



Democratic and Popular Republic of Algeria  
Ministry of Higher Education and Scientific Research  
University of Mohamed Khider – BISKRA  
Faculty of Exact Science, Natural Sciences and Life  
**Computer Science Department**

OrderNumber : GLSD2/M2/2019

## Report

Presented to obtain the diploma of academic Master in

# Computer Science

Option: **Software Engineering and Distributed Systems**

---

# Formal verification of the implementation of the MQTT protocol in IoT devices

---

By:  
**ABID AOUATEF**

Defended the 7 juillet 2019, in front of the jury:

OuaarHanane	MCB	President
Hmidi Zohra	MAA	Supervisor
SahliSihem	MAA	Examiner

# Signings

**With the help of God Almighty, we have  
now reached the end of the course of  
study that we have been waiting for  
throughout the academic years,**

**To our dear teachers ... for their continued  
struggle and their great sacrifice**

**To my dear parents, for all their  
sacrifices, their love, their tenderness,  
their support and their prayers  
throughout my studies,**

**To my dear brothers and my sisters for  
their constant encouragement, and their  
moral support,**

**To all my family for their support  
throughout my university career,**

**May this work be the accomplishment of  
your vows, so much so, and escape from  
your infallible support,**

**Thank you for always being here for me**

## **Abstract**

Internet of Things (IoT) is a technology used to provide smarter services to users by connecting various devices to the Internet and allowing these devices to exchange information with each other. This connections run out depending on certain protocols, including Message Queue Telemetry Transport (MQTT).

MQTT is a protocol suitable for application in Internet of Things (IoT) devices; it is designed around requirements for low bandwidth and small code footprint. The count of the embedded devices that make use of it is constantly increasing. Therefore, a mistake in its implementation would be critical from both operational and security perspective. For that, we need to implement a program with new test language for verification of implementations of the MQTT protocol in IoT devices such as TTCN-3. the main objective of our work is to provide a formal verification of implementations of the MQTT protocol in IoT devices by using a modular language TTCN-3.

**Keywords:** Internet of Things (IoT), Message Queue Telemetry Transport (MQTT), formal verification, modular language TTCN-3.

## **A**

**AUTOSAR:** Automotive Open System Architecture

## **D**

**DODAG:** Destination Oriented Directed Acyclic Graph

**DMR:** Designated Marksman Rifle

**DNS:** Domain Name System

**DTLS:** Data Transport Layer Security

## **E**

***E-CARP:** Enhanced CARP*

***ETSI-MTS:** ETSI Methods for Testing and Specification*

## **F**

**FTP:** file transfer protocol

## **H**

**HTML:** Hyper TextMarkup Language **HTTP:** Hyper Text Transfer Protocol

## **I**

**IBM:** International Business Machines

**IP:** Internet Protocol

**IPv:** Internet Protocol version

**ISDN:** Integrated Service Digital Network

## **L**

**LoWPAN:** low-power Wireless Personal Area Network

**L2CAP:** Logical link Control and Adaptation Protocol

## **M**

**MAC:** Medium Access Control

**MTV:** Maximum Transmission Unit

## **Q**

**QoS:** Quality of Service

## **R**

**RFID:** Radio-Frequency Identification

## **S**

**SDN:** Software-defined Network

**SDLC:** Software Development Life Cycle

**SIP:** Systematic Investment Plan

**SMS:** Short Message Service

**SRS:** Software Requirement Specification

**STF:** Smartphone Test Farm

**T**

**TCP:** Transmission Control Protocol

**U**

**URL:** Uniform Resource Locator

**UDP:**User Datagram Protocol

**W**

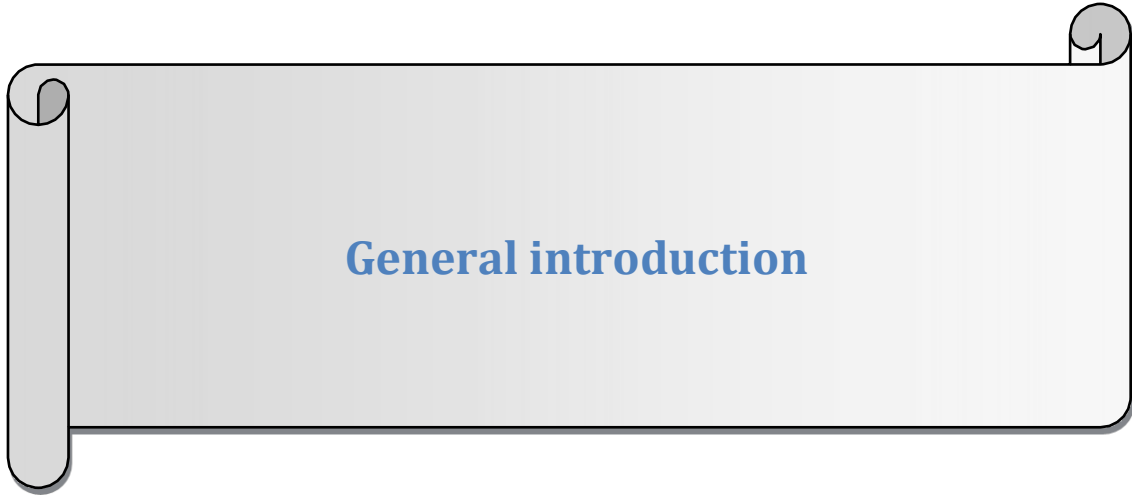
**WiFi:** *wireless fidelity*

**X**

**XML:** ExtensibleMarkup Language

Figure 1.1: Internet of things in four environments.....	4
Figure 1.2 : MQTT Architecture « Publish/ Subscribe ».....	11
Figure 1.3 : MQTT Packet Structure.....	12
Figure 2.1 : An Overview of Testing Methods.....	23
Figure 2.2 : Black Box Testing.....	24
Figure 2.3 : White Box Testing.....	24
Figure 2.4: Types of Performance Testing.....	26
Figure 2.5: The TTCN-3 Module Structure.....	29
Figure 2.6: Overview of TTCN-3 Types.....	30
Figure 3.1 : Simplified model of the approach.....	46
Figure 3.2: Declared the types.....	47
Figure 3.3: Defined the imports of the message structure.....	48
Figure 3.4: An Example of CONNECT Message with TTCN-3.....	48
Figure 3.5: An Example of PUBLISH Message with TTCN-3.....	49
Figure 3.6: An Example of SUBSCRIBE Message with TTCN-3.....	49
Figure 3.7: TTCN union type of MQTT message.....	50
Figure 3.8: Describe figure of component type and port type in our system.....	51
Figure 3.9: Function to verify the standards of CONNECT mqtt message.....	51
Figure 3.10: Test Function using alt statement of the message.....	53
Figure 3.11: Test case of our system.....	54

Table 1.1: OSI Model Layers.....	8
Table 1.2: TCP\IP Model Layers.....	8
Table 1.3: Application Layer Protocols.....	9
Table 3.1: Structure of an MQTT Control Packet.....	34
Table 3.2: Fixed Header Format.....	35
Table 3.3: MQTT Control Packet Type.....	35
Table 3.4: Flag Bits.....	36
Table 3.5: The QoS Levels.....	37
Table 3.6: Remaining Length values.....	38
Table 3.7: Packet Identifier.....	39
Table 3.8: MQTT Control Packet Type that contains Packet Identifier.....	39
Table 3.9: Payload for each Message Type.....	40
Table 3.10: Fixed Header of the CONNECT Packet.....	41
Table 3.11: Variable Header of the CONNECT Packet.....	41
Table 3.12: Fixed Header of the PUBLISH Packet.....	43
Table 3.13: Expected PUBLISH Packet Reponses.....	43
Table 3.14: Fixed Header of the SUBSCRIBE Packet.....	44
Table 3.15: Fixed Header of the DISCONNECT Packet.....	45



## **General introduction**



## General introduction

The need for connected devices has grown rapidly in the last few years. Starting with personal computers using the internet, continuing with smartphones and new wireless technologies and reaching a point where even small devices, not necessarily controlled by a human, should be able to communicate. They will be able to exchange any kind of data to enable new opportunities that we barely start to foresee. The growth of this new domain of connected devices talking to one another has been called the Internet of Things (IoT).

The Internet of Things (IoT) is a new paradigm with the aim of creating connectivity for (everything) that can carry a minimum of storage and computational power, such that these connected things can collaborate anytime, anywhere and in any form, within applications in various domains such as personal and social, transportation, enterprise businesses and service and utility monitoring . Some recent estimates suggest that the number of IoT devices exceeds 30 billion with more than 200 billion intermittent connections generating over 700B Euros in revenue by 2020. [1]

Various protocols are used for communication in an IoT system. TCP/IP is a popular protocol used in lower layers. Several protocols are adapted for the application layer in an IoT system, among them is Message Queue Telemetry Transport Protocol (MQTT).

This kind of implementations of networking protocol stacks is in need of thorough testing in order to ensure not only its security, but also its interoperability and compliance to relevant standards and specifications. Implementing test suites for verification of implementations of TCP/IP communications protocols in the Internet Engineering Task Force (IETF) world has traditionally been done in any number of ways, by using general-purpose programming languages such as C, C++, Java, Python and others [2]. In the International Telecommunication Union (ITU)/ETSI world. [3], TTCN-3 has been developed as a domain-specific language for the specific use case of writing protocol conformance tests.

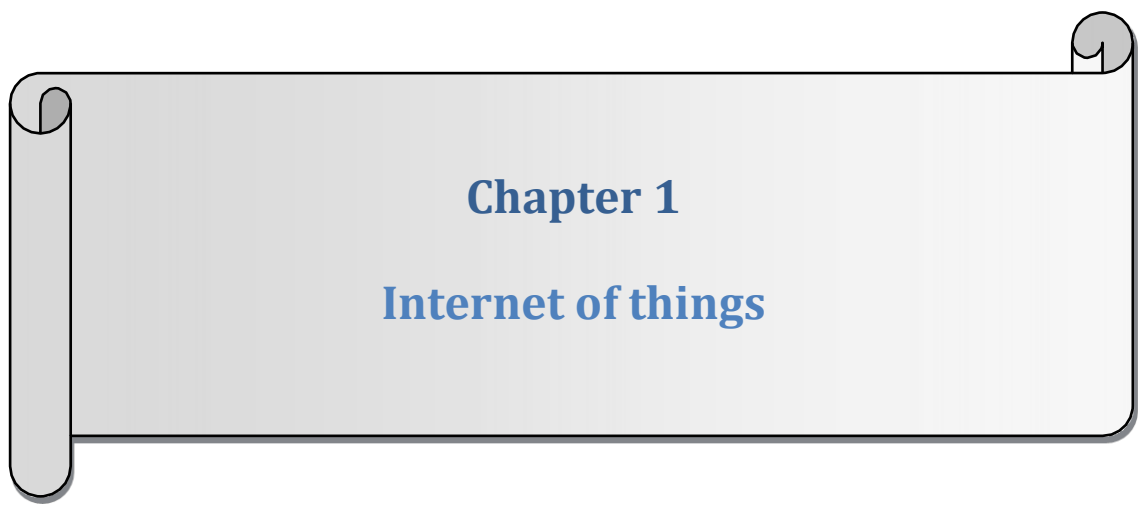
The objective of our project is to have answers for the questions below:

- How can we assure a proper protocol implementation?

- Could we make sure that it is correct in a more formal way?
- Can the MQTT protocol implementation in IoT devices be verified formally?

And to reach this goal we will organize our project in three chapters:

- The first chapter presents an overview of IoT such as characteristics of the internet of things, the IoT standards and protocols, IoT challenges and solutions.
- The second chapter is devoted to software testing such as definition, principal, objectives, levels, types of software testing and conformance testing. This chapter also details all about TTCN-3.
- Finally in the third chapter we will detail the conception and implementation of our system, the development environment and the tools used. To illustrate our implementation, we will explain the different experiences of the realized system and we will discuss the results obtained.



# **Chapter 1**

## **Internet of things**

## **1. Introduction**

The rapid growth in technology and internet connected devices has enabled Internet of Things (IoT) to be one of the important fields in computing. Standards, technologies and platforms targeting IoT ecosystem are being developed at a very fast pace. IoT enables things to communicate and coordinate decisions for many different types of applications including healthcare, home automation, disaster recovery, and industry automation.

In this chapter, we will discuss all aspects of the Internet of things, its characteristics, its applications and its standards, all of that after we know what is the definition of the Internet of things. We will also clarify some of the IoT's protocols and challenges to finish this chapter by the solutions of IoT.

