online]

[10] "MQTT Version 3.1.1"

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html [Online]

[11] Bernd Lieberth, "Figures Profibus Profinet Proflenergy"

[13] Dept. of Electron. for Autom., Brescia Univ., Italy "Experimental evaluation of Profinet performance"

[14] "PROFINET"

https://en.wikipedia.org/wiki/PROFINET [Online].

[15] "Get smart: The I-Device functionality" https://w3.siemens.com/mcms/automation/en/industrial-communications/profinet/Documents/articles/en/profinet-innovations-2010-2.html [Online].

[16] "Round and round: the Media Redundancy Protocol"https://w3.siemens.com/mcms/automation/en/industrial-communications/profinet/Documents/articles/en/profinet-innovations-2010-4.html

[17] SIEMENS "PROFINET answers for industry"

[18] SIEMENS "SIEMENS SIMATIC PROFINET System Description"

[19] "MQTT Security Fundamentals Wrap-Up"

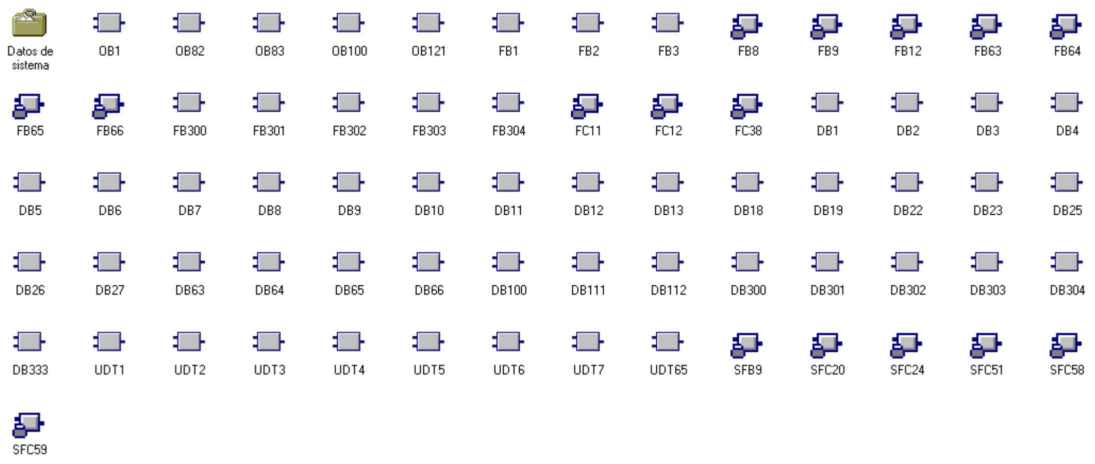
http://www.hivemq.com/blog/mqtt-security-fundamentals-wrap-up [Online]

[20] Max Felser, Paolo Ferrari, Alessandra Flammini "Profienet" The industrial electronics handbook industrial communication system

[21] Max Felser, Ron Mitchell "Profibus" The industrial electronics handbook industrial communication system