## **2. Internet of things (IoT)**

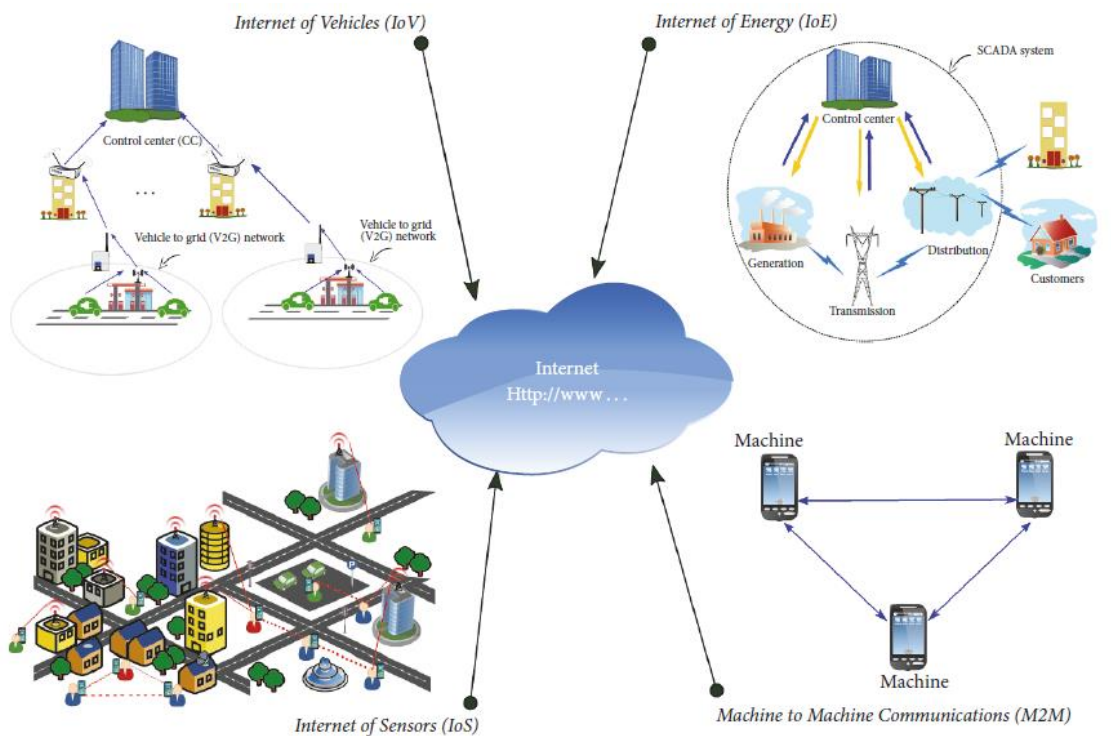
### **2.1. What is IoT?**

- According to ITU-T (International Telecommunication Union) [4], IoT is defined as: A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.
- According to IEEE (Institute of Electrical and Electronics Engineers) [5], IoT is defined as: A network of items – each embedded with sensors – which are connected to the internet.
- According to ETSI (European Telecommunication Standards Institute) [6]: Machine-to-Machine communications is the communication between two or more entities that does not necessarily need any direct human interventions. M2M services intend to automate decision and communication process.
- According to OASIS (The Organization for the Advancement of Structured Information Standards) [7], IoT is defined as: The System defined where the Internet is connected to the physical world via ubiquitous sensors.

### **2.2. IoT general definition**

The IoT envisions hundreds or thousands of end-devices with sensing, actuating, processing, and communication capabilities able to be connected to the Internet. These devices can be directly connected using cellular technologies such as 2G/3G/Long Term Evolution and beyond (5G) or they can be connected through a gateway, forming a local area network, to get connection to the Internet. The latter is the case where the end-devices usually form Machine to Machine (M2M) networks using various radio technologies[8].

IoT in four environments: Internet of Vehicles (IoV), Internet of Energy (IoE), Internet of Sensors (IoS), and Machine to Machine Communications (M2M) showing in the figure below.



## **Figure 1.1: Internet of Things (IoT).[9]**

### **2.1. Characteristics of the internet of thing**

The Fundamentals characteristics of the IoT are as follows[10]:

- **Interconnectivity**

With regard to the IoT, anything can be interconnected with the global information and communication infrastructure.

- **Things-related services**

The IoT is capable of providing thing-related services within the constraints of things, such as privacy protection and semantic consistency between physical things and their associated virtual things. In order to provide thing-related services within the constraints of things, both the technologies in physical world and information world will change.

- **Heterogeneity**

The devices in the IoT are heterogeneous as based on different hardware platforms and networks. They can interact with other devices or service platforms through different networks.

- **Dynamic changes**

The state of devices change dynamically, e.g., sleeping and waking up, connected and/or disconnected as well as the context of devices including location and speed. Moreover, the number of devices can change dynamically.

- **Enormous scale**

The number of devices that need to be managed and that communicate with each other will be at least an order of magnitude larger than the devices connected to the current Internet.

- **Safety**

As we gain benefits from the IoT, we must not forget about safety. As both the creators and recipients of the IoT, we must design for safety. This includes the safety of our personal data and the safety of our physical well-being. Securing the endpoints, the networks, and the data moving across all of it means creating a security paradigm that will scale.

- **Connectivity**

Connectivity enables network accessibility and compatibility. Accessibility is getting on a network while compatibility provides the common ability to consume and produce data.

## **2.2. IoT Application**

Most of the daily life applications that we normally see are already smart but they are unable to communicate with each other and enabling them to communicate with each other and share useful information with each other will create a wide range of innovative applications.

Few of these applications are [11]:

- **Smart Traffic System**

Traffic is an important part of a society therefore all the related problems must be properly addressed. There is a need for a system that can improve the traffic situation based on the traffic information obtained from objects using IoT technologies. For such an intelligent traffic monitoring system, realization of a proper system for automatic identification of vehicles and other traffic factors is very important for which we need IoT technologies instead of using common image processing methods. The intelligent traffic monitoring system will provide a good transportation experience by easing the congestion. It will provide features like theft-detection, reporting of traffic accidents, less environmental pollution.

- **Smart Home**

IoT will also provide DIY solutions for Home Automation with which we will be able to remotely control our appliances as per our needs. Proper monitoring of utility meters, energy and water supply will help saving resources and detecting unexpected overloading, water leaks etc. There will be proper encroachment detection system which will prevent burglaries. Gardening sensors will be able to measure the light, humidity; temperature, moisture and other gardening vitals, as well as it will water the plants according to their needs.

- **Smart Agriculture**

It will monitor Soil nutrition, Light, Humidity etc. and improve the green housing experience by automatic adjustment of temperature to maximize the production. Accurate watering and fertilization will help improving the water quality and saving the fertilizers respectively.

- **Smart Retailing and Supply chain Management.**

IoT with RFID (radio-frequency identification) provides many advantages to retailers. With RFID equipped products, a retailer can easily track the stocks and detect shoplifting. It can keep a track of all the items in a store and to prevent them from going out-of-stock, it places an order automatically. Moreover the retailer can even generate the sales chart and graphs for effective strategies.



### 2.3. IoT Standards and Protocols

For standardization and sake of implementation of protocols, these technologies are arranged in different layers. Typical internet networks follow Open Systems Interconnection (OSI) model [13] which is an ISO standard model for internet. The OSI model architectures the internet into seven layers - Physical, Data Link, Network, Transport, Session, Presentation and Application (table1.1). Though the actual implementation of OSI model is done through TCP-IP model which simplifies the seven-layer OSI model to four-layer internet protocol suite. In TCP-IP model (realistic implementation of OSI Model), the physical and data link layer are merged to form physical and network access layer and the session, presentation and application layers of OSI model are merged to a single application layer, as show in the table 1.2[12].

Number layer	Name layer	Explanation
Layer7	Application	Specifies how a particular application uses a network.
Layer 6	Presentation	Specifies how to represent data.
Layer 5	Session	Specifies how to establish communication with a remote system.
Layer 4	Transport	Specifies how to reliably handle data transfer.
Layer 3	Network	Specifies addressing assignments and how packets areforwarded.

<b>Layer 2</b>	<b>Data Link</b>	Specifies the organization of data into frames and how to send frames over a network.
<b>Layer 1</b>	<b>Physic</b>	Specifies the basic network hardware.

**Table 1.1:OSI Model Layers[12].**

**Table 1.2:TCP\IP Model Layers[12].**

### 2.3.1. Application Protocol

This is the highest layer within communication network. It is the interface between the (IoT) devices and the network. This layer is implemented through a

<b>Number layer</b>	<b>Name layer</b>	<b>Explanation</b>
<b>Layer 4</b>	<b>Application</b>	Specifies how a particular application uses a network.
<b>Layer 3</b>	<b>Transport</b>	Transport Specifies how to ensure reliable transport of data.
<b>Layer 2</b>	<b>Internet</b>	Internet Specifies packet format and routing.
<b>Layer 1</b>	<b>Network Access &amp; Physic</b>	Specifies frame organization and transmittal and the basic network hardware.

dedicated application at the device end. Like for a computer, application layer is implemented by the browser. It is the browser which implements application layer protocols[14].

The major application protocols such as CoAP, MQTT, DDS, AMQP and XMPP used in application protocol.

Protocols	UDP/TCP	Architecture	Security and QoS	Header Size (bytes)	Max Length(bytes)
<b>MQTT</b>	TCP	Pub/Sub	Both	2	5
<b>AMQP</b>	TCP	Pub/Sub	Both	8	-
<b>CoAP</b>	UDP	Req/Res	Both	4	20 (typical)
<b>XMPP</b>	TCP	Both	Security	-	-
<b>DDS</b>	TCP/UDP	Pub/Sub	QoS	-	-

**Table 1.3:**Application Layer Protocols.

#### **a. CoAP (Constrained Application Protocol)**

It is synchronous request/response application layer protocol that was designed by the Internet Engineering Task Force (IETF) [16] to target constrained-resource devices. It was designed by using a subset of the HTTP methods making it interoperable with HTTP; it makes use of the UDP protocol for lightweight implementation.

It also makes use of RESTful architecture, which is very similar to the HTTP protocol. It is used within mobiles and social network based applications and eliminates ambiguity by using the HTTP get, post, put and delete methods. Apart from communicating IoT data, CoAP has been developed along with DTLS for the secure exchange of messages. It uses DTLS for the secure transfer of data in the transport layer[15].

#### **b. Extensible Messaging and Presence Protocol (XMPP)**

Extensible messaging and presence protocol (XMPP) is a protocol that was originally designed for chats and messages exchange applications.

It is based on XML language and was standardized by IETF [15] more than a decade ago. Recently, its usage was extended for IoT and SDN applications due to the standardized use of XML which makes it easily extensible. XMPP supports both publish/subscribe and request/ response architecture. It is designed for near real-time applications and, thus, efficiently supports low-latency small messages [17].

### **c. Advanced Message Queuing Protocol (AMQP)**

The Advanced Message Queuing Protocol (AMQP) is another session layer protocol that was designed for financial industry. It runs over TCP and provides a publish/ subscribe architecture which is similar to that of MQTT. The difference is that the broker is divided into two main components: exchange and queues.

The exchange is responsible for receiving publisher messages and distributing them to queues based on pre-defined roles and conditions. Queues basically represent the topics and subscribed by subscribers which will get the sensory data whenever they are available in the queue[17].

### **d. Data Distribution Service (DDS)**

Data distribution service (DDS) is a messaging standards designed by the Object Management Group (OMG) [19]. It uses a publish/subscribe architecture and mostly used for M2M communications. It offers 23 qualities-of service levels which allow it to offer a variety of quality criteria, including: security, urgency, priority, durability, reliability, etc. It defines two sub layers: data-centric publish-subscribe and data-local reconstruction sub layers.

The first takes the responsibility of message delivery to the subscribers while the second is optional and allows a simple integration of DDS in the application layer. Publisher layer is responsible for sensory data distribution.

Data writer interacts with the publishers to agree about the data and changes to be sent to the subscribers. Subscribers are the receivers of sensory data to be delivered to the IoT application. Data readers basically read the published data and

deliver it to the subscribers and the topics are basically the types of data that are being published. In other words, data writers and data reader take the responsibilities of the broker in the broker-based architectures[18].

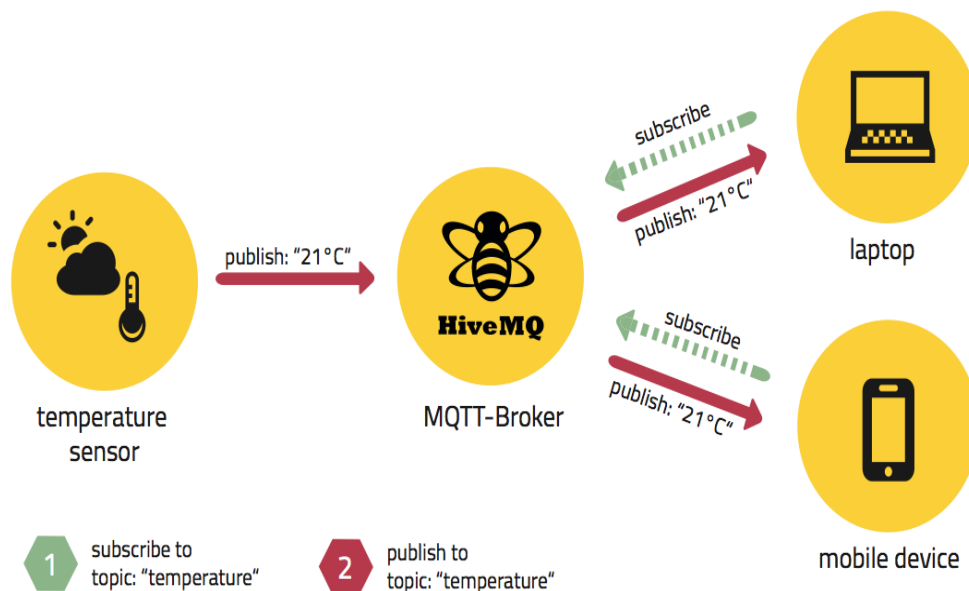
### e. MQTT (Message Queue Telemetry Transport)

MQTT was introduced by IBM[20] in 1999 and standardized by OASIS [7] in 2013. It is designed to provide embedded connectivity between applications and middleware's on one side and networks and communications on the other side. It follows a publish/subscribe architecture, as shown in Figure 1.2, where the system consists of three main components: publishers, subscribers, and a broker.

From IoT point of view, publishers are basically the lightweight sensors that connect to the broker to send their data and go back to sleep whenever possible.

Subscribers are applications that are interested in a certain topic, or sensory data, so they connect to brokers to be informed whenever new data are received.

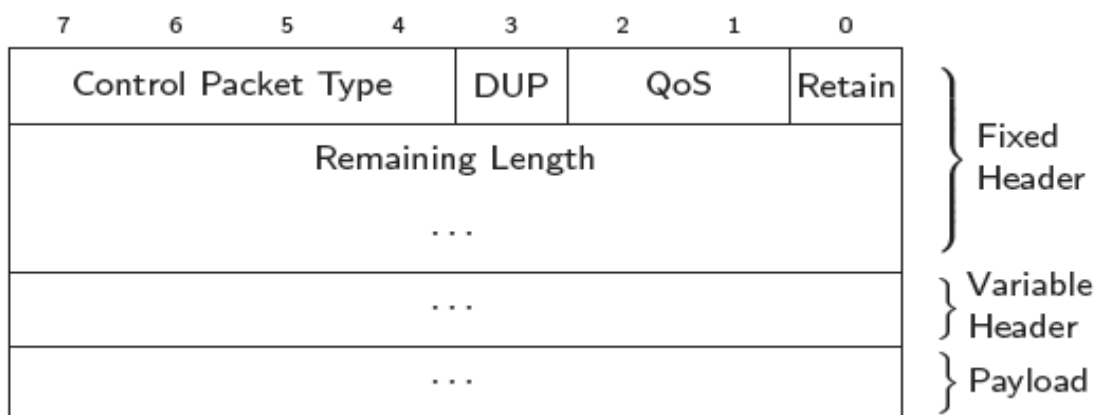
The brokers classify sensory data in topics and send them to subscribers interested in the topics[21].



**Figure 1.2:** MQTT Architecture "Publish/Subscribe". [22]

### ➤ MQTT Packet Structure

The MQTT protocol control Packet, consists of three parts; Fixed Header, Variable Header and Payload.



**Figure 1.3:** MQTT Packet Structure. [23]

### ➤ MQTT messaging

MQTT has 14 types of control signal, namely:

- CONNECT — Client request to connect to Server
- CONNACK — Connection Acknowledgement
- PUBLISH — A message which represents a new/separate publish
- PUBACK — QoS 1 Response to a PUBLISH message
- PUBREC — First part of QoS 2 message flow
- PUBREL — Second part of QoS 2 message flow
- PUBCOMP — Last part of the QoS 2 message flow
- SUBSCRIBE — A message used by clients to subscribe to specific topics

- SUBACK — Acknowledgement of a SUBSCRIBE message
- UNSUBSCRIBE — A message used by clients to unsubscribe from specific topics
- UNSUBACK — Acknowledgement of an UNSUBSCRIBE message
- PINGREQ — Heartbeat message
- PINGRESP — Heartbeat message acknowledgement
- DISCONNECT — Graceful disconnect message sent by clients before disconnecting

From those signals, there are only four main signals which are used directly by the client, namely PUBLISH, SUBSCRIBE, UNSUBSCRIBE, CONNECT. Other signals are part of the publish/subscribe mechanism[21].

### ➤ MQTT QoS

MQTT supports three service levels qualities:

- **QoS = 0:** means one delivery at most. The message is delivered according to the capabilities of the underlying network. No response is sent by the receiver and no retry is performed by the sender. The receiver gets the message either once or not at all.
- **QoS = 1:** means one delivery at least. This quality of service ensures that the message arrives at the receiver at least once, but there's a probability of duplicating messages on the receiver's side. If the publisher has not received the acknowledgment of delivery from the message broker, it sends the message again. After the duplicated message is received by the broker, the latter sends it again to all subscribers.
- **QoS = 2:** means one delivery exactly. This is the highest quality of service. It is used when neither loss nor duplication of messages are acceptable.[14]

### 2.3.2. Other IoT Protocols

- **IEEE 802.15.4e**

IEEE 802.15.4 is a data link standard that is commonly used in the MAC layer. The standard specifies the frame format, headers, destination and source addresses and identifies how the communication between the nodes can happen. The traditional frame formats used in networking are not suitable for power constrained IoT devices. In 2008, IEEE 802.15.4e was created to extend IEEE 802.15.4 and support low power communication. It uses time synchronization and channel hopping to enable high reliability, low cost communication in IoT data links [24].

- **IEEE 802.11 ah**

IEEE 802.11ah is a light (low-energy) version of the original IEEE 802.11 wireless medium access standard. It has been designed with less overhead to meet IoT requirements. IEEE 802.11 standards (also known as Wi-Fi) are the most commonly used wireless standards. They have been widely used and adopted for all digital devices including laptops, mobiles, tablets, and digital TVs. However, the original WiFi standards are not suitable for IoT applications due to their frame overhead and power consumption. Hence, IEEE 802.11 working group initiated 802.11ah task group to develop a standard that supports low overhead, power friendly communication suitable for sensors and motes[24].

- **Bluetooth Low Energy**

Bluetooth low energy, or Bluetooth smart it is another short-range communication standard for data link layer that is widely used in IoT. It is mostly used in in-vehicle networking. It has a small latency that is 15 times smaller than original Bluetooth standards. Its low energy can reach ten times less than the classic Bluetooth. Its access control uses a contention-less MAC with low latency and fast transmission. It adopts master/slave architecture and offers two types of frames: advertising and data frames. The advertising frame is used for discovery and is sent by slaves on one or more of dedicated advertisement channels. Master nodes sense advertisement channels to find



slaves and connect them. After connection, the master tells the slave its waking cycle and scheduling sequence. Nodes are awake usually only when they are communicating and they go to sleep otherwise to save their power [25].

- **Zigbee Smart Energy**

ZigBee smart energy is designed for a large range of IoT applications including smart homes, remote controls and healthcare systems. It supports a wide range of network topologies including star, peer-to-peer, or cluster-tree. A coordinator controls the network and is the central node in a star topology, the root in a tree or cluster topology and may be located anywhere in peer-to-peer. ZigBee standard defines two stack profiles: ZigBee and ZigBee Pro. These stack profiles support full mesh networking and work with different applications allowing implementations with low memory and processing power. ZigBee Pro offers more features including security using symmetric-key exchange, scalability using stochastic address assignment, and better performance using efficient many-to-one routing mechanisms [26].

- **6LoWPAN**

6LoWPAN, an acronym for IPv6 over low power wireless personal area networks, is a very popular standard for wireless communication. It enables communication using IPv6 over the IEEE 802.15.4 protocol. This standard defines an adaptation layer between the 802.15.4 link layer and the transport layer. 6LoWPAN devices can communicate with all other IP based devices on the Internet. The choice of IPv6 is because of the large addressing space available in IPv6. 6LoWPAN networks connect to the Internet via a gateway (WiFi or Ethernet), which also has protocol support for conversion between IPv4 and IPv6 as today's deployed Internet is mostly IPv4. IPv6 headers are not small enough to fit within the small 127 byte MTU of the 802.15.4 standard. Hence, squeezing and fragmenting the packets to carry only the essential information is an optimization that the adaptation layer performs [12].

- **IPv6 over Bluetooth Low Energy**

RFC 7668 specifies the format of IPv6 over Bluetooth low energy, which was discussed in Subsection II-E. It reuses most of the 6LowPAN compression techniques. Fragmentation is done at the logical link control and adaptation protocol (L2CAP) sub layer in Bluetooth. Thus, the fragmentation feature of 6LowPAN is not used here. Further, Bluetooth low energy does not currently support formation of multi-hop networks at the link layer. Instead, a central node acts as a router between lower-powered peripheral nodes. Thus, multi-hop feature in 6LowPAN is not used as well [24].

- **RPL**

Routing protocol for low-power and lossy networks (RPL) is an open routing protocol, based on distance vectors. It describes how a destination oriented directed acyclic graph (DODAG) is built with the nodes after they exchange distance vectors.

A set of constraints and an objective function is used to build the graph with the best path.

- The objective function and constraints may differ with respect to their requirements. For example, constraints can be to avoid battery powered nodes or to prefer encrypted links. The objective function can aim to minimize the latency or the expected number of packets that need to be sent [12].

- **CORPL**

An extension of RPL is CORPL, or cognitive RPL, which is designed for cognitive networks and uses DODAG topology generation but with two new modifications to RPL. CORPL utilizes opportunistic forwarding to forward the packet by choosing multiple forwarders (forwarder set) and coordinates between the nodes to choose the best next hop to forward the packet to. DODAG is built in the same way as RPL. Each node maintains forwarding set instead of its parent only and updates its neighbor with its changes using DIO messages.

Based on the updated information, each node dynamically updates its neighbor priorities in order to construct the forwarder set [24].

- **CARP**

Channel-Aware Routing Protocol is a nonstandard distributed routing protocol used in Underwater Wireless Sensor Networks (UWSNs).

Its assets include delivering packets in reasonable time with low energy demands. In addition, it is able to support link quality information that is calculated from historical successful data transfers.

The history is collected from adjacent sensors in order to choose the forwarding nodes. The main weakness of CARP is that it does not allow reusing previously gathered data.

An enhancement of CARP is denoted as E-CARP.

E-CARP allows the sink node to save previously received sensor data. Hence; ECARP drastically decreases the communication overhead [26].

- **DNS-SD (DNS-Service Discovery)**

This protocol stack uses standard DNS messages to discover services in an IOT network. Based on mDNS, DNS-SD is used to resolve services available in a network. The service discovery is implemented in two steps - in the first step, host names of the service providers are resolved and in the next step, IP addresses are paired with the host names using mDNS. It is important to identify host names as IP addresses can change in the network [27].

- **mDNS (Multicast Domain Name System)**

Multicast Domain Name System (mDNS) is a DNS like service discovery protocol to resolve host names to IP addresses in a local network without using

any unicast DNS server. It can be used without any additional infrastructure or DNS server in the network. The protocol operates on IP multicast UDP packets through which a node in the local network enquires the names of all other nodes. The client node sends a query message to respond by a node with specific name. When the node with the corresponding name receives the query, it responds with a multicast response message containing its IP address. Being a multicast response, the target device IP address and name is also saved by all the devices (nodes) of the network in their local caches [27].

### 3. Advantages and Disadvantages of IoT

#### ➤ Advantages [28]

- **Information:** To make better decision, we need to have more information. So as we all know that knowledge will help us take better and faster decisions. Suppose vegetables in the vegetable basket are going to get empty soon, so our smart basket will send us an SMS to inform us to get vegetables from the stores.
- **Tracking:** Another advantage of IoT is tracking. It provides advance level information that could not have been possible before this so easily. Let us take an example of medical store, the application will inform the store keeper about the upcoming expiration dates of the medicines, so that they can get replaced or whatever.
- **Time:** IoT saves more time which we generally used to get it wasted on gathering and processing information so that they can be accurately analyzed, in order to get better decision.
- **Money:** If the cost of tagging and monitoring equipment goes down than the market for IoT will cross-skies in a very short period.

#### ➤ Disadvantages [28]

- **Compatibility:** In current time, there is no universal standard of compatibility and facility for the tagging and monitoring devices or equipment. So the disadvantages of IoT is that as the number and nature of devices available in market, soon it will be getting tough to connect them using IPv4.

- **Complexity:** With the help of all complex systems, there are more and more chances of failure, suppose in the vegetable market app, if the application send message about vegetable basket getting empty to two or more people with whom it is associated with them. Then both the people will go to the shop to get the vegetable as asked by app. In such a case, it may be possible that the unnecessarily double purchase of the item may be done by the people.
- **Safety:** It is necessary to provide safety, else if it expired product id medicated to the patient then the ill reaction will responded by the body and damaging health.
- **Bandwidth:** It can be a problem for IoT applications, as it is limited.

## 4. IoT Challenges

- **Architecture Challenge**

Architectures should be open, and following standards, they should not restrict users to use fixed, end-to-end solutions. IoT architectures should be flexible to cater for cases such as identification (RFID, tags), intelligent devices, and smart objects (hardware and software solutions).

- **Technical Challenge**

IoT technology can be complex for variety of reasons. First, there are legacy heterogeneous architectures. Second, communication technologies, including fixed and mobile communication systems, power line communications, wireless communication, and short-range wireless communication technologies. At last, there are thousands of different applications; it is in natural to have different requirements on what parties need to communicate with each other, what kind of security solutions are appropriate, and so on.