8. Appendix

8.1 Program in controller 1



OB1:

```
//Communication between  Controlador 1
y 2
    L     MB    10
    T     "Send to Co2".E
    UN    M     42.0
    L     S5T#10MS
    SE    T      0
    U     T      0
    =     M     42.0
// PULSE to active REQ

    CALL  FB     8 , DB18
// Enviar DB112 a controlador2
     REQ   :=M42.0
     ID    :=W#16#1
     R_ID  :=DW#16#1
     DONE  :=M40.0
     ERROR :=M40.1
     STATUS:=MW43
     SD_1  :=P#DB112.DBX0.0 BYTE 2
    CALL  FB     9 , DB19
     EN_R  :=TRUE
     ID    :=W#16#2
     R_ID  :=DW#16#2
     NDR   :=M40.2
     ERROR :=M40.3
     STATUS:=MW45
     RD_1  :=P#DB13.DBX0.0 BYTE 2

    L     "Receive from Co2".G
    T     AB   137

// Communication of MQTT
//--------------------------------------------------
//Reset system
    U     E    136.1
    R     "PUBLISH0_DONE"
    U     E    136.1
    R     "CON_MQTT_DONE"
    U     E    136.1
    R     "PUBLISH1_DONE"
    U     E    136.1
    R     "PUBLISH2_DONE"
    U     E    136.1
    R     "DISCONNECT_DONE"
    U     E    136.1
    R     "conCleanFlag"
    U     E    136.1
    R     "pubCleanFlag"
    U     E    136.1
    R     "Cons_con_done"
    U     E    136.1
    R     "Cons_pub_done"

//--------------------------------------------------
//-CONNECT_MQTT-------
    U     "CON_MQTT_DONE"
    U     "CON_MQTT_DONE"
    SPB   SEL
// If the connection is done, it will jump to
publish block

    U     M     10.7
    FP    "Detecta_start"
    =     "CON_start"

     CALL  "FB300" , DB300
     INIT_COM:="CON_start"
     ABORT   :=E136.1
     FIN     :="Detect_CON_FIN"
     FAILED  :="Failed_Conne"
     FA_CODE :="CON_FA_CODE"
     ER_CODE :="CON_RR_CODE"

    U     "Detect_CON_FIN"
    O     "CON_MQTT_DONE"
    =     "CON_MQTT_DONE"

    U     "CON_MQTT_DONE"
    SPB   SEL
    BEA

//--------------------------------------------------
SEL:  L     2#1100000000000000
    L     EW   136
    UW
    SRW   14
    T     "Type_PUB"                  // Sacar
E136.7 y E136.8
    L     "Type_PUB"
    SPL   ERR
    SPA   PUB0
    SPA   PUB1
    SPA   PUB2
    SPA   PUB3

//--------------------------------------------------
//-PUBLIS_MQTT-----------------------------

    ERR:  SPA   FIN
    PUB0: SPA   FIN
    PUB1: U     E    136.1
    SPB   DIS
    U     M     10.7
    FP    "Detecta_start"
    =     "pubStart"

    CALL  FB   301 , DB301
     STA    :="pubStart"
     FIN    :="PUBLISH0_DONE"
     FAILED :="PUB_FAI"
     ER_CODE:="PUB_RR_CODE"
    BEA

//--------------------------------------------------
//-----------SoQ=1-------
    PUB2: U     E    136.1
    SPB   DIS
    U     M     10.7
    FP    "Detecta_start"
    =     "pubStart"

    CALL  FB   302 , DB302
     STA    :="pubStart"
     FIN    :="PUBLISH1_DONE"
     FAILED :="PUB_FAI"
```

```
      FA_CODE:="PUB_FA_CODE"
      RR_CODE:="PUB_RR_CODE"
      BEA

//-----------------------------------------------
//---------SoQ=2---------
     PUB3: U     E    136.1
     SPB   DIS
     U     M     10.7
     FP    "Detecta_start"
     =     "pubStart"

     U     "PUBLISH2_DONE"
     CALL  FB   303 , DB303
      STA    :="pubStart"
      FIN    :="PUBLISH2_DONE"
      FAILED :="PUB_FAI"
      FA_CODE:="PUB_FA_CODE"
      RR_CODE:="PUB_RR_CODE"
      BEA

//-----------------------------------------------
//-----DISCONNECT_MQTT---
     DIS:  CLR
     U     "DISCONNECT_DONE"
     SPB   FIN
     U     M     10.7
     FP    "Detecta_start"
     =     "pubStart"

      CALL  FB   304 , "DISCONNEC"
      STA    :=TRUE
      FIN    :="DISCONNECT_DONE"
      FAILED :="PUB_FAI"
      RR_CODE:="PUB_RR_CODE"
      BEA

      FIN:  BEA
```