- **Hardware Challenge**

Smart devices with enhanced inter-device communication will lead to smart systems with high degrees of intelligence. Its autonomy enables rapid deployment of IoT applications and creation of new services. Therefore, hardware researches are focusing on designing wireless identifiable systems with low size, low cost yet sufficient functionality. Hardware and protocol code sign for sleeping has been thus the first hardware challenge of IoT.

- **Privacy and Security Challenge**

Compared with traditional networks, security and privacy issues of IoT become more prominent. Much information includes privacy of users, so that protection of privacy becomes an important security issues in IoT. Because of the combinations of things, services, and networks, security of IoT needs. Existing security architecture is designed from the perspective of human communication, may not be suitable and directly applied to IoT system. Using existed security mechanisms will block logical relationship between things in IoT.

- **Standard Challenge**

Standards play an important role in forming IoT. standards developed by cooperated multiparties, and information models and protocols in the standards, shall be open. The standard development process shall also be open to all participants, and the resulting standards shall be publicly and freely available. In today's network world, global standards are typically more relevant than any local agreements.

- **Business Challenge**

For a mature application, its business model and application scenario are clear and easy to be mapped into technical requirements. So the developers do not need to spend much time on business-related aspects. But for IoT, there are too many possibilities and uncertainties in business models and application scenarios. It is thus inefficient in terms of business-technology alignment, and one solution will not fit possibilities for all. The IoT is a challenging traditional business model [29].

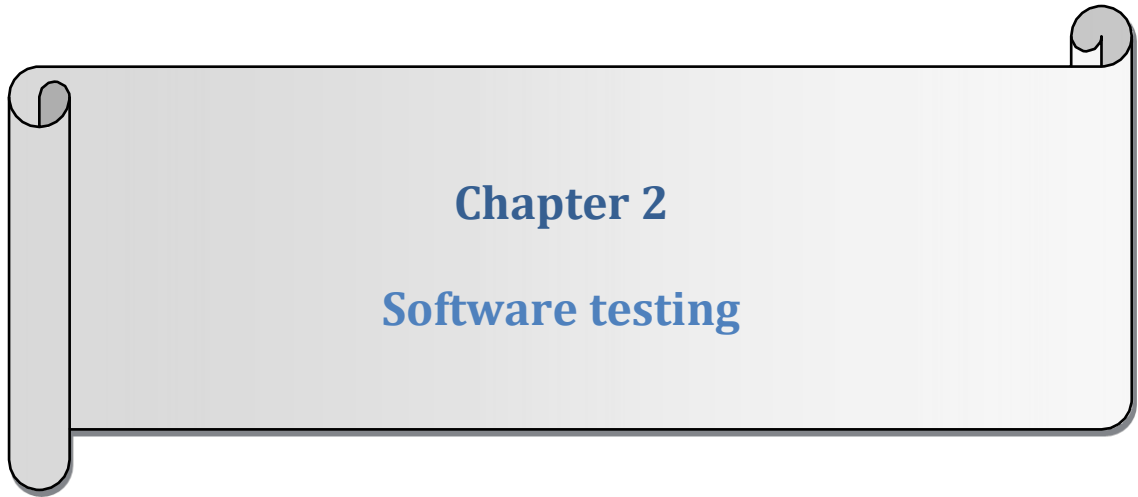
## **5. Solutions for IoT**

As far as we are concerned with the traditional storage of the data. Now as we are dealing with the decreasing inefficiency of cloud computing, there is increase of burden on the cloud servers due to IoT data being processed over there. So the solution to the big data problem is to replace cloud computing with fog computing, in which all the processing and analytics works are Internet of Things (IoT): In a Way of Smart World done on its respective routers instead of cloud servers, as a result all the data in the cloud become structured data. And the duty of the cloud server will get limited to making the data reachable the application device. For the challenge of security and privacy, we will have to increase the number of bytes being encrypted [28].

## **6. Conclusion**

In this chapter, we have discussed in generally the concept of the four trends of the Internet of things, first of all we have defined in the first part Internet of thing, and its Characteristics. In the second part we have defined the Standards and Protocols of IoT and the definition of each protocol. In the third part we address some challenges of IoT.

In the next chapter, we will define in details software testing in details, conformance testing and TTCN-3.



## **Chapter 2**

### **Software testing**



## **1. Introduction**

Testing is a fundamental step in any development process. It is a process of executing a program with the aim of finding error. To make our software perform well, it should be error free. If testing is done successfully, it will remove all the errors from the software.

## **2. Software testing**

Test is the search for mismatches between the behavior of the program and the expected behavior calculated from pre-established. Testing is defined as a process of evaluation that either the specific system meets its originally specified requirements or not. It is mainly a process encompassing validation and verification process that whether the developed system meets the requirements defined by user. Therefore, this activity results in a difference between actual and expected result.

Software testing refers to finding bugs, errors or missing requirements in the developed system or software [30].

## **3. Objectives of software testing**

- To ensure that application works as specified in requirements document.
- To provide a bug free application.
- To achieve the customer satisfaction.
- To ensure that error handling has been done gracefully in the application.
- To establish confidence in software.
- To evaluate properties of software.
- To discuss the distinctions between validation testing and defect testing.
- To describe strategies for generating system test cases.
- To understand the essential characteristics of tool used for test automation[30].

## **4. Verification and Validation**

It would not be right to say that testing is done only to find faults. Faults will be found by everybody using the software. Testing is a quality control measure used to verify that a product works as desired. Software testing provides a status report of

the actual product in comparison to product requirements (written and implicit). Testing process has to verify and validate whether the software fulfills conditions laid down for its release/use. Testing should reveal as many errors as possible in the software under test, check whether it meets its requirements and also bring it to an acceptable level [31].

## **5. Levels of testing**

Software testing integrates software test case design methods into a well-planned series of steps that result in successful construction of software that result in successful construction of software. Software testing levels gives the road map for testing. A software testing level should be flexible enough to promote a customized testing approach at same time it must be right enough. It is generally developed by project managers, software engineer and testing specialist. There are four different software testing strategies [32].

### **5.1. Unit testing**

This type of testing is performed at the bottom level by the developers before it is moved to the team of testing to execute the test cases. It is the smallest module that can be tested and verified at the each section or lines of code. In this output of one module becomes the input of another module. If the output of any one of the module fails so then the output to which we give the input also fails[33].

### **5.2. Integration testing**

Integration testing is performed immediately after the unit testing. In this all the modules are merged together to form a larger module and determine are they functioning in a proper way and then the testing is implemented on the modules. Testing is done so that in case if any bug remained in the unit testing it can be again tested in this testing so as to remove all the bugs[34].

### **5.3. System testing**

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic System testing is actually a

series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated function [35].

#### 5.4. Acceptance testing

Acceptance is used to conduct operational readiness of a product, service or system as part of a quality management system. It is a common type of non-functional software testing, used mainly in software development and software maintenance projects. This type of testing focuses on the operational readiness of the system to be supported. It is done when the completed system is handed over from the developers to the customers or users. The purpose of acceptance testing is rather to give confidence that the system is working than to find errors [36].

### 6. Categories of testing types

Test procedures often differ greatly between functional, structural tests and Non-functional testing. Traditionally, functional tests have been termed Black box tests, and structural tests White box tests. Between the two, one finds a large group of tests which do not fit into either category. These tests employ both functional and structural approaches as needed for test case selection and for developing test procedures, these increasingly important methods are called Gray box tests [37].

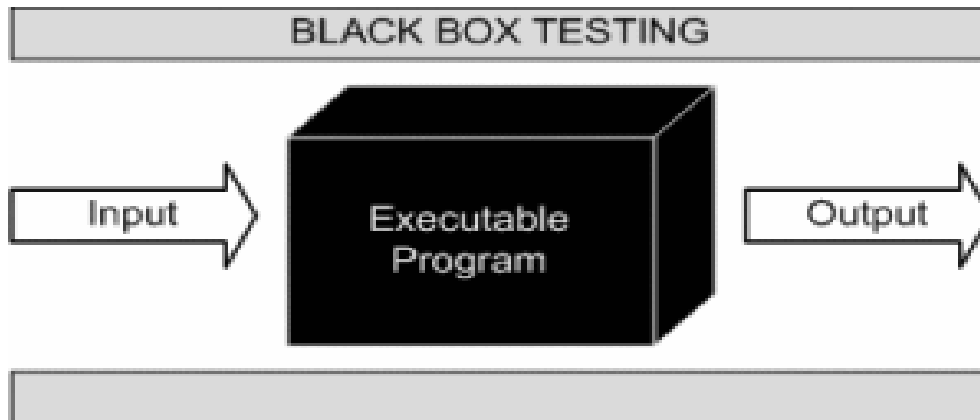


**Figure 2.1:** An overview of testing methods. [37]

#### 6.1. Black box testing

Black box testing is named so because as we know that in the tester's eyes it is named black box but inner side no one sees.

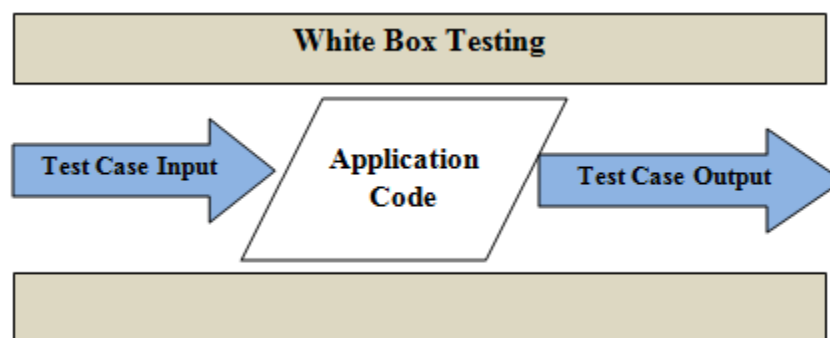
With black box testing, the outside world comes into contact with the test item (a program or program unit) only through a specified interface. This interface can be the application interface, an internal module interface, or the INPUT/OUTPUT description of a batch process. Black box tests check whether interface definitions are adhered to in all situations[38]



**Figure 2.2:** Black box testing. [38]

### 6.2. White box testing

In the case of white box testing, the inner workings of the test item are known. The test cases are created based on the knowledge. White box tests are thus developer tests. They ensure that each implemented function is executed at least once and checked for correct behavior. Examination of White box testing results can be done with the system specifications in mind [38].



**Figure 2.3:** White box testing. [38]

### 6.3. Gray box testing

Gray box testing, like black box testing, is concerned with the accuracy and dependability of the input and output of the program or module.

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application [39].

#### **6.4. Non-Functional testing**

Non-functional testing describes the tests used to measure the software characteristics such as response time, page load times, peak load limit, threshold limit for optimum performance of the software product on varying scale etc[40].

Non-functional testing includes different testing types as mentioned below:

##### **6.4.1. Performance testing**

Performance testing involves all the phases as the mainstream testing life cycle as an independent discipline which involve strategy such as plan, design, execution, analysis and reporting. Not all software has specification on performance explicitly. But every system will have implicit performance requirements. Performance has always been a great concern and driving force of computer evolution. The goals of performance testing can be performance bottleneck identification, performance comparison and evaluation. By performance testing we can measure the characteristics of performance of any applications. One of the most important objectives of performance testing is to maintain a low latency of a website, high throughput and low utilization[41].

Performance testing is in general a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage[41].

Some of the main goals of performance testing are[41]:

- Measuring response time of end to end transactions.
- Measurement of the delay of network between client and server.
- Monitoring of system resources which are under various loads.

Some of the common mistakes which happen during performance testing are:

- Ignoring of errors in input.
- Analysis is too complex.
- Erroneous analysis.
- Level of details is inappropriate.
- Ignore significant factors.
- Incorrect Performance matrix.
- Important parameter is overlooked.
- Approach is not systematic.

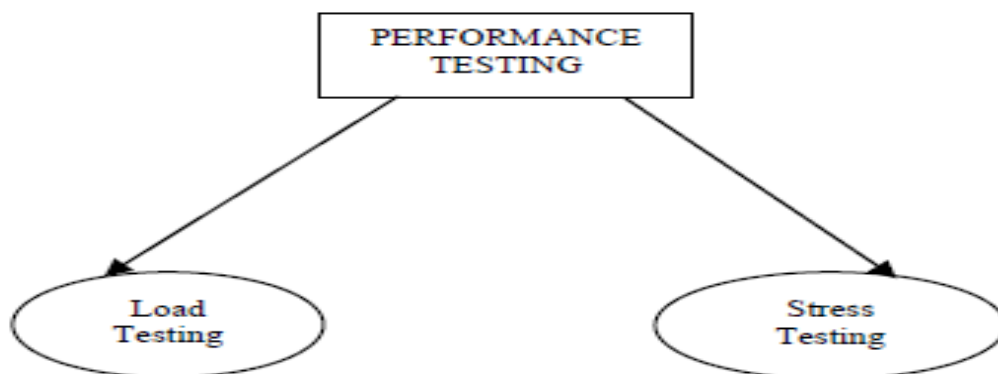
➤ The focus of performance testing is checking a software program[41]:

**Speed:** Determines whether the application responds quickly.

**Scalability:** Determines maximum user load the software application can handle.

**Stability:** Determines if the application is stable under varying loads.

There are two kinds of performance testing:



**Figure 2.4:** Types of performance testing.[42]

### ➤ **Load testing**

Load testing is an industry term for the effort of performance testing. The main feature of the load testing is to determine whether the given system is able to handle the anticipated no of users or not. The main objective of load testing is to check whether the system can perform well for specified user or not.

Load testing is also used for checking an application against heavy load or inputs such as testing of website in order to find out at what point the website or applications fails or at what point its performance degrades[42].

Examples:

- Testing a word processor by editing a very large document.
- For web application load is defined in terms of concurrent users or HTTP connections

### ➤ **Stress testing**

We can define stress testing as performing random operational sequence, at larger than normal volume, at faster than normal speed and for longer than normal periods of time, as a method to accelerate the rate of finding defects and verify the robustness of our product, or we can say stress testing is a testing, which is conducted to evaluate a system or component at or beyond the limits of its specified requirements to determine the load under which it fails and how.

Stress testing also determines the behavior of the system as user base increases. It is trying to break the system under test by overwhelming its resources or by taking resources away from it. Purpose is to make sure that the system fails and recovers gracefully[42].

Example:

- Double the baseline number for concurrent users/HTTP connections.
- Randomly shut down and restart ports on the network switches/routers that connect servers.

## **6.4.2. Security testing**

Software quality, reliability and security are tightly coupled. Flaws in software can be exploited by intruders to opens security holes. Security testing makes sure that only the authorized personnel can access the program and only the authorized personnel can access the functions available to their security level. The security testing is performed to check whether there is any information leakage in the sense by encrypting the application or using wide range of software's and hardware's and firewalls etc[40].

### **6.4.3.Recovery testing**

Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic, re-initialization, check pointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair is evaluated to determine whether it is within acceptable limits[40].

## **7. Limitations of testing**

Limitations are principles that limit the extent of something. Testing also has some limitations that should be taken into mind to set realistic expectations about its benefits. In spite of being most dominant verification technique, software testing has following limitations:

- Testing can be used to show the presence of errors, but never to show their absence. It can only identify the known issue or errors. It gives no idea about defects still uncovered. Testing cannot guarantee that the system under test is error free.
- Testing provides to help when we have to make a decision to either “release the product with errors for meeting the deadline” or to “release the product late compromising the deadline”.
- Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions [43].

## **8. TTCN-3 language**

TTCN was originally called the Tree and Tabular combined Notation, but has



since been renamed to the Test and Testing Control Notation. It is a purpose-built language with the sole purpose of implementing testing.

TTCN-3 is a flexible and powerful language applicable to the specification of all types of reactive system tests over a variety of communication interfaces. Typical areas of application are protocol testing (including mobile and Internet protocols), service testing (including supplementary services), module testing, testing of CORBA based platforms, API testing, etc. TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing[44].

### **8.1. Key TTCN-3 language features**

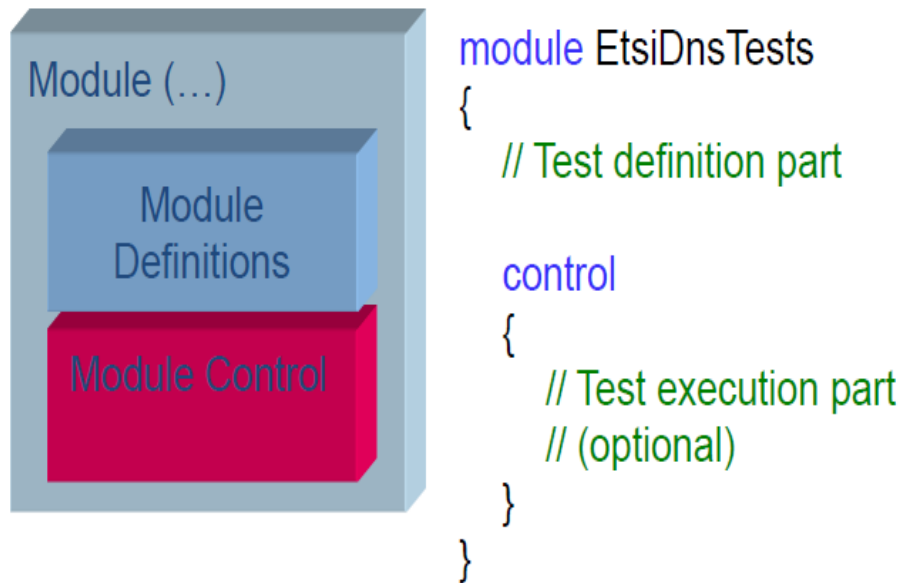
TTCN-3 is a high-level, abstract language. The code itself is platform independent, as well as test environment independent. In TTCN-3, you define only the abstract messages/signals as they are exchanged between the test system and the tested entity. The transport layers and connections are provided and handled by the tools Message encoding (serialization) and decoding (deserialization) is part of the tool/environment, and not part of the test definition itself[45].

The strong points of TTCN-3 for use in protocol tests are:

- A rich data/type system
- Parametric templating and powerful template matching
- Behavior specification using the alt and default behaviors

### **8.2. Language basics**

The most basic construct in TTCN-3 is the module. A module can contain a whole test suite or it may contain library code. Such library code can be used in other modules through



**Figure 2.5:** The TTCN-3 Module Structure. [46]

The import statement can be specified finely grained and the reforeallows minimal interfaces between modules. Modules consist of a definitions part and a control part (of which both are optional). The definitions part contains declarations1 for constants, types, templates, functions, altsteps, testcases, signatures and module parameters..

TTCN-3 contains a very large number of built-in types. The most basic data types and most widely used ones are the type's integer, boolean, charstring, float and record. The built-in data types can be used to declare user-defined types. Both, built-in data types as well as userdefined types can be used to declare templates (data descriptions) which are subsequently used for the message-based communication using the send and receive operations on the ports of a test component.

Constants, types, functions and basic statements (such as conditionals, assignments or loops) are similar to other well-known programming languages[46].

### 8.3. The concepts of TTCN-3

The TTCN-3 core language is a modular language which has a similar look and feel to a typical programming language like, e.g. C or C++. In addition to the typical programming language constructs, it contains all important features necessary to specify test procedures and campaigns like test verdicts to assess test runs, matching mechanisms to compare the reactions of the SUT with the expected outputs,

timer handling to specify time restrictions, the handling of test components to support distributed testing, the ability to specify encoding information, support for different kinds of communication (i.e. synchronous and asynchronous communication) and the possibility to log test information during a test run[47].

### 8.3.1. Subtypes

Subtypes are used to restrict the allowed range of an existing type declaration. This can avoid manual value verifications as subtype violations automatically lead to an error either at compile time or runtime. The TTCN-3 types are summarized in Figure 2.7[48].

Class of type	Keyword	Subtype
Simple basic types	integer	range, list
	float	range, list
	boolean	List
	verdicttype	List
Basic string types	bitstring	list, length
	hexstring	list, length
	octetstring	list, length
	charstring	range, list, length, pattern
	universal charstring	range, list, length, pattern
Structured types	record	list (see note)
	record of	list (see note), length
	set	list (see note)
	set of	list (see note), length
	enumerated	list (see note)
	union	list (see note)
Special data type	anytype	list
Special configuration types	address	
	port	
	component	
Special default type	default	
Array notation	[]	list (see note)
NOTE: List subtyping of these types is possible when defining a new constrained type from an already existing parent type but not directly at the declaration of the first parent type.		

Figure 2.6: Overview of TTCN-3 types.[49]

### 8.3.2. Components

A test configuration in TTCN-3 consists of one or more test components. Each component contains one or more communication ports which describe its interface[48].

Example [49]:

```
type port MyPort message {  
  
  incharstring }  
  
type component MyComponent {  
  
  portMyPortmsgInPort ; }  
  
type component MyExtendedComponent extends MyComponent {  
  
  var integer myValue ;}
```

### 8.3.3. Test cases

A test case is complete and independent specification of the actions required to achieve a specific test purpose. It typically starts in a stable testing state and ends in a stable testing state.

In TTCN-3, test cases are a special kind of function. Test cases define the behavior, which have to be executed to check whether the SUT passes a test or not. This behavior is performed by the MTC which is automatically created when a test case is being executed[49].

Example [48]:

```
testcasemyTestcase ( ) runs on myComponent {  
  
  setverdict ( f a i l ) ;  
  
  }  
  
control {  
  
  varverdicttype v := execute ( myTestcase ) ;  
  
  }
```

### 8.3.4. Templates

Templates are used to either transmit a set of distinct values or to test whether a set of received values matches the template specification. Templates can be defined globally or locally[49].

Example[48]:

```
type record PersonType {  
  charstring f i r s tName ,  
  charstringmiddleName optional ,  
  charstringlastName  
}  
  
templatePersonType Turing {  
  f i r s tName := " Alan " ,  
  middleName := omit ,  
  lastName := " Turing "  
}  
  
templatePersonTypeTuringFull modifies Turing {  
  middleName := Mathison }  
  
templatePersonTypeMyPerson ( charstring p f i r s tName , charstring p lastName ) {  
  f i r s tName := p f i r s tName ,  
  middleName := omit ,  
  lastName := p lastName
```

### 8.3.5. Alt statements

An alt statement expresses sets of possible alternatives that form a tree of possible execution paths[49].

Example[48]:

```
testcasemyTestcase ( ) runs on myComponent {  
  
alt {  
  
[ ] pt .receive ( expectedMes sage ) ;  
  
pt .send(answerMessage ) ;  
  
}  
  
[ ] any port .receive {  
  
setverdict ( f a i l ) ;  
  
}  
  
}  
  
}
```

## 9. Conclusion:

Software testing is very important phase of Software Development Life Cycle (SDLC). Software testing is vital element in the SDLC and can furnish excellent results if testing is done properly and effectively. Unfortunately, software testing is often less formal and rigorous than it should and a main reason for that is because we have struggled to define best principles, standards for optimal software testing.

To perform software testing effectively and efficiently, everyone involved with testing should be familiar with basic software testing importance, types and levels and categories. Already lot of work has been done in this chapter, and even continues by define all concepts of TTCN-3 language.

In the next chapter which is the last one, we willspecify the standards of MQTT and its messages, then we will explain how we transfer these messages in TTCN-3, finally we will represent our im



## **Chapter 3**

# **Conception and implementation of system**

## 1. Introduction

The MQ Telemetry Transport (MQTT) is a lightweight publish/subscribe protocol flowing over TCP/IP for remote sensors and control devices through low bandwidth communications. MQTT is used by specialized applications on small footprint devices that require a low bandwidth communication, typically for remote data acquisition and process control. This kind of implementations of networking protocol stacks is in need of thorough testing in order to ensure not only its security, but also its interoperability and compliance to relevant standards and specifications.

In this chapter we will present the conception of MQTT messages Packet, and the implementation of our project. First, we will present the specification of MQTT version 3-1-1, then how to verify a MQTT message using TTCN-3. Finally, we will describe the implementation of our system.

## 2. Structure of an MQTT control packet

The MQTT protocol works by exchanging a series of MQTT Control Packets in a defined way. This section describes the format of these packets.

An MQTT Control Packet consists of up to three parts, always in the following order as illustrated in Table 3.1[50]:

Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

**Table 3.1:** Structure of an MQTT Control Packet[51].



### 3. MQTT message format

As mentioned in the previous sentence, the message header for each MQTT command message is contained a fixed header. Some messages also require a variable header and a payload. We will describe all of that in details in the following sections[50].

#### 3.1. Fixed header

The message header for each MQTT command message contains a fixed header. The table below shows the fixed header format.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

**Table 3.2:** Fixed Header Format[51].

**Byte 1:** Contains the MQTT Control Packet type and Flags specific to each MQTT Control Packet type.

**Byte 2:** (At least one byte) contains the Remaining Length field.

The fields are described in the following sections. All data values are in big-endian order: higher order bytes precede lower order bytes. A 16-bit word is presented on the wire as Most Significant Byte (MSB), followed by Least Significant Byte (LSB)[51].

**MQTT Control Packet type** (Position: byte 1, bits 7-4)

Represented as a 4-bit unsigned value, the values are listed in the table below[51].

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment

PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

**Table 3.3:**MQTT Control Packet type[50].

## Flags

The remaining bits [3-0] of byte 1 in the fixed header contain flags specific to each MQTT Control Packet type as listed in the Table 3.4 below[50].

Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP	QoS	QoS	RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0

PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

**Table 3.4:Flag Bits[50].**

Where a flag bit is marked as “Reserved” in Table 3.4, it is reserved for future use.

**DUP** (Position: byte 1, bit 3)

This flag is set when the client or broker attempts to re-deliver a PUBLISH message. This applies to messages where the value of QoS is greater than zero (0), and an acknowledgment is required. When the DUP bit is set, the variable header includes a Message ID.

**QoS** (Position: byte 1, bits 2-1)

This flag indicates the level of assurance for delivery of a PUBLISH message. The QoS levels are shown in the table below[50].