FB300:

```
// Reset the FIN flag
   SET
   R     #FIN
//-----------------------------------------------
-----------Connection TCP/IP---------------

     U     #ABORT
     R     #FAILED

     CALL  "TDISCON" , "DB66"
// To quit the connection
      REQ   :=#ABORT
      ID    :=#ENLACE.id
      DONE  :=#DIS_DONE
      BUSY  :=#DIS_BUSY
      ERROR :=#DIS_ERROR
      STATUS:=#DIS_STATUS

     U     #FAILED
     SPB   FINF

//-----------------------------------------------
//--constitute connect packet------
     U     "Cons_con_done"
     SPB   STAT

     CALL  FB     1 , DB111
      Ver         :=#Ver
      KeepAlive   :=#KeepAlive
      ConnectFlags:=#ConnectFlags
      ClientID    :=#ClientID
      WillTopic   :=#WillTopic
      WillMessage :=#WillMessage
      UserName    :=#UserName
      Password    :=#Password
      DONE        :="Cons_con_done"

     U     "Cons_con_done"
     SPB   STAT
     BEA

    STAT: U     "Estado_CON"
  // If it finishes the connection, it jump to
    //Tsend
     SPB   SEND
     CALL  "TCON" , DB65
      REQ
:=#COMMUNICATION_START
      ID     :=#ENLACE.id
      DONE   :=#CON_DONE
      BUSY   :=#CON_BUSY
      ERROR  :=#CON_ERROR
      STATUS :=#CON_STATUS
      CONNECT:=#ENLACE
  // Information of enlace


     U     #CON_DONE
     FP    "Detecta_CON_DONE
     O     "Estado_CON"
     =     "Estado_CON"
    // State of connection

     U     "Estado_CON"
          // CUANDO HA TERMINADO
    LA    CONEXIÓN,Saltar a Tsend
    SPB   SEND
    U     #INIT_COM
    UN    #CON_BUSY
  // If PLC has finished the connection of
TCP, it will not active the TCOM
    =     #COMMUNICATION_START
    U     #CON_ERROR
    O     #CON_BUSY
    SPB   EFN1

    U     #CON_ERROR
   // SI TIENE ERROR, GUARDAMOS
ERROR PARA ANALIZAR.
    L     #CON_STATUS
    T     #CON_STATUS_SAVE
    BEA

EFN1: L     #CON_STATUS_SAVE
// If the connection has failed, it will out
put the error code.
    T     #ER_CODE
    L     1
    T     #FA_CODE
    SET
    S     #FAILED
    BEA

SEND: U     "ESTADO_TSEND"
    SPB   RCV

    CALL  "TSEND" , DB63
     REQ   :=#SEND_REQ
     ID    :=#ENLACE.id
     LEN   :=MW52
     DONE  :=#SEND_DONE
     BUSY  :=#SEND_BUSY
     ERROR :=#SEND_ERROR
     STATUS:=#SEND_STATUS
     DATA  :=P#DB23.DBX0.0 BYTE
196

    U     #SEND_DONE
    FP    "DETECTA_TSEND"
    O     "ESTADO_TSEND"
    =     "ESTADO_TSEND"
    U     "ESTADO_TSEND"
    SPB   RCV
    U     "Ciclo(0.1s)"              // CADA
CIETO TIEMPO PEDIR UNA VEZ DE
ENVIACIÓN
    FP    "Detecta_cilco(0.1s)"
    UN    #SEND_BUSY
    =     #SEND_REQ
    U     #SEND_REQ
    ZV    "COUN_SEN_CON"
    L     "COUN_SEN_CON"
    L     5
    >I
    SPB   EFN2
    U     #SEND_ERROR                // SI
HAY ERROR O HA TERMINADO, NO
VAMOS A ENVIAR
    O     #SEND_DONE
    R     #SEND_REQ
     U     #SEND_ERROR               // SI
TIENE ERROR, GUARDAMOS ERROR
PARA ANALIZAR.
    L     #SEND_STATUS
    T     #SEND_STATUS_SAVE
    BEA

EFN2: L     #SEND_STATUS_SAVE
// If the connection has failed, it will out
put the error code.
    T     #ER_CODE
    L     2
    T     #FA_CODE
    SET
    S     #FAILED
    SET
    R     "COUN_SEN_CON"
    SET
    R     "Estado_CON"
    BEA

//-------------------------------------------------
---------------------------------------------
RCV:  U     "ESTADO_RCV"               //
T no va a reiniciar el RLO
    SPB   CHEK
    CALL  "TRCV" , "DB64"
     EN_R    :=#RCV_START
     ID      :=#ENLACE.id
     LEN     :=4
     NDR     :=#RCV_NDR
     BUSY    :=#RCV_BUSY
     ERROR   :=#RCV_ERROR
     STATUS  :=#RCV_STATUS
     RCVD_LEN:=#RCV_RLEN
     DATA    :="CONACK".CONACK

    U     #RCV_NDR                   // Hay
que poner adelande
    FP    "DETECTA_RCV"
    O     "ESTADO_RCV"
    =    "ESTADO_RCV"

    U    "ESTADO_RCV"              // T
no va a reiniciar el RLO
    SPB   CHEK
    U    "Ciclo(0.1s)"
    FP    "Detectaciclo(0.1)_rec"
    UN    #RCV_BUSY
    =    #RCV_START
    U    #RCV_START
    ZV    "COUN_REV_CON"
    L    "COUN_REV_CON"
    L    5
    >I
    SPB   EFN3
    L    #RCV_RLEN
    T    #RCV_RLEN_SAVE
    L    #RCV_STATUS
    T    #RCV_STATUS_SAVE
    BEA

EFN3: L     #RCV_STATUS_SAVE
// If the connection has failed, it will out
put the error code.
    T    #ER_CODE
    L    3
    T    #FA_CODE
    SET
    S    #FAILED
    SET
    R    "COUN_REV_CON"
    SET
    R    "Estado_CON"
    SET
    R    "ESTADO_TSEND"
    BEA
// Check out the connection
CHEK: L
"CONACK".CONACK.ConnectReturncod
e
    L    0
    ==I
    SPB   FIN
    L
"CONACK".CONACK.ConnectReturncod
e
    T    #ER_CODE
    L    4
    T    #FA_CODE
    SET
    S    #FAILED
    SET
    R    "Estado_CON"
    SET
    R    "ESTADO_TSEND"
    SET
    R    "ESTADO_RCV"
    BEA
FIN:  SET
    R    "Cons_con_done"
    SET
    R    "Estado_CON"
    SET
    R    "ESTADO_TSEND"
    SET
    R    "ESTADO_RCV"
    SET
    S    #FIN
    SET
    R    "COUN_SEN_CON"
    SET
    R    "COUN_REV_CON"
    L    0
    T    #ER_CODE
    L    0
    T    #FA_CODE
    SET
    R    #FAILED
    BEA
FINF: BEA
```