QoS value	bit 2	bit 1	Description		
0	0	0	At most once	Fire and Forget	$\leq 1$
1	0	1	At least once	Acknowledged delivery	$\geq 1$
2	1	0	Exactly once	Assured delivery	$= 1$
3	1	1	Reserved		

**Table 3.5:**The QoS levels[50].

**RETAIN** (Position: byte 1, bit 0)

When set, the Retain flag indicates that the broker holds the message, and sends it as an initial message to new subscribers to this topic. This means that a new client connecting to the broker can quickly establish the current number of topics. This is useful where publishers send messages on a "report by exception" basis, and it might be some time before a new subscriber receives data on a particular topic. The data has a value of retained or Last Known Good (LKG)[50].

**Remaining Length** (Position: byte 2)

Represents the number of bytes remaining within the current message, including data in the variable header and the payload.

The variable length encoding scheme uses a single byte for messages up to 127 bytes long. Longer messages are handled as follows. Seven bits of each byte encode the Remaining Length data, and the eighth bit indicates any following bytes in the

representation. Each byte encodes 128 values and a "continuation bit". For example, the number 64 decimal is encoded as a single byte, decimal value 64, hex 0x40. The number 321 decimal (=128x2 + 65) is encoded as two bytes, least significant first. The first byte is 2+128 = 130. Note that the top bit is set to indicate at least one following byte. The second byte is 65.

The protocol limits the number of bytes in the representation to a maximum of four. This allows applications to send messages of up to 268 435 455 (256 MB). The representation of this number on the wire is: 0xFF, 0xFF, 0xFF, 0x7F.

The table below shows the Remaining Length values represented by increasing numbers of bytes[50].

Digits	From	To
1	0 (0x00)	127 (0x7F)

Digits	From	To
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

**Table 3.6:**Remaining Length values[51].

Remaining Length encoding is not part of the variable header. The number of bytes used to encode the Remaining Length does not contribute to the value of the Remaining Length. The variable length "extension bytes" are part of the fixed header, not the variable header.

### 3.2. Variable header

Some types of MQTT Control Packets contain a variable header component. It resides between the fixed header and the payload. The content of the variable header varies depending on the Packet type. The Packet Identifier field of variable header is common in several packet types[51].

#### ➤ Packet identifier

The variable header component of many of the Control Packet types includes a 2 byte Packet Identifier field. These Control Packets are PUBLISH (where QoS > 0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK.

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

**Table 3.7:**Packet Identifier bytes[50].

MQTT Control Packet types that require a Packet Identifier are listed in Table 3.8:

MQTT Control Packet type	Packet Identifier field
CONNECT	NO
CONNACK	NO
PUBLISH	YES (If QoS > 0)
PUBACK	YES
PUBREC	YES
PUBREL	YES
PUBCOMP	YES
SUBSCRIBE	YES
SUBACK	YES
UNSUBSCRIBE	YES
UNSUBACK	YES
PINGREQ	NO
PINGRESP	NO
DISCONNECT	NO

**Table 3.8:**MQTT Control Packet types that contains a Packet Identifier[50].

### 3.3. Payload

Some MQTT Control Packet types contain a payload as the final part of the packet, as described in Chapter 3. In the case of the PUBLISH packet this is the Application Message.

MQTT Control Packet type	Payload
CONNECT	Required
CONNACK	None
PUBLISH	Optional
PUBACK	None
PUBREC	None
PUBREL	None

PUBCOMP	None
SUBSCRIBE	Required
SUBACK	Required
UNSUBSCRIBE	Required
UNSUBACK	None
PINGREQ	None
PINGRESP	None
DISCONNECT	None

**Table 3.9:** Payload for each Message Types[51].

## 4. Some of command messages

### 4.1. CONNECT(Client requests a connection to a Server)

After a Network Connection is established by a Client to a Server, the first Packet sent from the Client to the Server MUST be a CONNECT Packet.

A Client can only send the CONNECT Packet once over a Network Connection. The Server MUST processes a second CONNECT Packet sent from a Client as a protocol violation and disconnects the Client[50].

The fixed header is represented in the table below:

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (1)				Reserved			
	0	0	0	1	0	0	0	0
byte 2...	Remaining Length							

**Table 3.10:** Fixed Header of the CONNECT Packet[51].

The variable header for the CONNECT Packet consists of four fields in the following order: Protocol Name, Protocol Level, Connect Flags, and Keep Alive.

	Description	7	6	5	4	3	2	1	0
Protocol Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (4)	0	0	0	0	0	1	0	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1
byte 5	'T'	0	1	0	1	0	1	0	0
byte 6	'T'	0	1	0	1	0	1	0	0
Protocol Level									
byte 7	Level(4)	0	0	0	0	0	1	0	0
Connect Flag									
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved	
byte 8	X	X	X	X	X	X	X	0	
Keep Alive									
byte 9	Keep Alive MSB								
byte 10	Keep Alive LSB								

**Table 3.11:** Variable Header of the CONNECT Packet[51].

The payload contains one or more encoded fields. They specify a unique Client identifier for the Client, a Will topic, Will Message, User Name and Password. All but the Client identifier are optional and their presence is determined based on flags in the variable header.

### Response

The broker sends a CONNACK message in response to a CONNECT message from a client. If the client does not receive a CONNACK message from the broker within a "reasonable" amount of time, the client closes the TCP/IP socket connection, and restarts the session by opening a socket to the broker and issuing a CONNECT



message. A "reasonable" amount of time depends on the type of application and the communications infrastructure[50].

#### 4.2. PUBLISH (Publish message)

A PUBLISH Control Packet is sent from a Client to a Server or from Server to a Client to transport an Application Message.

##### Fixed header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

**Table 3.12:** Fixed Header of the PUBLISH Packet[50].

##### Variable header

The variable header contains the following fields in the order: Topic Name, Packet Identifier.

##### Payload

The Payload contains the Application Message that is being published. The content and format of the data is application specific. The length of the payload can be calculated by subtracting the length of the variable header from the Remaining Length field that is in the Fixed Header. It is valid for a PUBLISH Packet to contain a zero length payload[51].

## Response

The receiver of a PUBLISH Packet MUST respond according to Table below Expected Publish Packet response as determined by the QoS in the PUBLISH Packet[50].

QoS Level	Expected Response
QoS 0	None
QoS 1	PUBACK Packet
QoS 2	PUBREC Packet

**Table 3.13:**Expected Publish Packet response[51].

## Actions

The Client uses a PUBLISH Packet to send an Application Message to the Server, for distribution to Clients with matching subscriptions.

The Server uses a PUBLISH Packet to send an Application Message to each Client which has a matching subscription.

### 4.3. SUBSCRIBE (Subscribe to topics)

The SUBSCRIBE Packet is sent from the Client to the Server to create one or more Subscriptions. Each Subscription registers a Client's interest in one or more Topics. The Server sends PUBLISH Packets to the Client in order to forward Application Messages that were published to Topics that match these Subscriptions. The SUBSCRIBE Packet also specifies (for each Subscription) the maximum QoS with which the Server can send Application Messages to the Client[50].

#### Fixed header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (8)				Reserved			
	1	0	0	0	0	0	1	0
byte 2	Remaining Length							

**Table 3.14:** Fixed Header of the SUBSCRIBE Packet[51].

Bits 3,2,1 and 0 of the fixed header of the SUBSCRIBE Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection.

### **Variable header**

The variable header contains a Packet Identifier

### **Payload**

The payload of a SUBSCRIBE Packet contains a list of Topic Filters indicating the Topics to which the Client wants to subscribe. The Topic Filters in a SUBSCRIBE packet payload MUST be UTF-8 encoded strings.

A Server SHOULD support Topic filters that contain the wildcard characters. If it chooses not to support topic filters that contain wildcard characters it MUST reject any Subscription request whose filter contains them.

Each filter is followed by a byte called the Requested QoS. This gives the maximum QoS level at which the Server can send Application Messages to the Client.

The payload of a SUBSCRIBE packet MUST contain at least one Topic Filter / QoS pair. A SUBSCRIBE packet with no payload is a protocol violation.

The requested maximum QoS field is encoded in the byte following each UTF-8 encoded topic name, and these Topic Filter / QoS pairs are packed contiguously[51].

### **Response**

When the Server receives a SUBSCRIBE Packet from a Client, the Server MUST respond with a SUBACK Packet.

The SUBACK Packet MUST have the same Packet Identifier as the SUBSCRIBE Packet that it is acknowledging[50].

## **4.4. DISCONNECT (Disconnect notification)**

The DISCONNECT Packet is the final Control Packet sent from the Client to the Server. It indicates that the Client is disconnecting cleanly.

### Fixed header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (14)				Reserved			
	1	1	1	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

**Table 3.15:**Fixed Header of the DISCONNECT Packet[50].

The Server MUST validate that reserved bits are set to zero and disconnect the Client if they are not zero.

### Variable header

The DISCONNECT Packet has no variable header.

### Payload

The DISCONNECT Packet has no payload.

### Response

After sending a DISCONNECT Packet the Client:

- MUST close the Network Connection.
- MUST NOT send any more Control Packets on that Network Connection.

## 5. Development environment used

To realize our system, we used TTCN-3 language, and some tools of development. We will describe our development environment in the following subsections:

### ➤ TITAN eclipse

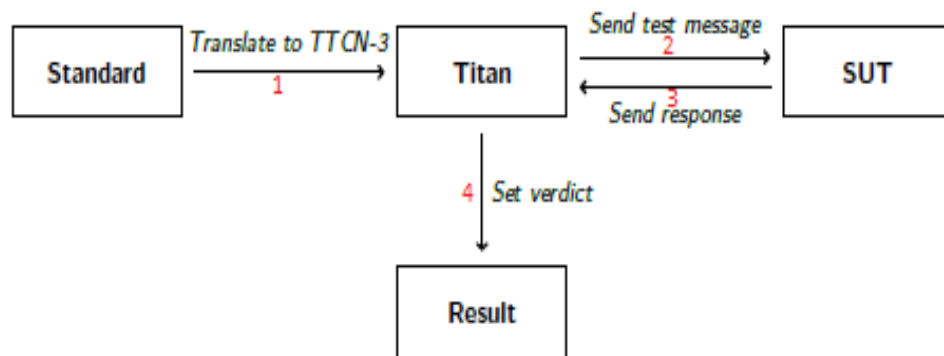
Eclipse Titan is a TTCN-3 compilation and execution environment with an Eclipse based IDE.

Titan consists of a core part, executing in a Unix/Linux-like environment and a set of

Eclipse plug-ins.[52]

In the figure 3.1 show the approach how it work:

- 1- Translate the standards of MQTT to TTCN-3 language, with eclipse titan.
- 2- Titan sends the test of the MQTT messages to the system under test (SUT).
- 3- SUT response to the test message and send the response to titan.
- 4- When titan receives the response from SUT, it will send a verdict (pass/fail/over time) to show the result.



**Figure 3.1:**Simplified model of the approach[53]

## 6. Deriving the tests

As we mentioned earlier, the goal of our system is to do a formal verification of the implementation of the MQTT using TTCN-3. The implementation of the system will be done by the TTCN-3 language through Eclipse Titan.

We have followed the following steps:

### ➤ Specify the standards of MQTT packet and message by TTCN-3

The first step in our implementation is specified the standards of (MQTT\_v3\_1\_1\_Message), as we know MQTT is a binary based protocol were the control elements are binary bytes and not text strings, for that we declared the types: octetstring and bitstring.

```

type octetstring OCT0_65535 length(0..65535) with {variant "" };

type octetstring OCT1_MSB length(1) with { variant "FIELDLENGTH(1),BITORDER(msb)" };
type octetstring OCT2_MSB length(2) with { variant "FIELDLENGTH(2),BITORDER(lsb)" };////!!
type octetstring OCT4_MSB length(4) with { variant "FIELDLENGTH(4),BITORDER(msb)" };

type universal charstring UCHAR0_65535 length(0..65535) with {variant "" };
type bitstring BIT1n length(1);
type bitstring BIT4n length(4) with {variant "" };
type integer INTO_255 (0..255) with {variant "" };
type integer INT_BIT16_MSB (0..65535) with { variant "FIELDLENGTH(16),BITORDER(msb)" };

```

**Figure 3.2:** Declared the types.

We need to defined the QoS (we choose enumerate type for it) , the UTF-8 encoding (because the topic names, Client ID, User names and Passwords are encoded as UTF-8 strings) , the header and MQTT identifier.

```

type enumerated QoS
{
  AT_MOST_ONCE_DELIVERY(0),
  AT_LEAST_ONCE_DELIVERY(1),
  EXACTLY_ONE_DELIVERY(2),
  RESERVED(3)
} with {variant "FIELDLENGTH(2)" }

type record Header
{
  BIT4n          packetType,
  BIT4n          flags,
  OCT4_MSB       remLength
} with {variant "FIELDORDER(msb)" }

type record MQTT_v3_1_1_Identifier
{
  Header          header,
  INT_BIT16_MSB  packet_identifier
}
with {variant "FIELDORDER(msb)" ;
      variant (header.remLength) "LENGTHTO(packet_identifier)" ;
      variant (packet_identifier) "BITORDERINFIELD(msb)"
}

type record UTF8EncodedString
{
  INT_BIT16_MSB  stringLength,
  UCHAR0_65535  stringItem
} with {variant "FIELDORDER(msb)"
      ; variant (stringLength) "BITORDERINFIELD(msb)"
      ; variant (stringLength) "LENGTHTO(stringItem)"
}

type record OctStringWithLength
{
  INT_BIT16_MSB  stringLength,
  OCT0_65535     stringItem
} with {variant "FIELDORDER(msb)"
      ; variant (stringLength) "BITORDERINFIELD(msb)"
      ; variant (stringLength) "LENGTHTO(stringItem)"
}

type record MQTT_v3_1_1_Empty
{
  Header          header
}with {variant "" }

```

**Figure 3.3:** Defined the imports of the message structure.

Now we move to specify the connect message as show in figure bellow:

```
type record MQTT_v3_1_1_ConnectFlags
{
  BIT1n          user_name_flag,
  BIT1n          password_flag,
  BIT1n          will_retain,
  QoS            will_qos,
  BIT1n          will_flag,
  BIT1n          clean_session,
  BIT1n          reserved
} with (variant "FIELDORDER(msb)"

)

type record MQTT_v3_1_1_ConnectPayload
{
  UTF8EncodedString      client_identifier,
  UTF8EncodedString      will_topic optional,
  OctStringWithLength    will_message optional,
  UTF8EncodedString      user_name optional,
  OctStringWithLength    password optional
} with (variant "FIELDORDER(msb)"

)

type record MQTT_v3_1_1_Connect
{
  Header              header,
  INT_BIT16_MSB       nameLength,
  UCHARO_65535        name,
  INTO_255            protocol_level,
  MQTT_v3_1_1_ConnectFlags  flags,
  INT_BIT16_MSB       keep_alive,
  MQTT_v3_1_1_ConnectPayload  payload
} with (variant "FIELDORDER(msb)" ;
variant (nameLength) "BITORDERINFIELD(msb)" ;
variant (nameLength) "LENGTHTO(name)" ;

)

```

**Figure 3.4:** An Example of CONNECT Message with TTCN-3.

The description of each different type of message is not the same even though they share the same control packet structure.

The data of publish and subscribe message is different than the data of connect message.

```

type record MQTT_v3_1_1_PublishHeader
{
  BIT4n          packetType,
  BIT1n          dup_flag,
  QoS            qos_level,
  BIT1n          retain_flag,
  OCT4_MSB       remLength
}
with (variant "FIELDORDER(msb)"
)
)
type record MQTT_v3_1_1_Publish
{
  MQTT_v3_1_1_PublishHeader  header,
  INT_BIT16_MSB              nameLength,
  UCHAR0_65535               topic_name,
  INT_BIT16_MSB               packet_identifier optional,
  octetstring                 payload
} with (
  variant "FIELDORDER(msb)" ;
  variant (packet_identifier) "BITORDERINFIELD(msb)" ;
  variant (nameLength) "BITORDERINFIELD(msb)" ;
  variant (nameLength) "LENGTHTO(topic_name)";
  // variant (header.remLength) "LENGTHTO(nameLength,topic_name,packet_identifier,payload)";
)

```

**Figure 3.5:** An Example of PUBLISH Message with TTCN-3.

```

type record MQTT_v3_1_1_SubscribePayload
{
  INT_BIT16_MSB          filterLength,
  UCHAR0_65535          topic_filter,
  QoS                    requested_qos
}with (variant "FIELDORDER(msb)";
  variant (filterLength) "BITORDERINFIELD(msb)" ;
  variant (filterLength) "LENGTHTO(topic_filter)";
  variant (requested_qos) "BITORDERINFIELD(msb)" ;
  variant (requested_qos) "FIELDLENGTH(8)" ;
  variant (requested_qos) "BITORDER(msb)" ;
)

type record of MQTT_v3_1_1_SubscribePayload MQTT_v3_1_1_SubscribePayloadList with (variant "");

type record MQTT_v3_1_1_Subscribe
{
  Header                header,
  INT_BIT16_MSB         packet_identifier ,
  MQTT_v3_1_1_SubscribePayloadList  payload
} with (variant "FIELDORDER(msb)"
  variant (packet_identifier) "BITORDERINFIELD(msb)"
)

type record of INTO 255 IntegerList with (variant "");

```

**Figure 3.6:** An Example of SUBSCRIBE Message with TTCN-3.

- Create the union type for the MQTT message



After we specify the messages of MQTT, we create the union type which collect all the 14 message and create another union type (MQTT\_v3\_1\_1\_ReqResp) for the 14 types of valid MQTT messages.

Each possible MQTT message falls within the scope of the TTCN-3 unions defined in figure 3.7. More specifically, the message can either be treated as an octet (byte) string, or as a structure, which on its own is one of the 14 types of valid MQTT messages.

```

type union MQTT_v3_1_1_Message
{
  MQTT_v3_1_1_ReqResp  msg, octetstring  raw_message
}

type union MQTT_v3_1_1_ReqResp
{
  MQTT_v3_1_1_Connect    connect_msg,
  MQTT_v3_1_1_Connack   connack,
  MQTT_v3_1_1_Publish   publish,
  MQTT_v3_1_1_Identifier puback,
  MQTT_v3_1_1_Identifier pubrec,
  MQTT_v3_1_1_Identifier pubrel,
  MQTT_v3_1_1_Identifier pubcomp,
  MQTT_v3_1_1_Subscribe subscribe,
  MQTT_v3_1_1_Suback    suback,
  MQTT_v3_1_1_Unsubscribe unsubscribe,
  MQTT_v3_1_1_Identifier unsuback,
  MQTT_v3_1_1_Empty     pingreq,
  MQTT_v3_1_1_Empty     pingresp,
  MQTT_v3_1_1_Empty     disconnect_msg
}

```

**Figure 3.7:** TTCN union type of MQTT messages

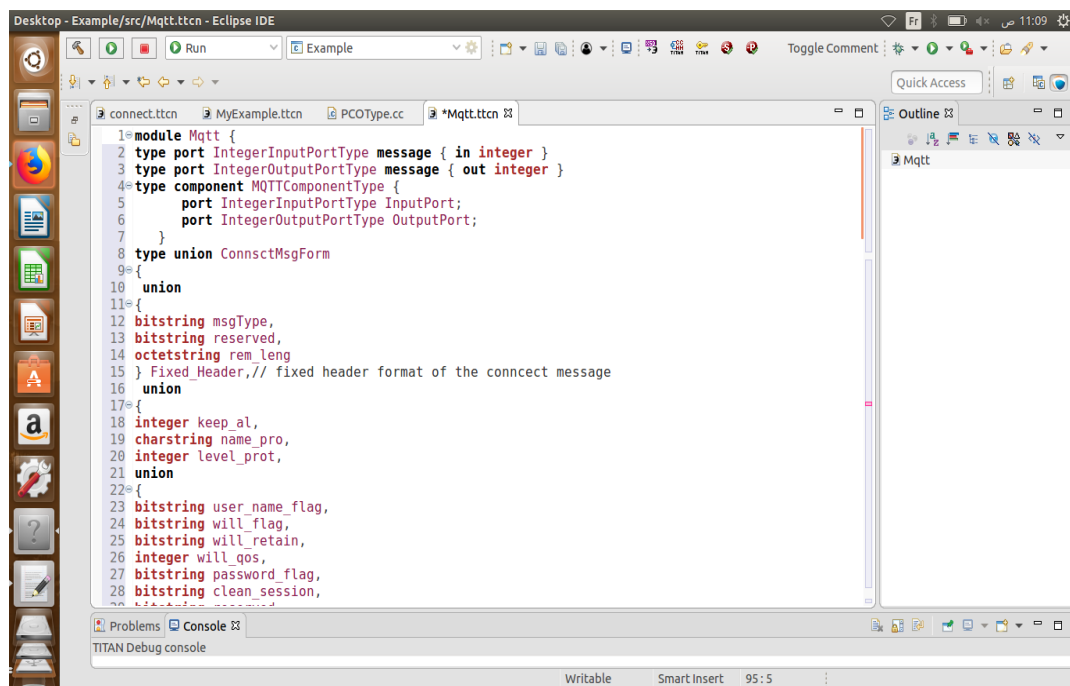
### ➤ Port type

Ports facilitate communication between test components and between test components and the test system interface. In our implementation we used port type for the communication between the TTCN-3 code and the implementation of MQTT.

We introduce two port types: IntegerInputPortType is the type of a port that acts as input for integers (we will represent sendingpacket request as integers). CharstringOutputPortType is the type of a port that serves to output integer (we will represent a receiving packet).

The component type defines which ports are associated with a component, the port names in a component type definition are local to that component type.

MQTTComponentType is the type of the MQTT message. aMQTT message has two ports: one to send packet message (integer) and one to receive packet message (integer).



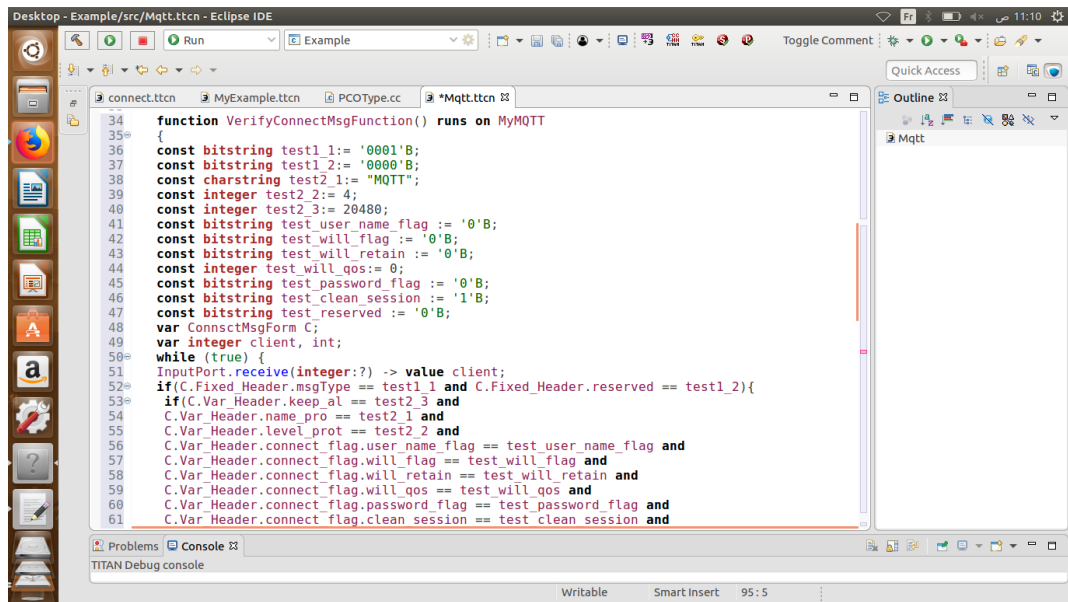
```
1 module Mqtt {
2   type port IntegerInputPortType message { in integer }
3   type port IntegerOutputPortType message { out integer }
4   type component MQTTComponentType {
5     port IntegerInputPortType InputPort;
6     port IntegerOutputPortType OutputPort;
7   }
8   type union ConnsectMsgForm
9   {
10    union
11    {
12      bitstring msgType,
13      bitstring reserved,
14      octetstring rem_leng
15    } Fixed_Header, // fixed header format of the connect message
16    union
17    {
18      integer keep_al,
19      charstring name_pro,
20      integer level_prot,
21      union
22      {
23        bitstring user name flag,
24        bitstring will_flag,
25        bitstring will retain,
26        integer will_qos,
27        bitstring password_flag,
28        bitstring clean_session,
29      }
30    }
31  }
32 }
```

**Figure 3.8:** Describe figure of component type and port type in our system

➤ **Declared function**

1) Function VerifyConnectMsgFunction()

It runs on a component of type MyMQTT and therefore has access to the ports of a MQTT message. In this function we put the correct message packet and compared with the message packet which we will test (MQTT implementation already exist).