FB301:

```
  SET
    R    #Done
    AUF   "Publish0Version2"
// Clean the DB
    U    "pubCleanFlag"
    SPB   STAR
    L    0
    T    #PointDB
    L    197
LID:  T    #Conuter
    L    0
    T    DBB [#PointDB]
    L    8
    L    #PointDB
    +D
    T    #PointDB
    L    #Conuter
    LOOP  LID
    SET
    S    "pubCleanFlag"
    U    "pubCleanFlag"
    SPB   STAR
    BEA
```

```
STAR: L     #HeaderFlags
      T     DBB    0



//--------------------------------------------------
// Len
      L     2
      T     DBB    1
//--------------------------------------------------
// Topic
      L     P##Topic
      LAR1
      L     B [AR1,P#1.0]
      T     DBW    2                  // Topic
length
      L     DBW    2
      L     DBB    1
      +I
      T     DBB    1                  // Calculate
Len
      L     32
      T     #PointDB                   // Start fill
DB en DBB4
      L     B [AR1,P#1.0]
L1:   T     #Conuter
      L     B [AR1,P#2.0]
      T     DBB [#PointDB]
      L     8
      +AR1
      L     8
      L     #PointDB
      +D
      T     #PointDB
      L     #Conuter
      LOOP  L1
//--------------------------------------------
//Identifier

      L     #QoS
      L     0
      ==I
      SPB   IDE

      L     2
      L     DBB    1
      +I
      T     DBB    1                  // Calculate
Len
      L     #Identifier
      T     DBW [#PointDB]
      L     16
      L     #PointDB
      +D
      T     #PointDB
//--------------------------------------------
//Message

IDE:  L     P##Message
      LAR1
      L     B [AR1,P#1.0]
      L     DBB    1
      +I
      T     DBB    1
 // Calculate Len
      L     B [AR1,P#1.0]
L2:   T     #Conuter
      L     B [AR1,P#2.0]
      T     DBB [#PointDB]
      L     8
      +AR1
      L     8
      L     #PointDB
      +D
      T     #PointDB
      L     #Conuter
      LOOP  L2
      SET
      S     #Done
      BEA
```

FB302:

```
      SET
      R     #FIN
//-------------------------------------------------
// Constitute publish0Packet
      U     "Cons_pub_done"
      SPB   TSE
      CALL  FB     3 , DB26
       QoS :=1
       Done:="Cons_pub_done"
      U     "Cons_pub_done"
      SPB   TSE
      BEA
TSE:  U     #Estado_TS_MQTT
      SPB   RCV
 // Si ha realizado el envio, va a saltar a
SOQ1.
      L     "Publish0Version2".Len
      T     #LengthT
      L     2
      L     #LengthT
      +I
      T     #LengthT
      CALL  "TSEND" , DB63
       REQ   :=#SEND_REQ
      ID    :=#ENLACE.id
      LEN   :=#LengthT
      DONE  :=#SEND_DONE
      BUSY  :=#SEND_BUSY
      ERROR :=#SEND_ERROR
      STATUS:=#SEND_STATUS
      DATA  :=P#DB25.DBX0.0 BYTE
196
// CADA CIETO TIEMPO PEDIR UNA
VEZ DE ENVIACIÓN
     U    #STA
     FP    #Detecta_ciclo
     UN    #SEND_BUSY
     =    #SEND_REQ
     U    #SEND_DONE
     FP    #Dectar_TS_MQTT
     O    #Estado_TS_MQTT
     =    #Estado_TS_MQTT
     U    #Estado_TS_MQTT
     SPB   RCV
// SI TIENE ERROR, GUARDAMOS
ERROR PARA ANALIZAR.
     L    #SEND_STATUS
     T    #SEND_STATUS_SAVE
     U    #SEND_ERROR
     SPB   FNR1
     BEA
FNR1: SET
     S    #FAILED
     L    #SEND_STATUS_SAVE
     T    #RR_CODE
     L    1
     T    #FA_CODE
     BEA
RCV:  U     "Ciclo(0.1s)"
     FP    #Detectaciclo
     UN    #RCV_BUSY
     UN    #ESTADO_RCV
     =    #RCV_START
     U    #RCV_START
     ZV    Z     1
     L    Z     1
     L    5
     >I
     SPB   FNR2
     CALL  "TRCV" , "DB64"
      EN_R    :=#RCV_START
      ID      :=#ENLACE.id
      LEN     :=4
      NDR     :=#RCV_NDR
      BUSY    :=#RCV_BUSY
      ERROR   :=#RCV_ERROR
      STATUS  :=#RCV_STATUS
      RCVD_LEN:=#RCV_RLEN
      DATA    :="PUBACK".PUBACK

     U    #RCV_NDR
   // Hay que poner adelande
     FP    #DETECTA_RCV
     O    #ESTADO_RCV
     =    #ESTADO_RCV
     U    #ESTADO_RCV
// T no va a reiniciar el RLO
     SPB   CHEK
     L    #RCV_RLEN
     T    #RCV_RLEN_SAVE
     L    #RCV_STATUS
     T    #RCV_STATUS_SAVE
     BEA
FNR2: L     #RCV_STATUS_SAVE
// If the connection has failed, it will out
put the error code.
     T    #RR_CODE
     L    2
     T    #FA_CODE
     SET
     S    #FAILED
     SET
     R    Z     1
     SET
     R    #Estado_TS_MQTT
     BEA
CHEK: L
"Publish_1".Publish_1.MessageIdentifier
     L
"PUBACK".PUBACK.MessgeIdentifier
     ==I
     SPB   FIN
     L
"PUBACK".PUBACK.MessgeIdentifier
     T    #RR_CODE
     L    3
     T    #FA_CODE
     SET
     S    #FAILED
     SET
     R    #Estado_TS_MQTT
     SET
     R    #ESTADO_RCV
     R    Z     1
     L    0                          // If the
connection has failed, it will out put the
error code.
     BEA
FIN:  SET
     R    #Estado_TS_MQTT
     SET
     R    #ESTADO_RCV
     SET
     S    #FIN
     SET
     R    Z     1
     L    0
// If the connection has failed, it will out
put the error code.
     T    #RR_CODE
     L    0
     T    #FA_CODE
     SET
     R    #FAILED
     BEA
```