**Figure 3.9:** Function to verify the standards of CONNECT mqtt message

In an infinite loop the MQTT message performs the following steps:

```

while (true) {
InputPort.receive(integer:?) ->value client;
//We test each byte using if else statement.
...
int:= 0;
OutputPort.send(int);
}
}else{
int:= 1;
OutputPort.send(int);
}
};

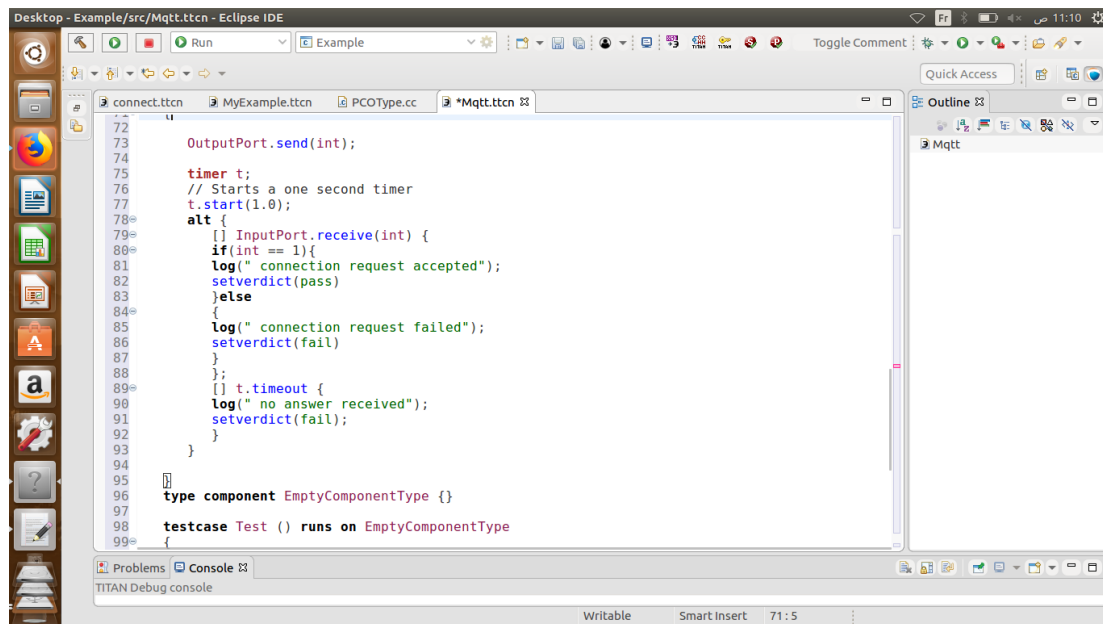
```

Receives an arbitrary integer from the port InputPort its value is redirected to the variable client to check if the client sent a correct message format of the CONNECT message

If the client send CONNECT message correctly, the variable int (integer) will take the value 0 else it will the value 1 that means the client not respect the standards of the message.

## 2) functionTestFunction():

The alt construct, similar to the switch/case construct in some programming languages. In the case of TTCN-3, it waits for one of the specified events to happen - either a response is received, the session is disconnected or a timer expires.



**Figure 3.10:** Test Function using alt statement of the message

For simplicity, the code as signing the int variable to the value of the Return Code in the CONNACK message is skipped.

The test verdict to pass in the component is means that is everything is ok.

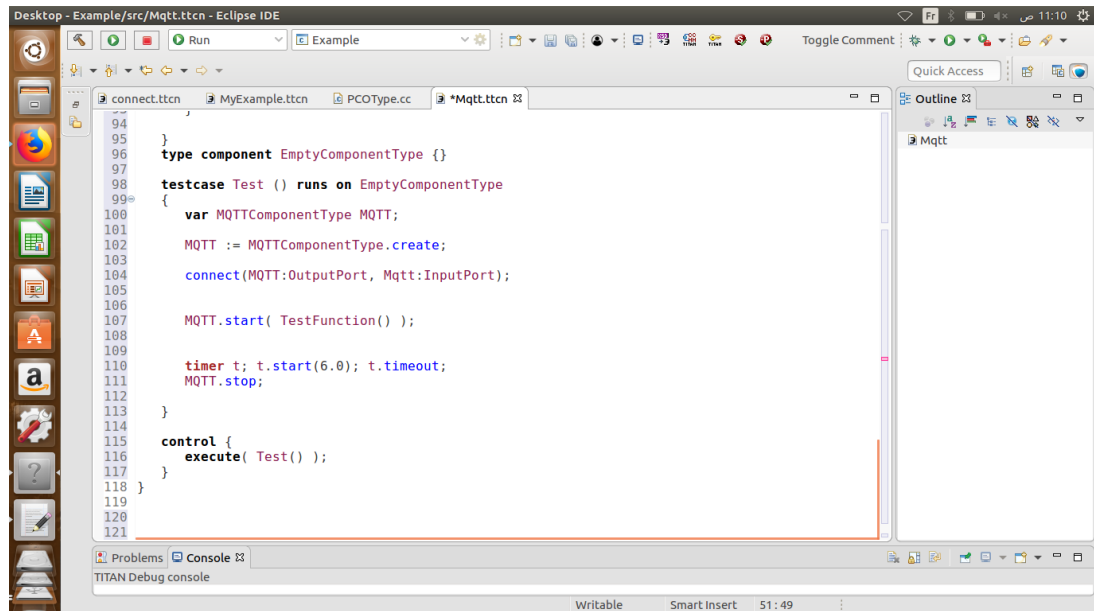
We declared a timer variable t starts a one second timer (Timer t). if t pass than 1 second it will fall in case of time is over as showing bellow:

```
[] t.timeout {
    log(" no answer received");
    setverdict(fail);
}
```

## ➤ Test cases

In TTCN-3, test cases are a special kind of function. Test cases define the behaviour, which have to be executed to check whether the SUT passes a test or not

In the test case definition we create, connect and start the components:



**Figure 3.11:** Test case of our system

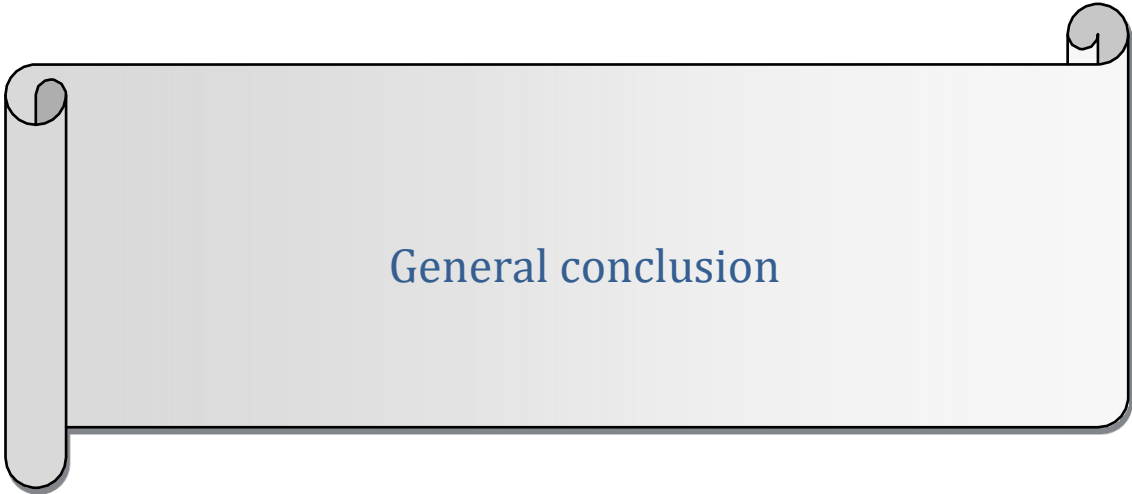
- Creates a component of type MQTTComponentType :  
MQTT :=MQTTComponentType.create;
- Connects the output port and with the input port of the MQTT message:  
connect(MQTT:OutputPort, Mqtt:InputPort);
- Starts the MQTT component and defines TestFunction() as its behavior:  
MQTT.start(TestFunction() );
- Because the exchanging message is in an infinite loop, we wait 6.0 seconds and shut it down:

timer t; t.start(6.0); t.timeout;

MQTT.stop;

## **7. Conclusion**

The purpose of the implementation chapter is to present the different practical aspects of our system. Despite the difficulties we encountered in connecting the "Eclipse Titan" environment and the "TTCN-3", which was a new language for us, but we provided appropriate solutions for achieve our goal.



General conclusion

## General conclusion

The Internet of Things (IoT) is defined as a paradigm in which objects equipped with sensors, actuators, and processors communicate with each other to serve a meaningful purpose.

In order to achieve the full potential of IoT, several software components have to be (re)engineered with adequate quality of service levels taking into consideration the IoT context constraints. For example, several communication protocols already exist and have been designed exclusively for small devices than previous protocols. One of these protocols is the message-queue telemetry transport (MQTT), which is designed to work on any quality of network and where the largest part of the processing is done by the server, not by the devices, the things, which can and should remain fairly small and simple. It's have different implementations.

This kind of implementations of networking protocol stacks is in need of thorough testing in order to ensure not only its security, also its conformance.

For that we use a new methods to formally assess the implementation of a communication protocol of MQTT adhere to the standard, this methods is testing by TTCN-3.

In our research:

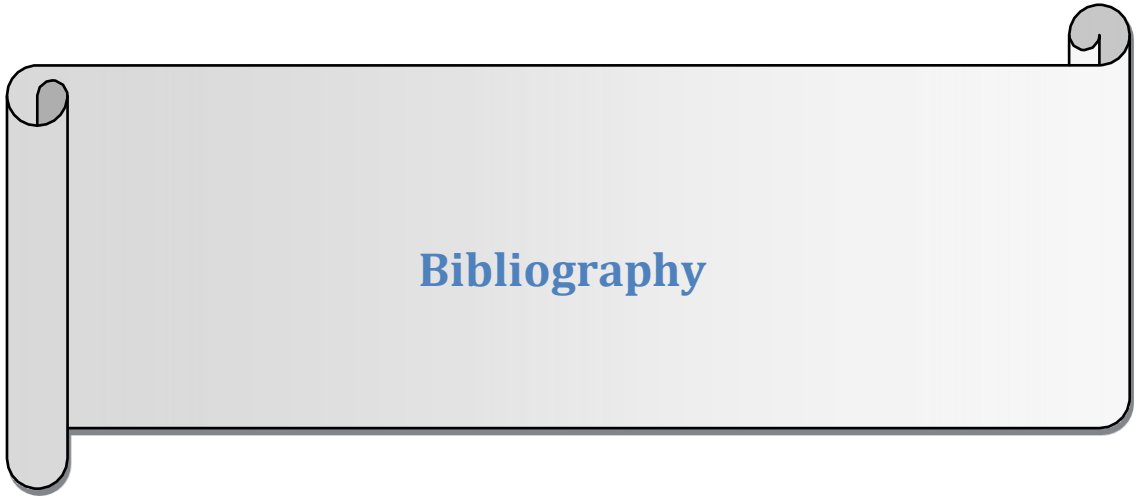
- ✓ We identified several possible formal methods which were shown to be suitable for testing of communication protocols (MQTT).
- ✓ We specify the details of the protocol MQTT from the structure control packet of each message packet to facilitate programming with TTCN-3 which has the ability not only to describe the expected behaviour of a given protocol, but also to execute the defined tests.
- ✓ We discussed the standards of the MQTT messages and how to interpret it into TTCN-3 Modules with examples to test the power of our system.
- ✓ We use as possible the cases of test to give our system more reliable.
- ✓ We don't verify all the MQTT messages, we just verify: CONNECT, PUBLISH and SUBSCRIBE command message.

We can cite some perspectives for researchers who want to use our work where they can improve our verification by implementing with another programming language or



to complete our working which we set it in the first of our research but we can't make it because we fall in the difficulties.

Another perspective to move to a very recent and very interesting area is any other protocol governing data exchange of the Internet of Things can be tested Depending on what we explained. The modeled message exchange can be mapped into a TTCN-3 test case and fired against a real system.



## Bibliography

Thanks.....	1
Abstract.....	
Glossary.....	
List of Figure.....	
List of Table.....	
<b>« General introduction »</b>	
General introduction.....	1
<b>« Chapter 1: Internet of things »</b>	
1. Introduction.....	3
2. Internet of things (IoT).....	3
2.1. Characteristics of the IoT.....	5
2.2. IoT Application.....	6
2.3. IoT Standards and Protocols.....	7
2.3.1. Application Protocols.....	9
a. Coap.....	9
b. XMPP.....	10
c. AMQP.....	10
d. DDS.....	10
e. MQTT.....	11
3.2.2. Other protocols.....	13
3. Advantages and Disadvantages of IoT.....	17
4. IoT challenges.....	19
5. Solutions for IoT.....	20
6. Conclusion.....	20
<b>« Chapter 2: Software Testing »</b>	
1. Introduction.....	21
2. Software testing.....	21
3. Objectives of testing.....	21
4. Verification and Validation.....	21
5. Levels of testing.....	22
5.1. Unit testing.....	22
5.2. Integration testing.....	22
5.3. System testing.....	22
5.4. Acceptance testing.....	23
6. Categories of testing types.....	23

6.1. Black box testing.....	23
6.2. White box testing.....	24
6.3. Gray box testing.....	24
6.4. Non-Functional testing.....	25
6.4.1. Performance testing.....	25
6.4.2. Security testing.....	27
6.4.3. Recovery testing.....	27
7. Limitations of testing.....	28
8. TTCN-3 language.....	28
8.1. Key TTCN-3 language features.....	28
8.2. Language basics.....	29
8.3. The concepts of TTCN-3.....	30
8.3.1. Subtype.....	30
8.3.2. Components.....	30
8.3.3. Test case.....	31
8.3.4. Templates.....	31
8.3.5. Alt statement.....	32
9. Conclusion.....	33
<b>« Chapter 3: Conception and Implementation of system »</b>	
1. Introduction.....	34
2. Structure of an MQTT control packet.....	34
3. MQTT message format.....	35
3.1. Fixed header.....	35
3.2. Variable header.....	39
3.3. Payload.....	40
4. Some of command message.....	41
4.1. CONNECT.....	41
4.2. PUBLISH.....	42
4.3. SUBSCRIBE.....	44
4.4. DISCONNECT.....	45
5. Development environment used.....	46
6. Deriving the test.....	47
7. Conclusion.....	54

**« General conclusion »**

General conclusion..... 55

**« Bibliography»**

Bibliography.....