FB303:

```
    SET
     R    #FIN
//-------------------------------------------------
// Constitute publish0Packet
     U    "Cons_pub_done"
     SPB  TSE

     CALL  FB     3 , DB27
      QoS :=2
      Done:="Cons_pub_done"
     U    "Cons_pub_done"
     SPB  TSE
     BEA

//-------------------------------------------------
--------------------------------------------
TSE:  U    #Estado_TS_MQTT
     SPB  REC
  // Si ha realizado el envio, va a saltar a
SOQ1.
     L    "Publish0Version2".Len
     T    #LengthT
     L    2
     L    #LengthT
     +I
     T    #LengthT

     CALL  "TSEND" , DB63
//   #SEND_REQ
      REQ   :=#SEND_REQ
      ID    :=#ENLACE.id
      LEN   :=#LengthT
      DONE  :=#SEND_DONE
      BUSY  :=#SEND_BUSY
      ERROR :=#SEND_ERROR
      STATUS:=#SEND_STATUS
      DATA  :=P#DB25.DBX0.0 BYTE
196
     U    #STA
     FP   #Detecta_ciclo
     UN   #SEND_BUSY
     =    #SEND_REQ
     U    #SEND_DONE
     FP   #Dectar_TS_MQTT
     O    #Estado_TS_MQTT
     =    #Estado_TS_MQTT
     U    #Estado_TS_MQTT
     SPB  REC
// SI TIENE ERROR, GUARDAMOS
ERROR PARA ANALIZAR.
     L    #SEND_STATUS
     T    #SEND_STATUS_SAVE
     U    #SEND_ERROR
     SPB  FNR1
     BEA
FNR1: SET
     S    #FAILED
     L    #SEND_STATUS_SAVE
     T    #RR_CODE
     L    1
     T    #FA_CODE
     BEA
//-------------------------------------------------
REC:  U    #ESTADO_RCV
     SPB  CEK1
     U    "Ciclo(0.1s)"
     FP   #RCV_DETECTA_CICLO
     UN   #RCV_BUSY
     UN   #ESTADO_RCV
     =    #RCV_START
     U    #RCV_START
 // It will try 5 times
     ZV   Z     4
     L    Z     4
     L    5
     >I
     SPB  FNR2
     CALL  "TRCV" , "DB64"
      EN_R    :=#RCV_START
      ID      :=#ENLACE.id
      LEN     :=4
      NDR     :=#RCV_NDR
      BUSY    :=#RCV_BUSY
      ERROR   :=#RCV_ERROR
      STATUS  :=#RCV_STATUS
      RCVD_LEN:=#RCV_RLEN
      DATA    :="PUBREC".PUBREC
     U    #RCV_NDR
  // Hay que poner adelande
```

```
     FP    #DETECTA_RCV
     O     #ESTADO_RCV
     =     #ESTADO_RCV
     U     #ESTADO_RCV
// T no va a reiniciar el RLO
     SPB   CEK1
     L     #RCV_RLEN
     T     #RCV_RLEN_SAVE
     L     #RCV_STATUS
     T     #RCV_STATUS_SAVE
     BEA
FNR2: L    #RCV_STATUS_SAVE
// If the connection has failed, it will out
put the error code.
     T     #RR_CODE
     L     2
     T     #FA_CODE
     SET
     S     #FAILED
     SET
     R     Z     4
     SET
     R     #Estado_TS_MQTT
     BEA
//--------------------------------------------------
CEK1: L
"Publish_2".Publish_2.MessageIdentifier
     L
"PUBREC".PUBREC.MessgeIdentifier
     ==I
     SPB   REL
// If is not equal
     L
"PUBREC".PUBREC.MessgeIdentifier
// If the connection has failed, it will out
put the error code.
     T     #RR_CODE
     L     3
     T     #FA_CODE
     SET
     S     #FAILED
     SET
     R     Z     4
     SET
     R     #Estado_TS_MQTT
     SET
     R     #ESTADO_RCV
     BEA
//--------------------------------------------------
REL:  U    #Estado_TS_MQTT_1
     SPB   COP
 // Si ha realizado el envio, va a saltar a
SOQ1.
     CALL  "TSEND" , DB63
   //   #SEND_REQ
      REQ   :=#SEND_REQ_1
      ID    :=#ENLACE.id
      LEN   :=4
      DONE  :=#SEND_DONE_1
      BUSY  :=#SEND_BUSY_1
      ERROR :=#SEND_ERROR_1
      STATUS:=#SEND_STATUS_1
      DATA  :="PUBRELEA".Release


     U     "Ciclo(0.1s)"              // CADA
CIETO TIEMPO PEDIR UNA VEZ DE
ENVIACIÓN
     FP    #Detecta_ciclo_1
     UN    #SEND_BUSY_1
     =     #SEND_REQ_1
     U     #SEND_REQ_1
     ZV    Z     5
     L     Z     5
     L     5
     >I
     SPB   FNR4
     U     #SEND_DONE_1
     FP    #Dectar_TS_MQTT_1
     O     #Estado_TS_MQTT_1
     =     #Estado_TS_MQTT_1
     U     #Estado_TS_MQTT_1
     SPB   COP
// SI TIENE ERROR, GUARDAMOS
ERROR PARA ANALIZAR.
     L     #SEND_STATUS_1
     T     #SEND_STATUS_SAVE_1
     BEA
FNR4: L    #SEND_STATUS_SAVE_1
// If the connection has failed, it will out
put the error code.
     T     #RR_CODE
     L     4
     T     #FA_CODE
     SET
     S     #FAILED
     SET
     R     Z     4
     SET
     R     Z     5
     SET
     R     #Estado_TS_MQTT
     SET
     R     #ESTADO_RCV
     BEA
COP:  U    #ESTADO_RCV_1
```

```
    SPB   CEK2
    U     "Ciclo(0.1s)"
    FP    #RCV_DETECTA_CICLO_1
    UN    #RCV_BUSY_1
    UN    #ESTADO_RCV_1
    =     #RCV_START_1

    U     #RCV_START_1               // It
will try 5 times
    ZV    Z      6
    L     Z      6
    L     5
    >I
    SPB   FNR5
    CALL  "TRCV" , "DB64"
     EN_R    :=#RCV_START_1
     ID      :=#ENLACE.id
     LEN     :=4
     NDR     :=#RCV_NDR_1
     BUSY    :=#RCV_BUSY_1
     ERROR   :=#RCV_ERROR_1
     STATUS  :=#RCV_STATUS_1
     RCVD_LEN:=#RCV_RLEN_1
     DATA    :=DB11.PUBCOM
    U     #RCV_NDR_1
  // Hay que poner adelande
    FP    #DETECTA_RCV_1
    O     #ESTADO_RCV_1
    =     #ESTADO_RCV_1
    U     #ESTADO_RCV_1
 // T no va a reiniciar el RLO
    SPB   CEK2
    L     #RCV_RLEN_1
    T     #RCV_RLEN_SAVE_1
    L     #RCV_STATUS_1
    T     #RCV_STATUS_SAVE_1
    BEA

FNR5: L     #RCV_STATUS_SAVE_1
// If the connection has failed, it will out
put the error code.
    T     #RR_CODE
    L     5
    T     #FA_CODE
    SET
    S     #FAILED
    SET
    R     Z      4
    SET
    R     Z      5
    SET
    R     Z      6
    SET
    R     #Estado_TS_MQTT
    SET
    R     #ESTADO_RCV
    SET
    R     #Estado_TS_MQTT_1
    BEA
//-------------------------------------------------
CEK2: L
"PUBRELEA".Release.MessgeIdentifier
    L     DB11.DBW    2
    ==I
    SPB   FIN
    L     DB11.DBW    2
 // If the connection has failed, it will out
put the error code.
    T     #RR_CODE
    L     6
    T     #FA_CODE
    SET
    S     #FAILED
    SET
    R     Z      4
    SET
    R     Z      5
    SET
    R     Z      6

    SET
    R     #Estado_TS_MQTT
    SET
    R     #ESTADO_RCV
    SET
    R     #Estado_TS_MQTT_1
    BEA
//-------------------------------------------------
FIN:  SET
    R     #Estado_TS_MQTT
    SET
    R     #ESTADO_RCV
    SET
    R     #Estado_TS_MQTT_1
    SET
    R     #ESTADO_RCV_1
    SET
    S     #FIN
    SET
    R     #FAILED
    SET
    L     0
    T     #FA_CODE
    L     0
    T     #RR_CODE
    SET
    R    Z     4
    SET
    R    Z     5
    SET
    R    Z     6
    BEA
```

FB304:

```
   SET
    R    #FIN


    CLR
    U    #Estado_TS_MQTT
    SPB   FIN
// Si ha realizado el envio, va a saltar a
SOQ1.
    CALL  "TSEND" , DB63
     REQ   :=#SEND_REQ
     ID    :=#ENLACE.id
     LEN   :=2
     DONE  :=#SEND_DONE
     BUSY  :=#SEND_BUSY
     ERROR :=#SEND_ERROR
     STATUS:=#SEND_STATUS
     DATA  :=DB12.DISCONNEC

    U    #STA
    FP    #Detecta_ciclo
    UN    #SEND_BUSY
    =    #SEND_REQ

    U    #SEND_DONE
    FP    #Dectar_TS_MQTT
    O    #Estado_TS_MQTT
    =    #Estado_TS_MQTT
    U    #Estado_TS_MQTT
    SPB   FIN
// SI TIENE ERROR, GUARDAMOS
ERROR PARA ANALIZAR.
    L    #SEND_STATUS
    T    #SEND_STATUS_SAVE
    U    #SEND_ERROR
    SPB   FNR
    BEA
FNR:  SET
    S    #FAILED
    L    #SEND_STATUS_SAVE
    T    #RR_CODE
    BEA
FIN:  SET
    R    #Estado_TS_MQTT
    SET
    S    #FIN
    SET
    R    #FAILED
    L    0
    T    #RR_CODE
    BEA
```

FB1:

```
   SET
    R    #DONE
//----------------------------------------------
    L    #Ver
    T    "connectVersion2".Vers

    L    #ConnectFlags
    T    "connectVersion2".ConnextFlags
    L    #KeepAlive
    T    "connectVersion2".KeepAlive
//-----------------------------------------------
//Iniciar el DB
    U    "conCleanFlag"
    SPB   STAR
    AUF   "connectVersion2"
    L    P#12.0
    T    #PointDire
    L    185
// Length of string
LID:  T    #Ini_counter
    L    0
    T    DBB [#PointDire]
    L    8
    L    #PointDire
    +D
    T    #PointDire
    L    #Ini_counter
    LOOP  LID
    SET
    S    "conCleanFlag"

//CONSTITUTE ClientID part
//----------------------------------------------
STAR: CALL  "InsertPayload" , DB22
      Payload:=#ClientID
      LastDir:=12
      newDir :=
//----------------------------------------------
    L    P##ConnectFlags
    LAR1

// CONSTITUTE other payload
//----------------------------------------------
    U     [AR1,P#0.2]
    SPB   WILL
```

```
TUSE: U     [AR1,P#0.7]
     SPB   USER
TPAS: U     [AR1,P#0.6]
     SPB   PASS
MAIN: SET
     S     #DONE
     BEA
WILL: CLR
     CALL  "InsertPayload" , DB22
      Payload:=#WillTopic
      LastDir:=MW52
      newDir :=
     CALL  "InsertPayload" , DB22
      Payload:=#WillMessage
      LastDir:=MW52
      newDir :=
     PA   TUSE
     USER: CLR
     CALL  "InsertPayload" , DB22
      Payload:=#UserName
      LastDir:=MW52
      newDir :=
     SPA   TPAS
     PASS: CLR
     CALL  "InsertPayload" , DB22
      Payload:=#Password
      LastDir:=MW52
      newDir :=
     SPA   MAIN
```

FB2:

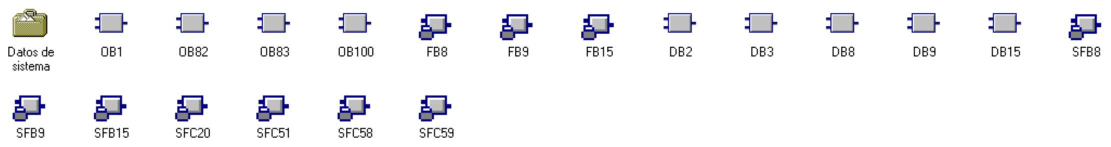
```
   AUF   "connectVersion2"
     L     #LastDir
     L     8
     *D
     T     #PointerDir
     L     8
     L     #PointerDir
     +D
 // Calculate the last bit
     T     #PointerDir
     L     P##Payload
     LAR1
     L     P#1.0
     +AR1
     L     B [AR1,P#0.0]
 // Cache the length of Payload
     T     DBB [#PointerDir]

     L     B [AR1,P#0.0]
// Cache the length of Payload
     L     #LastDir
     +I
     T     MW    52
     L     MW    52
     L     2
     +I
     T     MW    52
     L     MW    52
     T     DBB    1
     L     DBB    1
     L     2
     -I
     T     DBB    1
     L     B [AR1,P#0.0]
 // Length of string
LID:  T     "CounterByte"
     L     8
     +AR1
     L     8
     L     #PointerDir
     +I
     T     #PointerDir
     L     B [AR1,P#0.0]
     T     DBB [#PointerDir]
     L     "CounterByte"
     L     1
     -I
     L     "CounterByte"
     LOOP  LID
     BEA
```

FB3:

```
     SET
     R     #Done
     AUF   "Publish0Version2"
//-------------------------------------------------
// Clean the DB
     U     "pubCleanFlag"
     SPB   STAR
     L     0
     T     #PointDB
     L     197
LID:  T     #Conuter
     L     0
     T     DBB [#PointDB]
     L     8
     L     #PointDB
     +D
     T     #PointDB
     L     #Conuter
     LOOP  LID
     SET
     S     "pubCleanFlag"
     U     "pubCleanFlag"
     SPB   STAR
     BEA
```

```
STAR: L     #HeaderFlags
      T     DBB    0
//--------------------------------------------------
// Len
      L     2
      T     DBB    1
//--------------------------------------------------
// Topic
      L     P##Topic
      LAR1
      L     B [AR1,P#1.0]
      T     DBW    2
 // Topic length
      L     DBW    2
      L     DBB    1
      +I
      T     DBB    1
// Calculate Len
      L     32
      T     #PointDB
// Start fill DB en DBB4
      L     B [AR1,P#1.0]
L1:   T     #Conuter
      L     B [AR1,P#2.0]
      T     DBB [#PointDB]
      L     8
      +AR1
      L     8
      L     #PointDB
      +D
      T     #PointDB
      L     #Conuter
      LOOP  L1
//--------------------------------------------
//Identifier
      L     #QoS
      L     0
      ==I
      SPB   IDE
      L     2
      L     DBB    1
      +I
      T     DBB    1
 // Calculate Len
      L     #Identifier
      T     DBW [#PointDB]
      L     16
      L     #PointDB
      +D
      T     #PointDB
//--------------------------------------------
//Message
IDE:  L     P##Message
      LAR1
      L     B [AR1,P#1.0]
      L     DBB    1
      +I
      T     DBB    1
 // Calculate Len
      L     B [AR1,P#1.0]
L2:   T     #Conuter
      L     B [AR1,P#2.0]
      T     DBB [#PointDB]
      L     8
      +AR1
      L     8
      L     #PointDB
      +D
      T     #PointDB
      L     #Conuter
      LOOP  L2
      SET
      S     #Done
      BEA
```

8.2 Program in controller 2



OB1:

```
// Comunicacion entre IM151 y
controlador_2
      L     MB   200
      T     AB     0
      U     E      0.0
      =     A    136.0
// Comunicacion entre controlador 1 y
controlador 2
      L     MB   200
      T     DB2.DBB    0
  // Leer entrada de MB200
      UN    M      2.0
      L     S5T#10MS
      SE    T      0
      U     T      0
      =     M      2.0
 // PULSO para activa USEND
      CALL  "USEND" , DB8
  // Enviar DB2 a CP
       REQ   :=M2.0
       ID    :=W#16#2       R_ID
:=DW#16#2
       DONE  :=M0.0
       ERROR :=M0.1
       STATUS:=MW3
       SD_1  :=P#DB2.DBX0.0 BYTE 2

      CALL  "URCV" , DB9
  // Recibir datos a DB3
       EN_R  :=TRUE
       ID    :=W#16#1
       R_ID  :=DW#16#1
       NDR   :=M0.2
       ERROR :=M0.3
       STATUS:=MW5
       RD_1  :=P#DB3.DBX0.0 BYTE 2

      L     DB3.DBB    0
// Mostrar los datos a salida
      T     AB   137
      BEA
```